NTNU
Norwegian University of
Science and Technology

Determination of Real-Time Positions of a Seagoing Vessel Based on Real-Time Video

Stig Hornang

Master of Science in Engineering Cybernetics
Submission date: June 2010
Supervisor: Thor Inge Fossen, ITK
Co-supervisor: Pål Jacob Nessjøen, National Oilwell Varco

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Study the geometry and dynamics of a typical offshore crane scenario and define the requirements of a real-time prototype system. Investigate the field of computer vision and find algorithms which are appropriate for the task and develop custom algorithms when needed. Select an appropriate platform and implement a prototype system which shows proof-of-concept. Analyze the accuracy of the system by comparing the measurements with the real positions.

Assignment given: 25. January 2010
Supervisor: Thor Inge Fossen, ITK

Abstract

This thesis investigates the possibility of determining the position of a seagoing vessel based solely on the video streams from multiple cameras. The problem is addressed in relation to offshore crane operations where there is relative movement between supply vessel and crane which is mounted on a rig or platform. The thesis starts by describing the offshore crane scenario and states the required properties of a prototype system. The video analysis starts by deriving an algorithm for finding the vessel in the video stream. Other algorithms for video tracking, finding corresponding points and estimating position and attitude of the vessel's cargo deck are presented. The system is implemented as a C++ application and is tested in a downscaled environment with a realistic, wave-simulated ship model. The measured positions are compared with the real positions and the accuracy of the signals are analyzed.

Preface

This thesis concludes my master's degree Engineering Cybernetics at the Norwegian University of Science and Technology. This thesis has been a challenging project which has involved theoretical work and lots of implementation and experimental testing which suits me well. I would first of all like to thank Pål Jacob Nessjøen from National Oilwell Varco for a very interesting project. He also provided invaluable help in building the experimental setup and was a great travel companion to the Offshore Technology Conference in Houston. I would also like to thank my co-students for contributing with advice and providing a social, but productive working environment. I would also like to thank the personnel at the institute's workshop for building the motion platform, camera mounts and calibration cube at such a short notice.

NTNU, Trondheim 26.06.2010
Stig Hornang

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter the motivation for developing a video camera based vessel tracking system for offshore crane operations is presented. The relevant previous work in the field of computer vision is briefly investigated. Finally the contribution which this thesis brings to the field of computer vision and the contribution of experimental data is presented.

1.1 Motivation

Offshore crane operations are challenging because the crane is often hoisting or lowering some object which is placed on a moving supply vessel. The crane and the vessel have relative movement due to waves. Extra caution is required from the crane operator in the phase where an object is lifted or put down on the vessel. Lack of awareness from the crane operator can damage both the object being lifted, the ship's cargo deck and the crane due to the dynamic forces which may be large. Bad weather is one of the main factors which limits the operational window of crane operations.

Many operations at sea require the knowledge of a vessels position and attitude. A dynamic positioning system utilizes measurements in at least three degrees of freedom. Such a system is able to keep a vessel stationary even in heavy weather. Most supply vessels have installed a dynamic positioning system and therefore already know its position and attitude. In crane operations two independent systems are interacting and information from on system is not available in real-time in the other system without special compatible software, hardware and wireless communication between the two systems. Considering the variety of commercial dynamic positioning systems installed in supply vessels today, a common communication platform is cumbersome and costly to install in every ship. Even if a communication platform is established, not all ships equipped with dynamic positioning systems have measure devices for enough degrees of freedom to know every position of its cargo deck.

A system which is able to measure a ships position and attitude based on real-time video from from cameras is therefore an interesting scheme to investigate further. The information

about the position and attitude can be used in the determination of when to execute a crane operation. Or ultimately it can be use in heave compensation of the crane actuators.

1.2 Previous Work

A system which base its measurements on video data rely heavily on the science behind computer vision. And most of the relevant previous work is therefore found within this discipline.

Two of the most central tasks in computer vision are object recognition and video tracking. In video tracking there are different ways of representing the target. The target representation depends on the tracking method. The most straightforward approach is to represent the target object as a small image of the object and compare that with the current video frame by using two dimensional cross-correlation in the region of interest. The position with the largest correlation match defines the position of the object. The main problem with cross-correlation is that the spatial (pixel) representation of the object may change due to scale and rotation. (Comaniciu & Meer 1999) describes a method called mean-shift tracking which uses the probability distribution of the object as target representation. This removes the spatial dependence. For color image this is typically represented by a multi-dimensional histogram along each of the axes of the color space.

A common approach in object recognition is to use edge detection to capture the shape of an object. The most famous edge detection algorithm is the Canny edge detector (Canny 1986). The captured edges are compared with a previously stored template. Simpler geometric properties like straight lines and circles can be found by using the Hough-transform (Duda & Hart 1972) to extract the parameters of the found objects.

Working with 3-D computer vision requires a geometrical camera model which defines the relation between 3-D coordinates of the captured scene and the position in the camera image. Most cameras can be described by the pin-hole model which also will be presented in this thesis. Camera calibration is the task of finding the camera parameters of a particular camera. The simplest calibration methods are based on direct linear transformation and ends up solving a linear system. More advanced calibration techniques exists like (Zhang 1999) which estimates both the geometrical camera parameters as well as lens distortion by viewing a plane from different unknown orientations. The problem of finding a position in space from the pixel coordinates is an undetermined problem, but by using two or more cameras the problem can be solved if correspondent points in multiple cameras are found.

The main problem in 3-D computer vision is to find correspondent points in two or more cameras. This can be solved by arranging the cameras in a frontal parallel setup and performing stereo rectification. This reduces the correspondence search to horizontal line (Bradski & Kaehler 2008). Alternatives are to constrain the search by utilizing known information about the object to be tracked. Since stereo arranged cameras capture the scene from approximately the same angle methods like cross-correlation can be used to find corresponding points.

Estimating both position and attitude of a rigid body requires the measurement of at least three 3-D points. Estimating the translation and rotation of rigid body based on multiple

3-D points is often called 3-D rotation fitting. (Eggert et al. 1997) compares four major algorithms for estimating 3-D rigid body transformation. These algorithms assume that there are no outliers in the data sets. A method which is targeted at data sets with outliers is the RANSAC algorithm (Fischler & Bolles 1981).

1.3 Contribution

This thesis investigate the applicability of computer vision algorithms in vessel position and attitude measurement. Both new and existing algorithms are presented. A special algorithm which finds the port and starboard edges of supply vessel is derived. The algorithm finds the lines in an image which best matches the supply vessel edges. The Canny edge detector and the Hough-transform is used as the first steps in this algorithm. A simple but effective correspondence algorithm which uses simple geometric assumptions about the captured scene is derived without the need for stereo rectification, but only basic camera calibration.

The 6 DOF motion estimation is based on previous results by (Kanatani 1994) and (Arun et al. 1987), but the algorithms are extended to handle dynamic point sets such that continuous measurements are provided when the tracking properties of the scene changes.

A prototype system is implemented in C++ using multiple open-source frameworks and libraries. The implemented application features a user interface which provides easy camera calibration, source selection and data export and various visualization features. Many implementation specific aspects are addressed in this thesis and typical performance bottlenecks when dealing with high-definition video are explained and taken care of. The prototype system is tested in a downscaled environment with a wave simulated ship model and the quality of the measurements are discussed.

1.4 Abbreviations

CCS	Camera Coordinate System
WCS	World Coordinate System
ROI	Region of Interest
UI	User Interface
DOF	Degrees of Freedom
FOV	Field of View
NED	North East Down
NOV	National Oilwell Varco
OTC	Offshore Technology Conference

Chapter 2

System Specification

The type of system to be specified in this chapter is commonly called a machine vision system which is an automated system. The automated property is what differs machine vision from human vision which requires actions taken by a human. In contrast to a human vision which is *very* versatile, this chapter specifies a vision system to be used in a limited environment. With "limited environment" means for instance what scenarios the system is usable in, what kind of vessels that can be measured and how much user interaction the system requires. A vision system which handles "everything" is nearly impossible to create. The specifications in this chapter will narrow the usable area for the vision system, but every aspect can greatly reduce the complexity of the system.

2.1 Problem and Scene Description

This specification's starting point is the scene outlined in the 1.1. There are two floating structures, the supply vessel and the larger platform which has a crane located on it. The position, ultimately of any point, on the supply vessel's cargo deck relative to the platform or rig should be measured. This information must be based only on the video information from two or more cameras which are located on the crane. The position data must be available in real-time in the drivers cabin at the crane.

Camera positioning

The crane base is located approximately 50 meters above sea level. The crane is a boom crane where the crane has three main movements. The crane base can rotate, the angle of the boom can change and the hook can be lowered and hoisted with a winch. At least two cameras will be mounted on the crane base and will therefore swing along with it. It is important that the relative position and view angle between the cameras are fixed at all times. The cameras position and field of view must not change due to boom angle change. The separation between the cameras is a compromise between accuracy and the difficulty of finding correspondence

Figure 2.1: Typical supply vessel and platform.

as will be described in 4.4. The field of view should capture all the positions at sea level where the crane can or usually position its hook. This means the complete deck area of a supple vessel which is only a few meters above the sea level. The cameras field of view (FOV) should overlap as much as possible. Areas that is only covered by one camera is useless for 3-D position tracking.

Marine vessel

The vessel does *not* need any special markers to be able to be tracked. It is therefore even more important to specify the characteristics of the marine vessels that frequently visits a platform or rig. Marine vessels which are involved in lifting operation offshore are mostly supply vessels. Supply vessel have a significantly large and rectangle shaped deck which covers at least 50% of the aft area of the vessel. The port and starboard edges are parallel along the deck. The aft edge is undefined, but sometimes it is as straight and sometimes it is rounded if the vessel is also an anchor handling vessel.

Scene dynamics

The size and weight of involved structures put a limit on the magnitude of the acceleration which can be expected since the forces are involved are limited. A large supply vessel has much

slower movement than a small boat. Generally when working with large marine structures a sampling rate of 10 Hz is sufficient for measurement and feedback control. This is a very low limit compared two industrial robot vision where the velocities and accelerations are a lot higher and therefore require much higher sample rate. The low sample rate is especially an advantage related to machine vision where the sampling rate cannot easily be increased due to limited computer power.

The rotational movement of a marine vessel is called yaw, pitch and roll (SNAME 1950). Where yaw is the vessels heading and roll and pitch is the longitudinal and transversal angles of the vessel respectively. The maximum roll and pitch angles which can be expected is of course dependent on the weather conditions, but angles larger than 15° not considered as relevant. Simulations later in this thesis shows that at least 6 meter significant wave height is necessary to achieve these angles with a supply vessel that is not positioned in an optimal way. The relatively small roll and pitch angles are a great advantage for this vision system. Tracking without rotation is easier to accomplish and the small rotations ensures that approximately the same area is visible in the video image at all times.

2.2 Measurement Quality

The quality of the measurements put a limit on the usefulness of the measurement signals. For use in feedback control the signal must be filtered appropriately, must have little or no lag and have sufficient accuracy. The measurement signal for this system should be of such quality that only limited improvement of the system would make the measurement signal usable in a heave compensation feedback loop. This thesis does not cover closing the feedback loop or cover the dynamics of a crane, but will focus on creating a measurement system based on video which at later stage could be used in a feedback control.

Nonetheless, here follows some requirements which are a good starting point of the measurement system:

- The sample rate should be at least 10 Hz.
- The accuacy should be less then ± 0.1 meters.
- The position of any point on the cargo deck should be available.
- The position measurement must be available and continuous for the complete time of one lifting operation (one container lift).
- The measurements may be offset with a constant value, but should be correctable by operator intervention.
- A limited part of the cargo deck may be occluded by swinging cargo without the loss of measurement signal.

2.3 Visualization

The measurement data should be visualized in such way that it is clear that the signal represents the movement of the vessel. Some visualization techniques are:

- Augmented reality.
- 3-D model.
- Graphs

Augmented reality means that the original video image is superimposed with a synthetic image which follows the movement of the ship. It is not required to create a fully functional operators UI, but the UI should show the principle of the above techniques. The rest of the UI can be created as needed for algorithm testing and debugging.

2.4 Testing Environment

The system should be tested in an environment which is similar to the real environment. A scenario can be scaled down with any factor and will still be visually identical as the real scene, but there are factors which cannot be scaled. One of them is the gravity acceleration g. Therefore any natural movement in a small scale environment will not appear realistically like waves on water. A way of canceling the scale-down effects is to increase the speed of time which in turn requires higher sample rate. Sample rate is limited by the vision system and cannot easily be increased. This concludes that testing the system using the natural movements of a scaled down system is not possible in real-time.

To create a realistic small scale environment the vessel movement must be simulated and performed by actuators. The details of the simulated environment is covered in detail in 8.2.

2.5 Hardware

The computational demands of machine vision is rather large. If limitations on size, power consumption and ruggedness of the processing device can be removed, at least for a prototype system, the processing capabilities for a given price can be maximized. The consumer PC industry is the main driving force for cheap and high performance computing power today and it is definitely in this market one will find the highest performance per price. Additionally the software development on a regular PC platform is very easy compared to embedded system development with dedicated processing chips. These facts points out that a desktop PC is the best development platform and target processing unit.

The specific type of cameras that will be used is be treated in 8.1.

Chapter 3

Camera theory

The geometrical properties of a digital camera are needed to define the relation between position in space and position in the captured image. The simple and well-known pinhole camera model will be presented. This model will further be used to form a calibration method and a triangulation method for recovering real world coordinates from pixel coordinates from two or more cameras. This triangulation is the basis for finding the 3-D position of a point in space using multiple cameras.

3.1 Coordinate Systems

Two coordinate systems are defined. One is fixed to the camera and is called the camera coordinate system (CCS). The exact position and orientation of this coordinate system relative to the camera is defined later. The other coordinate system is the world coordinate system (WCS). The position and orientation of the WCS can be chosen arbitrarily, but once chosen it stays fixed relative to the CCS.

3.2 The Pinhole Camera

Ordinary video cameras use lenses to focus rays of light onto a light sensing array of semiconductors which translates light intensities into a digital representation of the light intensities. The geometry of regular cameras can be approximated by a model known as the pinhole camera model. A pin hole camera is a rectangular box which projects an image of the outside world on one of the walls by letting light through a small hole in the opposite wall. See figure 3.1. The small pinhole puts a constraint on the incoming rays such that light ray from an arbitrary point on the outside object will only illuminate the opposite wall at one position.

Using a pinhole camera instead of a camera with glass lenses results that every object will appear in focus independent of distance. Although this eliminates the focus problem, the

pinhole camera has a major drawback in light sensitivity. The light getting through to the inside of the camera is very limited. A real camera uses a a convex lens to perform the pin hole effect. By using lenses the light sensitivity of the camera is improved. The pinhole model will generally model a well focused imaging system (Ma et al. 2003). Because the pinhole model disregards the focusing needed in a real camera it is assumed that the focus is always correct.

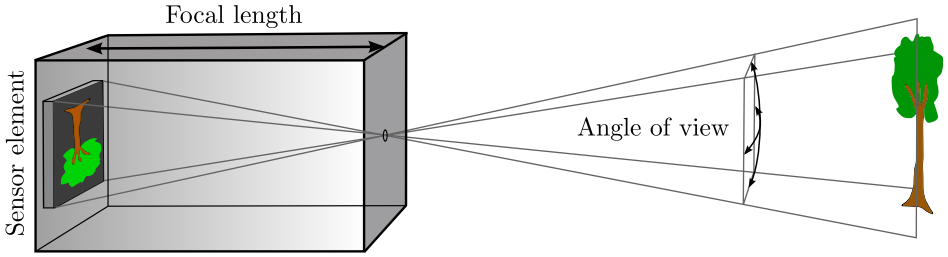


Figure 3.1: The pinhole camera.

3.3 Intrinsic model

The intrinsic model is related to the geometry between points in the real world and points in the image plane. The image plane is the position where the sensor chip is placed. In a real pinhole camera the image plane is placed behind the pinhole. In the model derived here it will be considered to be in front of the reference point. This is will just be a non flipped version of the image that would be formed by a regular pin hole model.

A right handed coordinate system for the camera position and orientation will be defined. This will later on be referred to as the CCS (Camera Coordinate System). The origin of this coordinate system is placed where the pinhole originally would be positioned. This is also the position where the lens of a real camera would be positioned. The z-axis of this coordinate system is defined according to the view direction of the camera and it is also called the *optical axis*. The x-axis points parallel to the horizontal edge of the image plane in the direction of increasing pixel index. The y-axis points parallel to the vertical edge of the image plane and in the direction of increasing pixel index. The most common way of indexing pixels is from the top left to the right bottom when viewed in the positive z-direction. This convention will be used throughout this derivation.

Let (x_c, y_c, z_c) be the coordinates of a point p_w. By looking at figure 3.2 an arbitrary point on the straight line from the origin to the point p_w as a function of the z coordinate is $l(z) = \left(x_c\frac{z}{z_c},\ \ y_c\frac{z}{z_c}\right)$. The image plane is the plane where the a real camera would capture its image. The interesting point is the x and y-coordinate of the crossing of the line with the image plane.

The distance from the origin to the image plane is known as the focal length f. The coordinates in the image plane is found from $l(f) = \left(x_c\frac{f}{z_c},\ \ y_c\frac{f}{z_c}\right)$. The basic equations for mapping a real world coordinate expressed in the CCS x_c, y_c, z_c to the image plane where subscript p

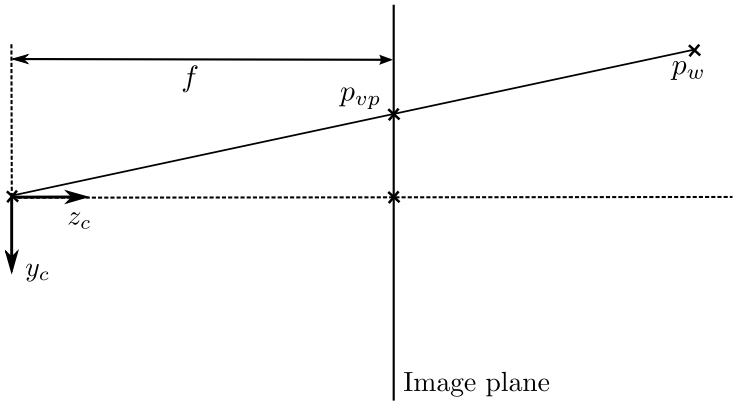


Figure 3.2: The CCS and the image plane of the camera.

denotes the coordinates in the image plane therefore becomes

$$x_p = f\frac{x_c}{z_c} \tag{3.1}$$

$$y_p = f\frac{y_c}{z_c} \tag{3.2}$$

It is convenient to express this relation as a matrix computation, but (3.1) and (3.2) are not linear mappings. At first the mapping can be defined as the linear operation with f and the division by z_c as a separate operation.

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \frac{1}{z_c}\begin{bmatrix} f & 0 \\ 0 & f \end{bmatrix}\begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{3.3}$$

Homogeneous coordinates are introduced by adding the z_c-coordinate as the third coordinate. To obtain the real coordinates of the image plane the division has to be done separately. The following is defined to describe the homogeneous coordinates

$$X = z_c \cdot x_p \tag{3.4}$$

$$Y = z_c \cdot y_p \tag{3.5}$$

$$\begin{bmatrix} X \\ Y \\ z_c \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{3.6}$$

Next, the mapping from the image plane to the actual digital image plane will be defined.

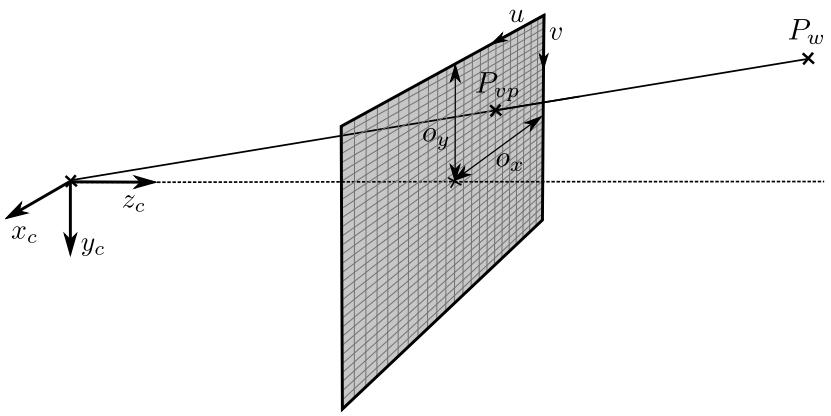


Figure 3.3: Position of the image plane relative to the CCS.

A digital camera has a light sensitive chip which is an array of rectangular sensor elements which is connected to each pixel in the resulting image. The term resolution defines how many pixels there are on defined length of the image plane. This is physical measure which can be read from the specifications of camera. The pixels need not be quadratic and as a consequence the resolution need not be equal in the x and y direction.

To model the resolution two scalars s_x and s_y are defined for the x_c and y_c direction respectively. These scaling factors will be positive according to the defining the x_c and y_c-axis in 3.3. Next, the position $(x_p, y_p) = (0, 0)$ need not to correspond to the pixel with the index $(0, 0)$. In fact, this is seldom the case as the z-axis usually intersects the image plane somewhere in the middle of the pixel matrix. This point is often called the principal point (Ma et al. 2003). The offset is modeled by to scalars o_x and o_y for the x_c and y_c-axis respectively. By using homogeneous coordinates both the scaling s_x, s_y and the offset (o_x, o_y) can be expressed with one transformation matrix as follows

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} \tag{3.7}$$

Where (u, v) are the pixel coordinates. A skew factor s_θ is also introduced as it makes the practical decomposition of the final camera matrix a lot easier. The skew factor is proportional to $\cot\theta$ where θ is the angle between the image axis x_c and y_c (Ma et al. 2003). The skew factor should be small for modern cameras.

By multiplying with z_c on both sides of (3.7) using (3.4) and defining

$$U = u \cdot z_c \tag{3.8}$$

$$V = v \cdot z_c \tag{3.9}$$

The relation becomes

$$\begin{bmatrix} U \\ V \\ z_c \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ z_c \end{bmatrix} \tag{3.10}$$

which can easily be multiplied with the perspective projection matrix

$$\begin{bmatrix} U \\ V \\ z_c \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{3.11}$$

$$\begin{bmatrix} U \\ V \\ z_c \end{bmatrix} = \begin{bmatrix} s_x f & s_\theta f & o_x & 0 \\ 0 & s_y f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{3.12}$$

The intrinsic transformation matrix is denoted $\mathbf{K}$. As seen in (3.12) there is no way to compute both focal length and resolution from experiments as they are factors of the same parameter.

3.4 Extrinsic model

The extrinsic parameters of the camera model are related to the position and orientation of the camera relative to the WCS. A matrix that both rotate and translate a point in the real world to the CCS, can be expressed in homogeneous coordinates as follows

$$\mathbf{p_c} = \begin{bmatrix} \mathbf{R_w^c} & \mathbf{T} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \mathbf{p_w} \tag{3.13}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.14}$$

Where $\mathbf{R}_w^c$ and $\mathbf{T}$ transforms a point $\mathbf{p}_w$ from be given in the WCS to be given in the CCS as $\mathbf{p}_c$. The translation vector is the position of the origin of the WCS given in the CCS.

By combining (3.12) and (3.14) the complete mapping from real world coordinates (x_w, y_w, z_w) to pixel coordinates is in homogeneous coordinates is expressed as follows

$$\begin{bmatrix} U \\ V \\ z_c \end{bmatrix} = \begin{bmatrix} s_x f r_{11} + s_\theta r_{21} + o_x r_{31} & s_x f r_{12} + s_\theta r_{22} + o_x r_{32} \\ s_y f r_{21} + o_y r_{31} & s_y f r_{22} + o_y r_{32} \\ r_{31} & r_{32} \end{bmatrix}$$
$$\begin{bmatrix} s_x f r_{13} + s_\theta r_{23} + o_x r_{33} & s_x f t_x + s_\theta t_y + o_x t_z \\ s_y f r_{23} + o_y r_{33} & s_y f t_y + o_y t_z \\ r_{33} & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.15}$$

(3.15) is commonly called the camera or calibration matrix. The only obvious requirement is that the lower most row vector of the three left most elements must be a unit vector because the elements originate directly from the rotation matrix of the extrinsic camera parameters.

It is also worth mentioning that the matrix can be scaled by any factor and still be valid. This is because the homogeneous pixel coordinates (U, V) is scaled with z_c.

3.5 Determination of Camera Parameters

In this section a simple method for determining the parameters of the calibration matrix will be presented. The method is input with a set of real world coordinates and corresponding image pixel coordinates and then solves an overdetermined system of linear equations for finding the parameters of the matrix.

A calibration rig is usually constructed to perform the practical calibration procedure. Such rigs have points where the position of each calibration point relative to a common reference point is known. The reference point can be chosen to be the origin of the WCS and is typically one of the calibration points on the rig.

The camera calibration matrix is written in a general form with twelve parameters

$$\begin{bmatrix} U \\ V \\ z_c \end{bmatrix} = \begin{bmatrix} c_{11} & c_{11} & c_{13} & c_{14} \\ c_{21} & c_{21} & c_{23} & c_{24} \\ c_{31} & c_{31} & c_{33} & c_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.16}$$

The expressions for the pixel coordinates is written

$$u = \frac{U}{z_c} = \frac{c_{11}x_w + c_{12}y_w + c_{13}z_w + c_{14}}{c_{31}x_w + c_{32}y_w + c_{33}z_w + c_{34}} \tag{3.17}$$
$$v = \frac{V}{z_c} = \frac{c_{21}x_w + c_{22}y_w + c_{23}z_w + c_{24}}{c_{31}x_w + c_{32}y_w + c_{33}z_w + c_{34}} \tag{3.18}$$

A vector $\mathbf{c}$ of the unknowns in the camera matrix is defined

$$\mathbf{c} = [c_{11}\ c_{12}\ c_{13}\ c_{14}\ c_{21}\ c_{22}\ c_{23}\ c_{24}\ c_{31}\ c_{32}\ c_{33}\ c_{34}]^{\mathrm{T}}$$

The equations (3.17) and (3.18) is rearranged so the unknowns can be separated in the previous vector

$$\begin{aligned} c_{11}x_w + c_{12}y_w + c_{13}z_w + c_{14} - uc_{31}x_w - uc_{32}y_w - uc_{33}z_w - uc_{34} = 0 \\ -c_{21}x_w + c_{22}y_w + c_{23}z_w + c_{24} - vc_{31}x_w - vc_{32}y_w - vc_{33}z_w - vc_{34} = 0 \end{aligned} \tag{3.19}$$

From (3.19) the following linear system of equations can established where (x_w^n, y_w^n, z_w^n) denotes physical calibration point n with corresponding pixel position (u^n, v^n).

$$\begin{bmatrix} x_w^1 & y_w^1 & z_w^1 & 1 & 0 & 0 & 0 & 0 & -u^1x_w^1 & -u^1y_w^1 - u^1z_w^1 & -u^1 \\ 0 & 0 & 0 & 0 & x_w^1 & y_w^1 & z_w^1 & 1 & -v^1x_w^1 & -v^1y_w^1 - v^1z_w^1 & -v^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_w^n & y_w^n & z_w^n & 1 & 0 & 0 & 0 & 0 & -u^nx_w^n & -u^ny_w^n - u^nz_w^n & -u^n \\ 0 & 0 & 0 & 0 & x_w^n & y_w^n & z_w^n & 1 & -v^nx_w^n & -v^ny_w^n - v^nz_w^n & -v^n \end{bmatrix} \begin{bmatrix} c_{11} \\ \vdots \\ c_{34} \end{bmatrix} = \mathbf{0} \tag{3.20}$$

which is a general homogeneous system

$$\mathbf{Vb} = \mathbf{0} \tag{3.21}$$

Assuming that all the equations are linearly independent at least $n \geq 6$ calibration points must be recorded to find all the twelve parameters. The solution of the homogeneous system (3.21) is called the *null space* where $\mathbf{c} = \mathbf{0}$ is the trivial solution which is uninteresting. Another basis for the null space is the eigenvector associated with the least eigenvalue of $\mathbf{V}^{\mathrm{T}}\mathbf{V}$ (Zhang 1999). Any scalar multiple of this basis is a solution.

Since the camera matrix can be scaled with any factor due to the homogeneous coordinates, the element c_{34} can be set to 1 in (3.19). This leads to little bit different system that can be solved by least squares instead of the least eigenvalue method described above. This can be advantageous in an implementation if e.g. only least square solving is available.

3.6 Recovering Real World Coordinates

Recovering real world coordinates is the process of finding the x, y and z position of a point in the real world when only the pixel coordinates from two or more cameras and the camera matrices are known. In this chapter it is assumed that two cameras are available, although more cameras may be used.

Each pixel in a camera defines an infinite long ray pointing out in space. Intuitively it is easy to understand that by using only one camera it is possible to tell which direction a point is positioned, but not how far away it is located. By having two cameras viewing the same point there are two rays pointing out in space. Ideally they are intersecting each other, but the fact that the two lines in 3D space do not need to intersect, hints that this is an overdetermined system which may not have a solution.

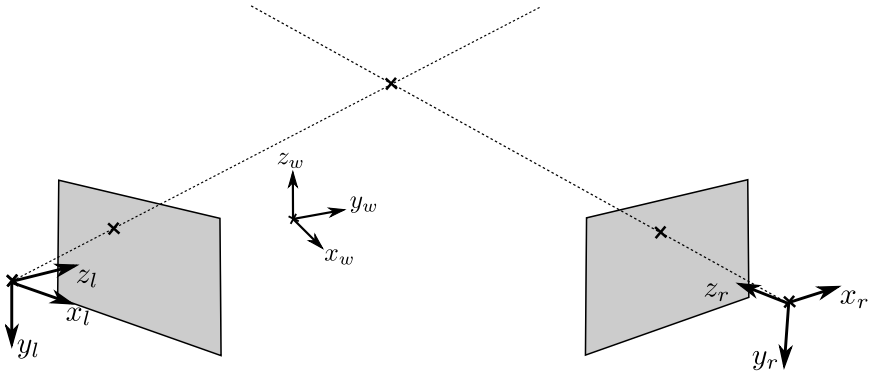


Figure 3.4: The geometry of two cameras viewing the same real world point. Each pixel defines an infinite long ray out in space.

The known parameters are the camera matrices for both cameras and the pixel coordinates in both cameras which are u^l, v^l, u^r and v^r. For each camera there are two mappings (3.17) and (3.18) which defines the relation between real world coordinates and pixel coordinates. The system that needs to be solved to find the single point position is established based on (3.17) and (3.18). The equations are separated based on the x, y and z-values which are the unknowns.

$$(c_{11} - c_{31}u)x + (c_{12} - c_{32}u)y + (c_{13} - c_{33}u)z = c_{14} - c_{34}u \tag{3.22}$$
$$(c_{21} - c_{31}v)x + (c_{21} - c_{32}v)y + (c_{23} - c_{33}v)z = c_{24} - c_{34}v \tag{3.23}$$

There are two equations, (3.22) and (3.23), for each camera. For a stereo system there are then four equations which can be put into a linear system such as this. The pixel coordinates and the camera matrices for the two cameras are denoted with subscript l and r for left and right camera respectively.

$$\begin{bmatrix} c^l_{11} - c^l_{31}u^l & c^l_{12} - c^l_{32}u^l & c^l_{13} - c^l_{33}u^l \\ c^l_{21} - c^l_{31}v^l & c^l_{21} - c^l_{32}v^l & c^l_{23} - c^l_{33}v^l \\ c^r_{11} - c^r_{31}u^r & c^r_{12} - c^r_{32}u^r & c^r_{13} - c^r_{33}u^r \\ c^r_{21} - c^r_{31}v^r & c^r_{21} - c^r_{32}v^r & c^r_{23} - c^r_{33}v^r \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c^l_{14} - c^l_{34}u^l \\ c^l_{24} - c^l_{34}v^l \\ c^r_{14} - c^r_{34}u^r \\ c^r_{24} - c^r_{34}v^r \end{bmatrix} \tag{3.24}$$

$$\mathbf{X}\mathbf{p_w} = \mathbf{b} \tag{3.25}$$

(3.24) is an overdetermined linear system which can be solved with linear least square. One

solution is by solving the corresponding normal equation (3.26) as derived in (Edwards & Penney 2005).

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{p}_{\mathbf{w}} = \mathbf{X}^{\mathrm{T}}\mathbf{b} \tag{3.26}$$

The linear system can easily be extended to include more than two cameras.

Chapter 4

Video Tracking Design

From the previous chapter it is concluded that by having two or more cameras and tracking the same point in all the cameras views it is possible to compute the 3-D position of that point. This will form the basis for the video measurement system that will be developed through the next chapters. Finding the position and attitude of the vessel involves several steps like finding the vessel, selecting tracking points, finding stereo correspondence and computing 3-D motion. This chapter will develop an algorithm for finding the vessel in the video and present a single point tracking algorithm.

4.1 Tracking Initialization

This analysis starts by looking at the characteristics which a supply vessel poses to the video frame compared to the sea. The first to notice is that the sea is largely random. A wave can be followed from one frame to the next, but after a short time the view can be unrecognizable. If a supply vessel is present in the video it provides much more structured pixel data. Not only is the pixel data similar in one frame to the next, but it usually has geometric properties which are simple. Supply vessel usually have a square shaped loading deck where the width of the deck is constant along its whole length. The lines which a supply vessels poses to a video frame will be used as basis for the tracking initialization algorithm derived in the next sections.

The Hough-transform for extracting *infinite straight lines* from images will be presented here. Before the Hough transform can be applied the image must be filtered in such way that the edges of the supply vessel become clear. The most common method is to use a edge detection algorithm. One of the more common edge detection algorithms is the Canny edge detector. It is a multi-stage algorithm where the output is a binary image in equal size as the input image where white pixels indicate an edge. Refer to the original paper (Canny 1986) for details about the algorithm.

The Hough transform for finding straight lines parametrizes each possible line with two pa-

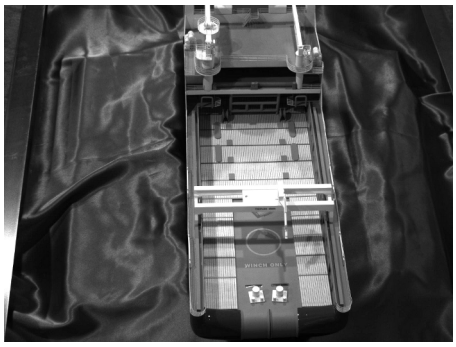


Figure 4.1: The original video image.

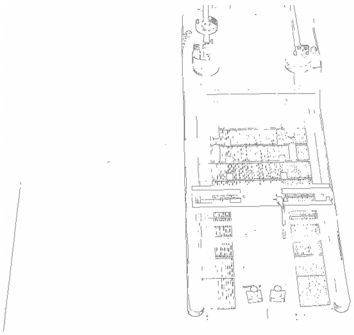


Figure 4.2: Output from the canny edge detector.

rameters, ρ and θ. Let ρ define the length of the normal from the origin (upper left corner of image) to the line to be parametrized. Let θ define the angle between the normal and the x-axis. See figure 4.4. The hough transform algorithms starts by establishing a two dimensional accumulator along the θ and ρ parameters. The parameter range and resolution of the accumulator is selected when implementing the algorithm.

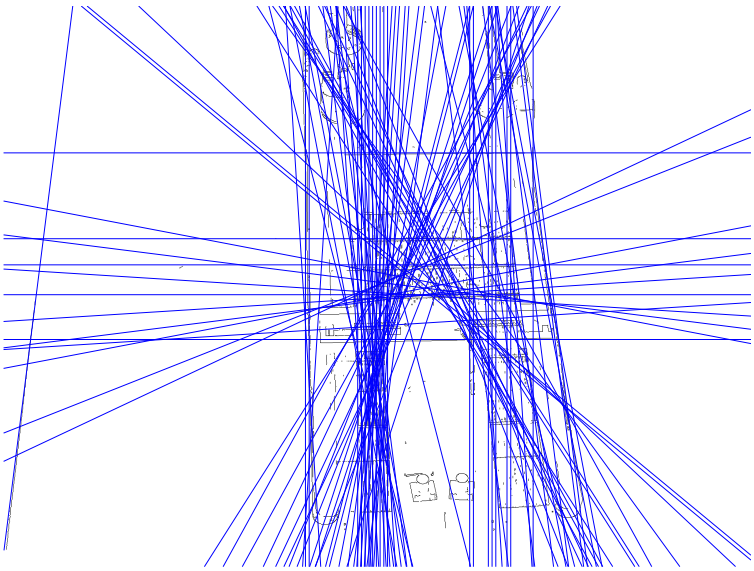


Figure 4.3: The Hough lines projected on top of the output from the Canny edge detector.

The algorithm starts by checking each pixel in the input image. Each pixel denotes whether a pixel is an line or not. High value means high probability and vice versa. For each pixel (u, v) all the combinations of θ and ρ which corresponds to a line passing through pixel (u, v) are found. The corresponding positions in the accumulator are added with the pixel value or incremented with one if pixel values are boolean and true. Pixels which resemble a line will give parameters which are approximately the same and the accumulator will as a result have positions with larger values. Those positions can be found by searching for local maximums.

The output from the Hough transform algorithm is a list of parameter pairs which are considered matched lines. Number of lines depends on adjustable parameters in a particular implementation.

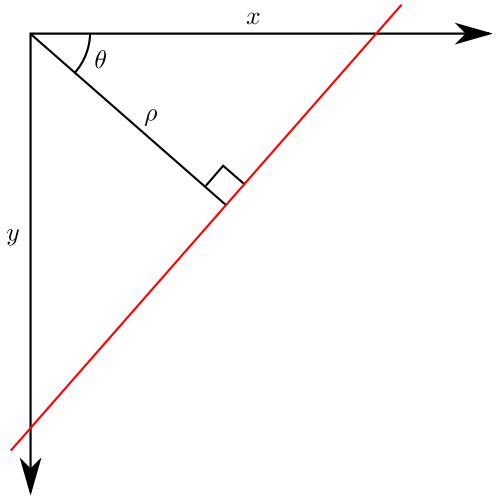


Figure 4.4: Parametrization of an arbitrary line in terms of angle θ and distance from origin ρ.

Rectifying lines into the sea surface

The output from the Hough transform originates mostly from the geometric properties of the vessel in the video frame. Edges which are always present are the sides of the vessel. Highly dependent on the structure of the deck and the cargo which may be present at the deck, there will certainly be lines which do not correspond to the sides. A search algorithm must be developed to find the best candidate of lines which represent the port and starboard sides of the vessel. Due to the positioning of the camera as described in chapter 2, parallel lines at the sea surface will only appear parallel in the video frame if the image plane of the camera is parallel to the plane which the parallel lines defines. This is only true if the camera is looking straight down on the vessel. Since this is not a very likely situation it cannot be assumed that vessel sides will appear parallel in the video frame.

By making assumptions about the captured scene, the lines found by the Hough-transform can be rectified in a form which makes parallel vessel edges to appear parallel in the video frame in most situations. This is possible since the camera's positions relative to the sea surface is known through camera calibration. An assumption must be made for this to be possible is: The edges found must be at zero z-level. If this is not true the non-parallel lines will not be rectified correctly and/or lines which are not parallel could be misleadingly be rectified to parallel lines in the video.

It is assumed that the plane $z = 0$ in the WCS corresponds to the sea surface. Using (3.17) and (3.18) and setting $z_w = 0$ the mapping from WCS coordinates to pixel indexes becomes

$$\begin{aligned} u &= \frac{c_{11}x_w + c_{12}y_w + c_{14}}{c_{31}x_w + c_{32}y_w + c_{34}} \\ v &= \frac{c_{21}x_w + c_{32}y_w + c_{14}}{c_{31}x_w + c_{32}y_w + c_{34}} \end{aligned} \tag{4.1}$$

x_w and y_w in (4.1) are now fully determined from the pixel coordinates u and v and can be found by solving (4.2)

$$\begin{bmatrix} uc_{31} - c_{11} & uc_{32} - c_{12} \\ vc_{31} - c_{21} & vc_{32} - c_{22} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \end{bmatrix} = \begin{bmatrix} c_{14} - uc_{34} \\ c_{24} - vc_{34} \end{bmatrix} \tag{4.2}$$

$$\mathbf{Ax} = \mathbf{b} \tag{4.3}$$

x_w and y_w can be found to be

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \frac{1}{\det \mathbf{A}} \begin{bmatrix} u(c_{34}c_{22} - c_{24}c_{32}) + v(c_{14}c_{32} - c_{34}c_{12}) + c_{24}c_{12} - c_{14}c_{22} \\ u(c_{31}c_{24} - c_{21}c_{34}) + v(c_{11}c_{34} - c_{31}c_{14}) + c_{21}c_{14} - c_{11}c_{24} \\ 0 \end{bmatrix} \tag{4.4}$$

where

$$\det \mathbf{A} = u(c_{21}c_{32} - c_{31}c_{22}) + v(c_{31}c_{12} - c_{11}c_{32}) + c_{11}c_{22} - c_{21}c_{12} \tag{4.5}$$

Now the new Hough-parameters ρ_w and θ_w for the *rectified* lines must be found. This can be done by selecting two points on the original line and computing the corresponding position of those two points in the xy-plane. The two points can be given as

$$\mathbf{p_1} = [\rho\cos(\theta), \rho\sin(\theta)] \tag{4.6}$$

$$\mathbf{p_2} = \mathbf{p_1} + [\sin(\theta), -\cos(\theta)] \tag{4.7}$$

Let now $\mathbf{v}_1$ and $\mathbf{v}_2$ be the corresponding xy 2-D points for $\mathbf{p}_1$ and $\mathbf{p}_1$ in the xy-plane computed by (4.4). Those two points define the line properly, but a little more computation must be done to find the appropriate Hough parameters. The θ Hough parameter defines the angle of the vector which intersects the line perpendicularly from the origin. This intersection point is not necessary $\mathbf{v}_1$ or $\mathbf{v}_2$ so it must be found.

The intersection point is located on the line defined by $\mathbf{v_1}$ and $\mathbf{v_2}$ and can be expressed as

$$\mathbf{v_i} = \mathbf{v_1} + u(\mathbf{v_2} - \mathbf{v_1}) \tag{4.8}$$

where u is a scalar. Let u_i be the value when the intersection is perpendicular. Then the dot product is zero, that is

$$(\mathbf{v_1} + u_i(\mathbf{v_2} - \mathbf{v_1})) \cdot (\mathbf{v_2} - \mathbf{v_1}) = 0 \tag{4.9}$$

Solving for u_i yields

$$u_i = \frac{y_1(y_2 - y_1) - x_1(x_2 - x_1)}{(y_2 - y_1)^2 - (x_2 - x_1)^2} \tag{4.10}$$

The intersection point $\mathbf{v_i}$ can now be defined

$$\mathbf{v}_i = \mathbf{v}_1 + u_i(\mathbf{v}_2 - \mathbf{v}_1) \tag{4.11}$$

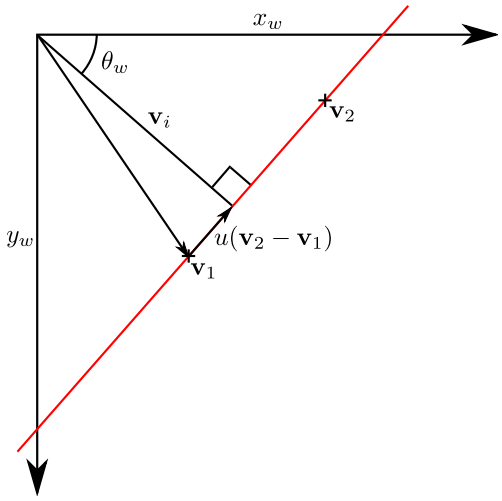


Figure 4.5: Finding Hough-params for a line given two points $\mathbf{v_1}$ and $\mathbf{v_2}$ on the line.

and the Hough-parameters can be defined

$$\rho_w = |\mathbf{v}_i| \tag{4.12}$$
$$\theta_w = \text{atan2}(y_i, x_i) \tag{4.13}$$

where x_i and y_i are the coordinates of $\mathbf{v}_i$.

Depending on the implementation or choice of the particular Hough-transform algorithm, the output θ may not be within the interval from 0 to π. For easing the later analysis the θ_w-parameter is normalized so it is always between 0 and π and the ρ is corrected accordingly. The following simple check is used.

if $\theta_w > \pi$ **then**
  $\theta_w \leftarrow \theta_w - \pi$
  $\rho_w \leftarrow -\rho_w$
else if $\theta_w < 0$ **then**
  $\theta_w \leftarrow \theta_w + \pi$
  $\rho_w \leftarrow -\rho_w$
end if

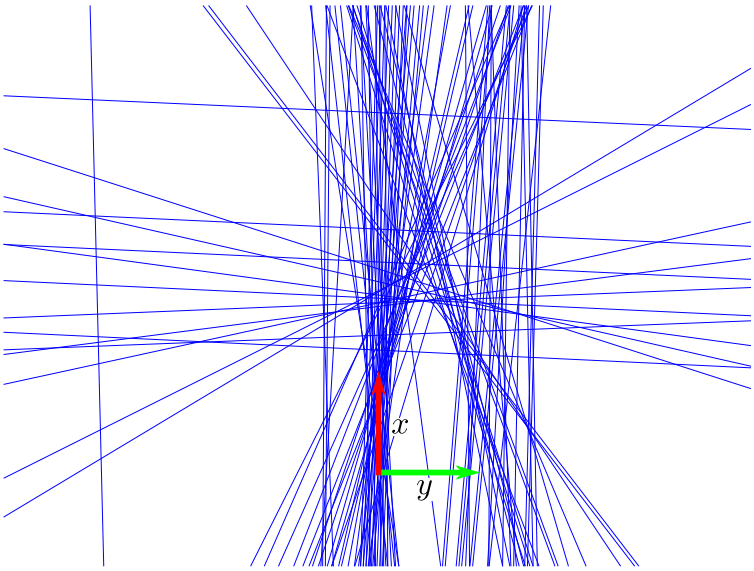


Figure 4.6: Rectified Hough lines

After all the lines have been rectified they now represent lines which are parallel to the sea surface. This means that lines which were originally parallel to each other at sea surface will appear parallel in the rectified data set. The next step in the analysis is to group lines which have similar angles. There are are many ways to group lines based on similar angle. One criteria could be that within each group the θ deviation must be less than some value. It is not trivial to find an optimal solution. A simple and fast solution will be presented here.

A random line is chosen, a line group is created and the chosen line is added. All the other lines are compared to the first added line. The lines which have a *theta* value which does not differ more than a predefined value are also added to a group. Only lines which have not yet been added to a line group are compared. The algorithm is given in pseudo code in algorithm 1.

When all the lines has been been grouped together based on similar theta angle, what there is left is to find the group which has a line distribution which best matches the vessel sides. A form of voting system should be created to find the best match. If the lines are concentrated at distances which are the positions of the vessel sides there should be a good match. If the lines seem to be rather randomly positioned there should be a poor match. One of the parameters is the spacing between the sides of the vessel, or the *width* of the vessel. If the width is fixed to a constant value, the only free parameter is the position of the sides of the vessel. A common method to search for a particular pattern in a data set is to use cross-correlation. The next paragraphs will present a way of using this technique to find the position of the vessel edges in a dataset of nearly parallel lines.

Algorithm 1 Group lines by angle

```
g ← 0
m {this value is the maximum deviation from the first line added}
for every line as i do
  if i is processed then
    continue
  end if
  i ← processed
  for every line as j do
    if j is processed then
      continue
    end if
    d ← abs(i.theta - j.theta)
    if d < m or d > π − m then
      j ← processed
      add to group g
    end if
  end for
  g ← g + 1
end for
```

A cross-correlation based search method

Cross-correlation compares searches a signal with the use of a predefined template. The result tells where there was the best match between the template and the signal. The signal in this case is a line group, or more specifically the position of the lines. The lines should be nearly parallel since each group is grouped based on similar angles. The other parameters is the ρ value which tells how far the line is located from the origin. The signal to be searched in must have a defined length which covers all the positions which have a line. The signal will be a boolean signal since only the existence of a line is known, not in how great extent the existence is valid. The signal will have "true" at the positions which there exists lines and "false" elsewhere.

Next the template which is used in the searching must be created. The simplest would be to find the two lines, if any, whose distance between each other is exactly the predefined vessel width. This would not work in the presence of noise and errors. Therefore a template with takes errors into account is a requirement. The template must consist of two regions which represent the port and starboard positions. The highest value in the template should be at the positions which are separated with the "vessel width" distance. The value should decrease with some magnitude further away from this position. The center and decrease rate should be controlled easily. A curve which have these properties are the normal distribution curve used in statistics. It is usually used to describe the probability distribution for a stochastic

variable. Its functions is given in (4.14).

$$\Phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{4.14}$$

The peak is controlled with the μ parameters. The "width" or how fast the curve decreases from the peak is controlled by the σ parameter which is commonly called standard deviation. About 95% of the area under the curve is located within 2σ on each side of the curve. The total area under the curve is always 1. The search template is created by using two normal distribution functions placed at a spacing apart. (4.15) is used for that. It is also divided by 2 such that area under the graph is still 1.

$$T(x) = \frac{1}{2\sqrt{2\pi\sigma^2}} \left(e^{\frac{-(x-\mu_1)^2}{2\sigma^2}} + e^{\frac{-(x-\mu_2)^2}{2\sigma^2}} \right) \tag{4.15}$$

Figure 4.7 shows an example where the vessel width is set to be 23 meters and the standard deviation is 1 meter. The template is constructed such that values that fall outside the 2σ-limit on each side of the template are neglected.

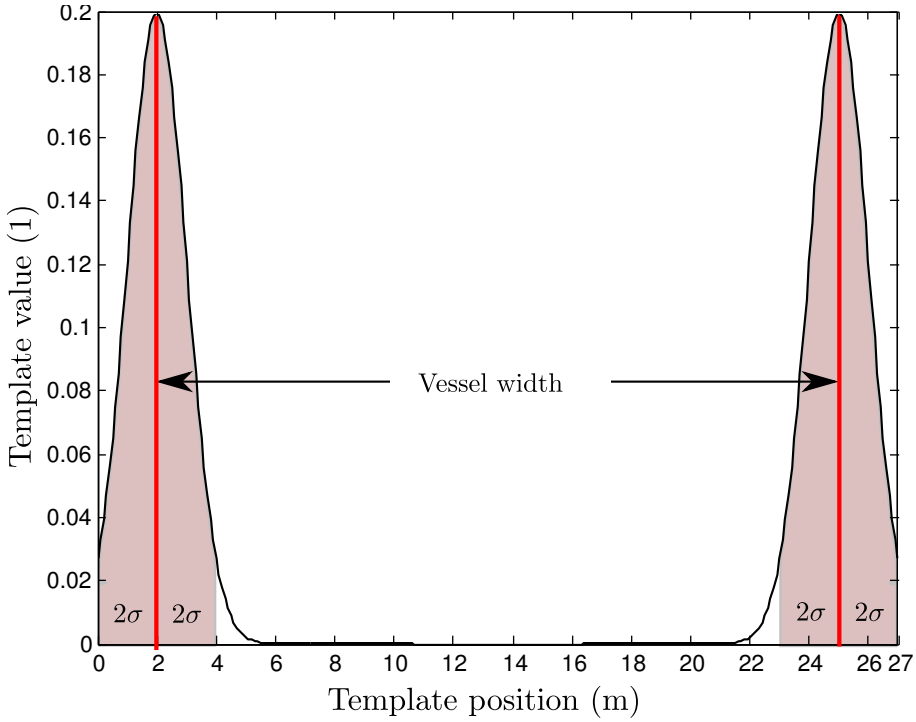


Figure 4.7: Template for searching for vessel sides with values $\mu_1 = 2$, $\mu_2 = 25$, $\sigma = 1$

To perform the cross-correlation the boolean signal must be created based on the line group and the template must be discretized with some accuracy. The expression for computing discrete cross-correlation is given by (4.16) where f is the template function and g is the

signal to be searched in.

$$s(n) = \sum_{m=-\infty}^{\infty} f(m)g(n+m) \tag{4.16}$$

Let the template be of limited length such that $f(x) \neq 0 \ \forall \ x \ \in [0,t]$ and 0 everywhere else, and the signal be of limited length such that $g(x) \neq 0 \ \forall \ m \ \in [a,b]$ and 0 everywhere else. (4.16) can be rewritten to

$$s(n) = \sum_{m=0}^{t} f(m)g(n+m), \ n \in [a, b-t] \tag{4.17}$$

One problem with using only cross-correlation is that a large amount of lines can be positioned in only one of the areas and it will still give a high score, even when it should not be considered a match since the other edge is not found. To make the algorithm more robust against an extra check is done along with the cross-correlation. For each computed value of (4.17) it is also checked that the signal has at least one line within the 2σ area in both normal distribution curves. If that is not the case the particular value is set to 0 to ensure a low score.

Algorithm 2 Cross correlation with edge check

```
maxMatch ← 0
matchPos ← 0
for n = a to b - t do
  leftEdgePresent ← false
  rightEdgePresent ← false
  for m = 0 to t do
    s[n] ← s[n] + f[m] · g[m + n]
    if (m < μ1 + 2σ and g[m + n] is true then
      leftEdgePresent ← true
    end if
    if (m > μ2 − 2σ and g[m + n] is true then
      rightEdgePresent ← true
    end if
  end for
  if leftEdgePresent is false or rightEdgePresent is false then
    s[n] ← 0
  end if
  if s[n] > maxMatch then
    maxMatch ← s[n]
    matchPos ← n
  end if
end for
return matchPos {return the position which there was a match}
```

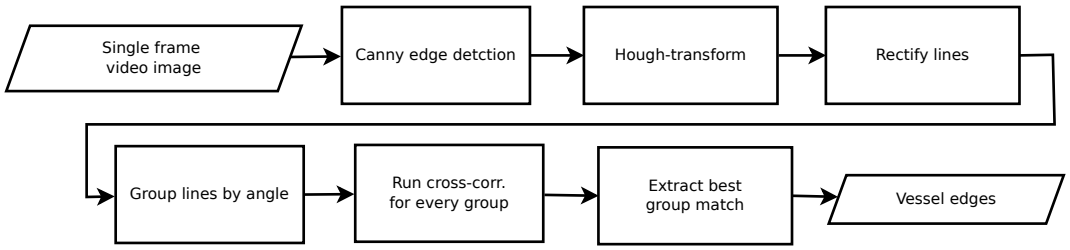


Figure 4.8: Computation steps in finding the vessel

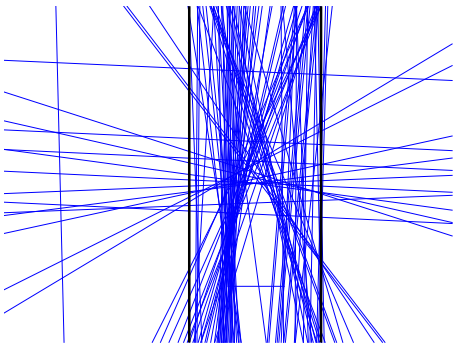


Figure 4.9: Matched lines for a particular data set.

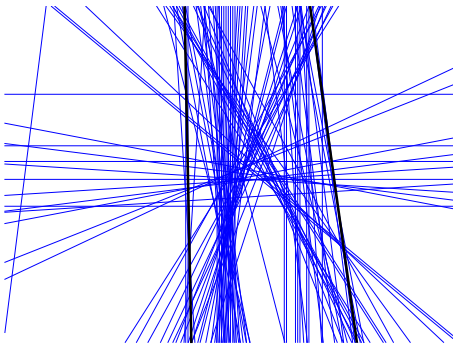


Figure 4.10: The matched lines projected on the original line set.

The cross-correlation algorithm 2 is run for every line group. Finding the best match is done simply by selecting the group with the best match and extracting the position of the matched template. The information acquired from the algorithm developed in this chapter does not give any boundedness in the stern/aft direction other than that the vessel is located somewhere in the region bounded by found vessel edges and the video frame itself.

4.2 Single Point Tracking Method

As described in 2.1 no information about the visual appearance of the vessel should be known beforehand. This disqualifies any method which looks for special markers on the vessel. The tracking method must use only the video image itself for tracking. It is also important that the tracked point does not drift, this means that the algorithm must not end up tracking another point than the initial point.

Cross-correlation based tracking is chosen because it does not change the target representation of the tracked object and the representation can be acquired from the video frame itself and used for successive matches in later frames. This makes the algorithm independent of what is viewed by the cameras. With the cross-correlation method, the target representation is simply a small image of the object to find. The method is not very tolerant to scale and rotation, but as described in chapter 2 a large vessel at sea does not really rotate much. And the distance from the camera to the vessel is relatively constant so scale change would neither be a problem.

Two dimensional cross correlation

Template matching is a two dimensional variant of cross-correlation. The output of such algorithm is a an image where the intensities define how great match there is between the template and the image at a particular position. (4.18) define the basic two dimensional cross-correlation. $T(x, y)$ defines the pixel intensities in the template image at position (x, y). $I(x, y)$ defines the image intensities and $R(x, y)$ is the output match image.

$$R(x, y) = \sum_{m,n} T(m, n) \cdot I(m + x, n + y) \tag{4.18}$$

The best match between the template and the image is found by searching for the maximum value of $R(x, y)$. The found values for x and y denote the upper left corner of the template whose position gives the best match.

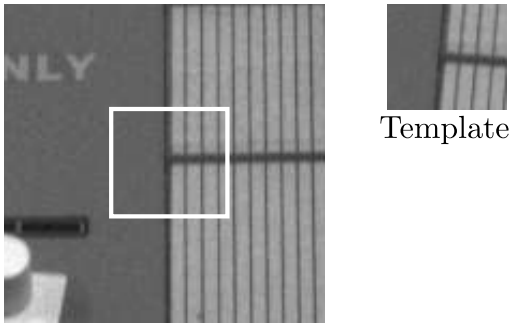


Figure 4.11: A small part of an image (the ROI) with a white square showing the highest match for the template on the right.

Sup-pixel accuracy

Sub-pixel accuracy is possibility of determining the position of an object in an image beyond the spatial quantification of the image (the image pixels). This possible when each pixel has more than one possible value as is true with gray scale and color images. (Skoglund & Felsberg 2006) describes two methods which can be used to implement sub-pixel accuracy. One of the is interpolation. This method will be used here.

First the regular template match performed as described in the previous section. When the position of the template is found, the patch and the matched region in the image including a small region around the patch are linearly interpolated and a new cross-correlation is run again. The matched position is the regular cross-correlation plus the sub-pixel cross-correlation multiplied with a factor which corresponds to the interpolation factor.

Tracking fail detection

Tracking fail detection is the ability to detect when tracking of a single point is not working properly. This is important because tracking of the wrong point leads to wrong correspondence and can give large errors in the computed 3-D position.

The cross-correlation match gives a hint of how good the match was. If the tracked object comes occluded there will be a much lower match. This can be use to detect an occlusion type of tracking fail. Another check is to compute the acceleration of a tracked point. Since it is a physical object that is tracked the maximum acceleration is limited. If the acceleration is above some threshold it can stated that the movement is not true and another type of tracker fail is detected.

4.3 Selection of tracking points

To establish a good tracking constellation the tracking points must be chosen wisely. The two most important properties are widespread constellation and stable tracking points.

The stability of a video tracking is largely dependent on the feature that is being tracked. Not every point in a video can easily be tracked. Difficult or impossible points are: evenly colored areas, straight lines, repetitive pattern and image contaminated by noise. Easy features to track are: corners and symbols. Selecting features which are easy to track will greatly have an effect on the quality of the tracking.

A widespread constellation means that the tracking points are spaced far apart from each other. This gives a better measurement of the edges of the vessel and intuitively better defines movement of the rigid object. If the points are close together small errors in the measurement will lead very large angular errors.

Algorithm

The focus of this algorithm is to ensure both a widespread constellation and stable tracking points. To ensure that the tracking points are spaced far apart a small extension to the tracking initialization algorithm from 4.1 is developed. Multiple rectangles in the plane $z = 0$ are created. Depending on the roll and pitch angles of the ship the plane $z = 0$ approximately coincides with the vessel's cargo deck. When the rectangles are mapped to the camera view they will be transformed to general polygons with four corners.

The rectangles are lay out in two rows, one for the port side and one for the starboard side. Both rows are infinitely long but only the rectangles that are visible in the video frame as polygons are computed. The transformed polygons are indexed such that the port and aftest polygon is rectangle number one. The next polygon is the aftest starboard polygon and so on. The numbering is showed in figure 4.12. Which polygons that should be populated with points is tuned to a specific vessel or a setting which works for most vessel is chosen.

To add a tracking point to a polygon, first "good" tracking points must be found in the area within the polygon. Good tracking points are characterized by corners in images and can be found by searching for high second derivatives in the image. A previously implemented algorithm will be used for this. It is described briefly in 7.2. The implemented algorithm uses results from (Shi & Tomasi 1994).

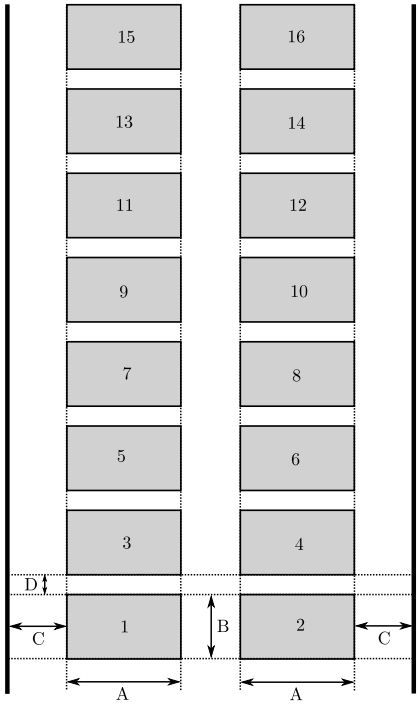


Figure 4.12: Rectangle layout with adjustment factors: A: polygon width, B: polygon height, C: edge spacing, D: polygon forward spacing.

Algorithm 3 Add trackers

desiredPolygons {this is a array of polygons which one want to have tracker in.}
for every *desiredPolygons* as i **do**
  if enough trackers added **then**
    return
  end if
  if i already has tracker **then**
    continue
  end if
  goodPoint $\leftarrow$ find good point in i
  add tracker to *goodPoint*
end for

4.4 Point Correspondence

Point correspondence is the task of finding the pixel coordinate in a second camera given a pixel coordinate in the first camera which corresponds to the same physical point. If two cameras are placed not too far apart and they have the same orientation, then a method which looks for similarities in the two images can be used. However, having two cameras too close together leads to poor depth accuracy. The camera positioning is therefore a compromise between depth accuracy and the difficulty of finding correspondence.

The method which will be used in this application is mainly based on cross-correlation which is a method which looks for similarities. Cross-correlation alone is not feasible because there is a large chance that wrong correspondence is found. To minimize the area to search for correspondence two constraints will be used. The first is the epipolar constraint and the second is a custom planar constraint.

Epipolar constraint

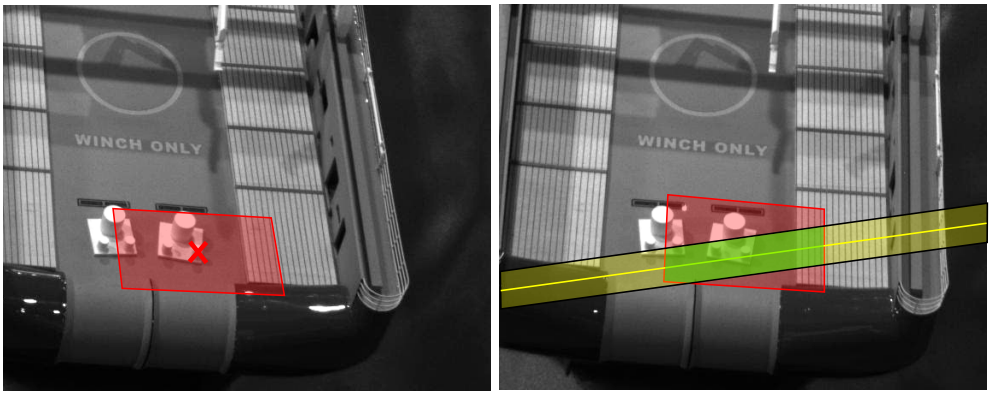
As described in 3.6 a pixel in one camera corresponds to an infinite line in space. This infinite line will be projected as a straight line in any other camera viewing the same volume. This means that searching for correspondence in the second camera can be reduced to searching for correspondence along a single straight line ± some error margin. This search line is based on the epipolar constraint. The corresponding line can be computed if the camera matrices for both cameras are known. The details of epipolar geometry can be found in (Bradski & Kaehler 2008). Previously implemented algorithms will be used to compute the epilines in chapter 7.

Planar constraint

The epipolar constraint limits the correlation search along a line ± some error margin. If there are similar patterns somewhere else along this line the correlation search could easily match with the wrong point. To constraint the search even further the search region in the second camera is constrained to the approximate same xy-plane region as the region surrounding the point in the first camera. Like the Hough-lines rectify algorithm assumes, this method also assumes that the object to be tracked is a plane parallel to the xy-plane (the sea surface). This is true for a cargo deck when the vessel has approximately zero roll and pitch angles. This will happen from time to time and if the correspondence search fails, typically because the epipolar line does not intersect with the xy-region, then successive retries must be done since the vessel is constantly moving.

There are many ways to find a corresponding xy-area in the second camera. The chosen method for this application is to create a square in the xy-plane which encircles the tracker in the first camera. This square is mapped to the image plane of the second camera. A valid correspondent point is required to be inside the square. The mapping of real world planes to pixel coordinate planes and vice versa is possible because one of the real-world coordinates is

constrained to a fixed number.



First camera Second camera

Figure 4.13: First camera finds surrounding xy-polygon (transparent red). Second camera maps the polygon onto the same xy-area (transparent red). The epiline (yellow) with some error margin (transparent yellow) together with the polygon creates the search area (transparent green).

Chapter 5

Position and Attitude Estimation

This chapter looks at problem of finding the position and attitude of the vessel cargo deck based the computation of multiple 3-D points. With multiple 3D points which are attached to the vessel it is possible to compute position and attitude of the vessel.

5.1 Position and Attitude Representation

Two coordinate systems are defined. The n-frame is fixed relative to the cameras and are located on the sea surface with the z-axis pointing down and the x and y-axis pointing arbitrarily along the sea surface such that they form a right-handed coordinate system. For simplicity the cameras are calibrated such that the WCS defined in chapter 3 becomes the definition of the n-frame. From now n-frame equals the WCS. It is assumed that the n-frame is fixed and does not move. This is not completely true when the cameras are attached to a floating structure, but the movement are very slow compared to that of the supply vessel when attached to a large floating rig. It is assumed that the crane base, and therefore the cameras, does not rotate when the tracking is active.

The position and attitude of the vessel can be defined by a coordinate system which is attached to the vessel. This coordinate system is commonly called the body coordinate system or the b-frame. The b-frame origin can be fixed to any point and with any orientation to the vessel, but a convention is to have the x-axis along the longitudinal axis of the vessel, the y-axis to starboard and the z-axis such that a right-handed coordinate system is formed (SNAME 1950). The origin of the b-frame is often chosen to coincide with the principle axes of inertia. But since very little information is available about the tracked vessel the center of the cargo deck is chosen since it is the central part in this tracking system. The orientation is chosen as stated previously in this paragraph.

The transformation between n-frame and b-frame is represented by rotation and translation. (5.1) transforms a point from being given in the b-frame to be given in the n-frame. $\mathbf{T}^n_{nb}$ is the vector from the n-frame's origin to the b-frame's origin, which is denoted by the subscript,

and it is given in the n-frame which is denoted by the superscript. $\mathbf{R}_b^w$ is rotation matrix which defines the rotation of the b-frame relative to the n-frame.

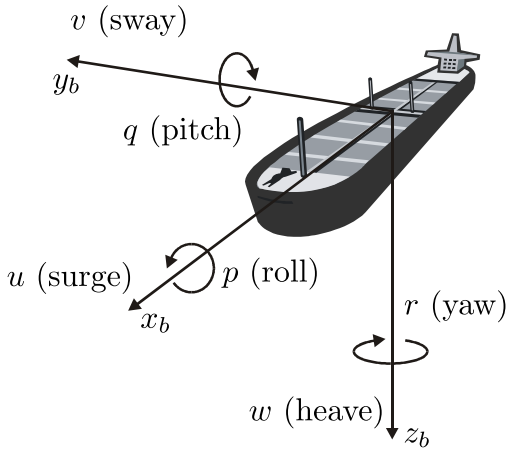


Figure 5.1: Illustration of the vessel-fixed reference system and its motion variables.

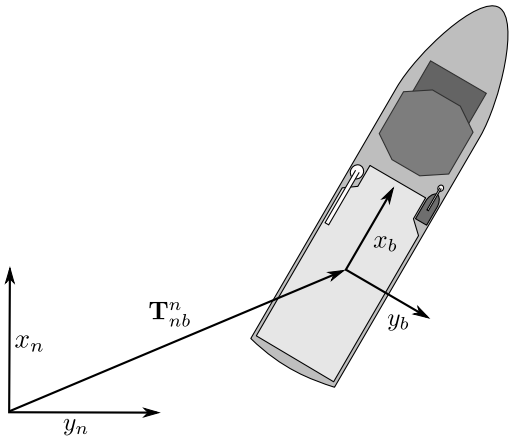


Figure 5.2: The relation between the n-frame and the b-frame.

$$\mathbf{p}^n = \mathbf{R}_b^n \mathbf{p}^b + \mathbf{T}_{nb}^n \tag{5.1}$$

5.2 Estimation of 3-D Rigid Body Transformation

The estimation of position and attitude is based on the rigid body transformation of a 3D point set. The chosen algorithm is based on the results from (Eggert et al. 1997) which compares four major algorithms for estimating 3-D rigid body transformation. The compared algorithms does not take into account outliers in the data set like the RANSAC algorithm does (Fischler & Bolles 1981). Therefore it is assumed that every point is valid. The simplest algorithm was chosen for this thesis as it was concluded that none of the compared algorithms were superior.

The derivation starts by presenting a singular value decomposition based algorithm from (Arun et al. 1987). Suppose that there are two point sets $\mathbf{m}_i$ and $\mathbf{d}_i$ which have N correspondent points. Correspondent means that each point is related to a specific point in the other set. Let the relation for point i be given by

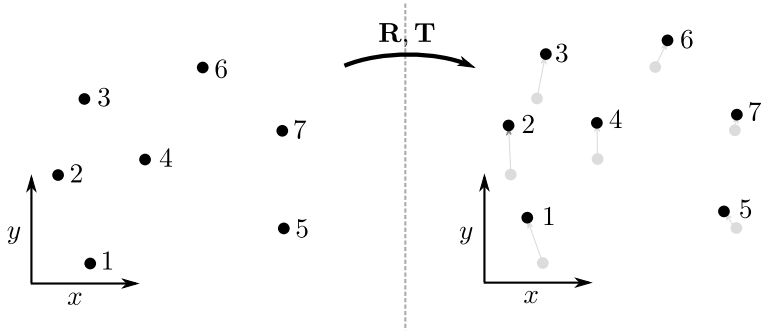


Figure 5.3: The rotation and translation of a point set.

$$\mathbf{d}_i = \mathbf{R}\mathbf{m}_i + \mathbf{T} + \mathbf{V}_i \tag{5.2}$$

where $\mathbf{V}_i$ is an unknown error vector which exists due to measurement and discretization noise. $\mathbf{R}$ and $\mathbf{T}$ can be estimated by minimizing a least squares error criterion given by:

$$S = \sum_{i=0}^{N} \parallel \mathbf{d}_i - \hat{\mathbf{R}}\mathbf{m}_i - \hat{\mathbf{T}} \parallel^2 \tag{5.3}$$

where $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ are the estimated values for $\mathbf{R}$ and $\mathbf{T}$.

If the point sets have the same centroid the term $\hat{\mathbf{T}}$ can be removed from (5.3). $\hat{\mathbf{T}}$ can be computed afterwards. Let $\mathbf{m}_{c_i}$ and $\mathbf{d}_{c_i}$ be the original point sets translated such that they have the centroid at the origin.

$$\bar{\mathbf{d}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{d}_i \tag{5.4}$$

$$\bar{\mathbf{m}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{m}_i \tag{5.5}$$

The following sets now have the origin as centroid

$$\mathbf{d}_{c_i} = \mathbf{d}_i - \bar{\mathbf{d}} \tag{5.6}$$

$$\mathbf{m}_{c_i} = \mathbf{m}_i - \bar{\mathbf{m}} \tag{5.7}$$

(5.3) can be simplified to

$$S = \sum_{i=1}^{N} \parallel \mathbf{d}_{c_i} - \hat{\mathbf{R}} m_{c_i} \parallel^2 \tag{5.8}$$

Writing out (5.8) gives

$$S = \sum_{i=0}^{N} \left(\mathbf{d}_{c_i} - \hat{\mathbf{R}}\mathbf{m}_{c_i} \right)^{\mathrm{T}} \left(\mathbf{d}_{c_i} - \hat{\mathbf{R}}\mathbf{m}_{c_i} \right) \tag{5.9}$$

$$= \sum_{i=0}^{N} \left(\mathbf{d}_{c_i}^{\mathrm{T}}\mathbf{d}_{c_i} - \mathbf{d}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} - (\hat{\mathbf{R}}\mathbf{m}_{c_i})^{\mathrm{T}}\mathbf{d}_{c_i} + (\hat{\mathbf{R}}\mathbf{m}_{c_i})^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} \right) \tag{5.10}$$

$$= \sum_{i=0}^{N} \left(\mathbf{d}_{c_i}^{\mathrm{T}}\mathbf{d}_{c_i} - \mathbf{d}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} - \mathbf{m}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}^{\mathrm{T}}\mathbf{d}_{c_i} + \mathbf{m}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} \right) \tag{5.11}$$

$$= \sum_{i=0}^{N} \left(\mathbf{d}_{c_i}^{\mathrm{T}}\mathbf{d}_{c_i} + \mathbf{m}_{c_i}^{\mathrm{T}}\mathbf{m}_{c_i} - 2\mathbf{d}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} \right) \tag{5.12}$$

(Eggert et al. 1997) states further that minimizing (5.12) is equivalent to maximizing

$$\sum_{i=0}^{N} \mathbf{d}_{c_i}^{\mathrm{T}}\hat{\mathbf{R}}\mathbf{m}_{c_i} = \mathrm{trace}(\hat{\mathbf{R}}\mathbf{H}) \tag{5.13}$$

where $\mathbf{H}$ is a 3×3 correlation matrix defined by

$$\mathbf{H} = \sum_{i=0}^{N} \mathbf{m}_{ci}\mathbf{d}_{ci}^{\mathrm{T}} \tag{5.14}$$

The proof of this is given in (Arun et al. 1987). Let the singular value decomposition of $\mathbf{H}$ be given by $\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}$. Then the matrix $\hat{\mathbf{R}}$ which best maps the first point set to the second point set can be given by

$$\hat{\mathbf{R}} = \mathbf{V}\mathbf{U}^{\mathrm{T}} \tag{5.15}$$

(Eggert et al. 1997) states that optimal $\hat{\mathbf{R}}$ given by (5.15) may not always have a positive determinant. A negative determinant corresponds to a reflection and a left-handed coordinate

system rather then the desired *rotation*. A solution which compensates for this is given by (Kanatani 1994) where Trace($\hat{\mathbf{R}}^{\mathrm{T}}\mathbf{H}$) is maximized over all rotations by

$$\hat{\mathbf{R}} = \mathbf{U}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{U}\mathbf{V}^{\mathrm{T}}) \end{pmatrix}\mathbf{V}^{\mathrm{T}} \tag{5.16}$$

The original document (Eggert et al. 1997) maximizes Trace($\hat{\mathbf{R}}\mathbf{H}$) where $\hat{\mathbf{R}}$ is not transposed. It can be shown that the solution by (Kanatani 1994) will find $\hat{\mathbf{R}}^{\mathrm{T}}$ and not $\hat{\mathbf{R}}$. This means that the solution of (Kanatani 1994) can be transposed such that Trace($\hat{\mathbf{R}}\mathbf{H}$) is maximized over all rotations by

$$\hat{\mathbf{R}} = \left(\mathbf{U}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{U}\mathbf{V}^{\mathrm{T}}) \end{pmatrix}\mathbf{V}^{\mathrm{T}}\right)^{\mathrm{T}} \tag{5.17}$$

$$= \mathbf{V}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{U}\mathbf{V}^{\mathrm{T}}) \end{pmatrix}\mathbf{U}^{\mathrm{T}} \tag{5.18}$$

Translation

The optimal translation between the two point sets are found as distance between the centroid of the set $\mathbf{d_i}$ and the rotated centroid of the set $\mathbf{m_i}$

$$\hat{\mathbf{T}} = \bar{\mathbf{d}} - \hat{\mathbf{R}}\bar{\mathbf{m}} \tag{5.19}$$

The algorithm which finds the optimal transformation between two point sets which are attached to a rigid object is now established.

5.3 Point Set Managing

At any time there exists two point sets which are measured in the n-frame by the cameras. The *current* point set is the set which have the latest computed 3-D positions. The *current* point set is defined as $\mathbf{d}_i$ where $i \in [1,\ N]$ and the centroid is called $\bar{\mathbf{d}}$. The *initial* point set is another set which is a captured version of an *earlier* current point set and it is defined as $\mathbf{m}_i$ where $i \in [1,\ N]$ and the centroid is called $\bar{\mathbf{m}}$. A point set managing strategy which updates the two point set correctly must be defined before the previously presented algorithm can be used. The algorithm should solve the following problems:

- Ability to choose an arbitrary body origin and attitude relative to current the point set.

- Provide continuous measurement in the event of a point set change.

The ability to choose an arbitrary body origin is solved by defining a "body offset" vector $\mathbf{p}_{cb}^{b}$ which defines the origin of the b-frame relative to the current centroid. The body origin is called $\mathbf{p}_{nb}^{n}$ then defined such that

$$\mathbf{p}_{nb}^{n} = \bar{\mathbf{d}} + \hat{\mathbf{R}}_{b}^{n}\mathbf{p}_{cb}^{b} \tag{5.20}$$

where $\hat{\mathbf{R}}_{b}^{n}$ is the estimation of $\mathbf{R}_{b}^{n}$ whose computation will be explained in the next section.

Startup procedure

It is assumed that at least three 3-D points are available as this is the minimum number of points required to define 6 DOF motion. The position of the body origin and the attitude must be chosen in the startup procedure. For now it is assumed that they are known and are called $\mathbf{T}_{init}$ and $\mathbf{R}_{init}$. These are the initial values for $\mathbf{T}_{nb}^{n}$ and $\mathbf{R}_{b}^{n}$ in (5.1). Different strategies for finding estimates of these will be discussed later in this chapter.

Let $\mathbf{m}_{i}^{n}$ be the particular point set at some time t. This is the *initial* point set. Let $\mathbf{d}_{i}$ be the *current* point set at any later time. The next step is to use $\mathbf{R}_{init} = \mathbf{R}_{b}^{n}$ to align this saved point set onto a body frame. This means transform them from being given in the n-frame to being given in the b-frame. Let $\mathbf{m}_{i}$ be the *initial* point set transformed to the body frame such that

$$\mathbf{m}_{i} = \mathbf{R}_{init}^{\mathrm{T}}\mathbf{m}_{i}^{n} \tag{5.21}$$

Next the body offset vector $\mathbf{p}_{cp}^{b}$ must be computed. It is computed according to

$$\mathbf{p}_{cp}^{b} = \mathbf{R}_{init}^{\mathrm{T}}(\mathbf{T}_{init} - \bar{\mathbf{m}}^{n}) \tag{5.22}$$

where $\bar{\mathbf{m}}^{n}$ is the centroid of the set $\mathbf{m}_{i}^{n}$ for $i \in [1,\ N]$. To compute the vessel attitude the point set $\mathbf{m}_{i}^{b}$ is compared with the current point set $\mathbf{d}_{i}$ for each new available point set measurement using the method presented in the previous section. The computed $\hat{\mathbf{R}}_{b}^{n}$ will be an estimate of $\mathbf{R}_{b}^{n}$ which most optimally transforms $\mathbf{m}_{i}^{b}$ to be aligned with $\mathbf{d}_{i}$.

The estimated position of the body origin can now be computed with

$$\hat{\mathbf{T}}_{nb}^{n} = \bar{\mathbf{d}} + \hat{\mathbf{R}}_{b}^{n}\mathbf{p}_{cp}^{b} \tag{5.23}$$

and based on (5.1) any other point $\mathbf{p}^{b}$ in the body frame can be computed in the world frame by

$$\mathbf{p}^n = \hat{\mathbf{R}}_b^n \mathbf{p}^b + \mathbf{T}_{nb}^n \tag{5.24}$$
$$= \hat{\mathbf{R}}_b^n \mathbf{p}^b + \bar{\mathbf{d}} + \hat{\mathbf{R}}_b^n \mathbf{p}_{cp}^b \tag{5.25}$$
$$= \hat{\mathbf{R}}_b^n \left(\mathbf{p}^b + \mathbf{p}_{cp}^b \right) + \bar{\mathbf{d}} \tag{5.26}$$
$$\tag{5.27}$$

Point set change handling

It can not be assumed that the same set of 3D points are available at all times. Points may disappear due to loss of tracking and new points may arrive due to improvement of the tracking. To provide continuous position and attitude measurement in the event of point set change special actions must be performed. When the current point set change the initial point set must also be cleared and updated as the *initial* and *current* sets must be correspondent.

Let $\mathbf{d}_i^{new}$ be the new point set where new points have arrived or old ones may have disappeared. In general the new point set can be completely different than the previous. Since the new point set centroid may be different, the body offset vector must be updated such that that body origin will be located at same position as before. It is updated based on the current origin such that

$$\mathbf{p}_{cp}^{b,new} = \hat{\mathbf{R}}_n^b \left(\hat{\mathbf{T}}_{nb}^n - \bar{\mathbf{d}}_i^{new} \right) \tag{5.28}$$
$$= \hat{\mathbf{R}}_n^b \left(\bar{\mathbf{d}}_i + \hat{\mathbf{R}}_b^n \mathbf{p}_{cp}^b - \bar{\mathbf{d}}_i^{new} \right) \tag{5.29}$$
$$= \hat{\mathbf{R}}_n^b \left(\bar{\mathbf{d}}_i - \bar{\mathbf{d}}_i^{new} \right) + \mathbf{p}_{cp}^b \tag{5.30}$$

If the old and the new centroids are the same the equation reduces to $\mathbf{p}_{cp}^{b,new} = \mathbf{p}_{cp}^b$ which is intuitively correct.

To provide continuous attitude measurement the needed actions are similar to that of providing the initial attitude matrix $\mathbf{R}_{init}$ during the startup process. The initial attitude matrix is taken from the latest estimated attitude matrix such that $\mathbf{R}_{init}^{new} = \hat{\mathbf{R}}$.

5.4 Estimating $\mathbf{R}_{init}$ and $\mathbf{T}_{init}$

Given the requirement of a body frame which are situated on the middle of the cargo deck, the difficulty of estimating $\mathbf{R}_{init}$ and $\mathbf{T}_{init}$ lies in the unknown position of the tracked 3-D points on the actual vessel. In this section a method which approximately finds a body origin which are located at the cargo deck center and an attitude which are aligned correctly with the vessel. The algorithm uses both information found by the tracking initialization algorithm and properties of the 3-D point set.

From the tracking initialization algorithm two parameters are found: the vessel heading and the center line of the vessel.

Attitude

The vessel heading is represented with a two dimensional vector in the xy-plane which points in the forward direction. Let the forward vector be given by $\mathbf{v_f} = [x_f,\ y_f]$. An initial yaw angle is computed with $\psi_{init} = \text{atan2}(y_f,\ x_f)$. Assuming zero roll and pitch angles $\mathbf{R}_{init}$ can easily be computed with a rotation around the z-axis where the rotation angle is the rotation from the n-frame to the the b-frame. The resulting matrix transforms a vector from the b-frame to the n-frame

$$\mathbf{R}_{init} = \mathbf{R}_{z,\psi_{init}} = \mathbf{R}_b^w = \begin{bmatrix} \cos(\psi_{init}) & -\sin(\psi_{init}) & 0 \\ \sin(\psi_{init}) & \cos(\psi_{init}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.31}$$

Since the roll and pitch angles not known during the startup procedure, the body frame selected by the system may easily be misaligned compared to the longitudinal and transversal axes of the vessel. It can be assumed in most cases that the mean roll and pitch angles of a vessel are zero because this is the stable position of the vessel when it is at rest. This can be used to filter out constant offsets originating from unknown startup angles. It can be implemented by finding the average the of pitch and roll angles, and subtracting the average values on the roll and pitch measurements.

Position

Finding good estimate of the initial body origin is some what more cumbersome. However, since the initial yaw angle is found separately, the most aft tracking point can be calculated. Assuming that the vessel has tracking points close to the aft edge, the most aft point would be closest to the aft edge. When the aft edge is found all edges except the extent of the bow is known. If some idea about the longitudinal size of the cargo deck is known, the body origin can be placed approximately at the cargo deck center.

The estimated position is subject to a z-axis offset which depends on the current position of the vessel when the startup procedure was run. This offset can be counteracted with more or less advanced algorithms which estimates a plane position and attitude in the 3-D point cloud. This is difficult though since containers cranes and other high objects can give outlying points, which do not belong to the cargo deck, but still are valuable tracking points.

Chapter 6

Software Design

In machine vision systems it is difficult to simulate the image processing algorithms. Therefore implementing a real-time application is a way of showing proof-of-concept for the theoretical work. This chapter starts by choosing a development platform which will greatly define development path. An important part of the software design is to establish a modular view of the application. This means placing the algorithms defined in the previous chapters into appropriate modules and define their interface towards the rest of the system.

6.1 Platform Choice

Various aspects must be considered when choosing an implementation platform. Important platform properties are: built-in features, cost, flexibility and compatibility. Two secondary aspects are personal previous experience and academic value.

The abstraction level of different platforms can be very different and for *programming language* platforms it is separated by the used programming language. High level platforms may not even have a programming language concept and tend to be more an "application-specific development environment". These are merely tools for creating an application without the explicit need for programming.

Available functionality for a particular platform is usually limited to the available libraries that there exists for the programming language. Using functionality from a low-level platform in a high-level platform is possible, but is often cumbersome unless the high-level application has been prepared for it. The other way around is usually *not* possible. Although exceptions exists like Matlab which provides a C-interface which makes it possible to use Matlab functionality in C and Fortran applications (Mathworks 2010). However, the basic rule is to choose the platform which provides the most of the useful functionality natively which means in the same environment or programming language.

Main application properties

The most significant properties of the application to be implemented are

- Processing high-definition video streams from multiple cameras simultaneously.
- Easy implementation of custom algorithms.
- Various video and image processing algorithms.
- Graphical user interface with flexible visualization.
- Utilizing multi-core computers.

Discussion

The needed properties can be divided into custom algorithms which must be implemented independently of the chosen platform and functionality that can be provided by a platform. The custom parts of the application are most of the tracking initialization algorithm, the manager of the point cloud estimation and large parts of the user interface. Parts that are typically provided by a platform are camera drivers, video input handling, multi-core optimization and UI creation tools.

Matlab is a platform which often used in the scientific arena. Although there exists an image and video processing functionality the support for *real-time* machine vision is limited. However, Matlab is useful for fast implementation for algorithm verification and for simulations of physical systems.

Another platform which always arises as an alternative in real-time systems is LabView from National Instruments which has its own machine vision portfolio. Here all the typical platform properties mentioned above are built-in. LabView is considered as a good candidate, but is not chosen due to the following reasons:

- Not out of the box support for multiple USB cameras which was used in the early testing stage.
- No previous developer acquaintance with LabView.
- Implementing custom algorithms gives no advantage compared to any other programming languages.

Platform conclusion

The above facts and uncertainties makes LabView a not so good option and Matlab even worse. A third alternative, Scorpion Vision Software is also an option, but it was discovered

too late in the development process to be a candidate. When the high-level platforms are thrown away, the fall back is to use general purpose programming languages. The most widespread general purpose programming language is C. And the widespread use means that there is a huge amount of libraries available to do common tasks. And as C is a unlicensed language there is a large driving force for open source development which usually means that user contributed help is readily available on the Internet.

Although libraries are very useful they do not replace an application framework. An application framework extensively controls the execution of the application and eases the development of threaded and event-based programming. C frameworks exist, but C++ code for standalone applications is preferred due to the object oriented paradigm. Example of two commercial C++ application frameworks are Qt (application and UI framework) and ICD's Control Design Platform (control system framework). Qt quickly arises as a very good alternative as it is both available for free download under the GNU LGPL license and in a commercial version if the application is to be sold. Therefore Qt is chosen as framework for this application. For the image processing part the open source library OpenCV is used. Qt and OpenCV are presented below.

Qt

Qt is a cross-platform application and UI framework for C++. Qt has a large UI library, takes care of multi-threading and event based programming which is important in UI programming. Being a cross-platform framework, Qt abstracts the operating systems underlying API into a generic C++ class interface which is the same across multiple platforms (Thelin 2007). Qt uses standard C++, but makes extensive use of a special pre-processor to add features like event based programming with little effort from the developer. Although Qt is portable this requires of course that any other library that the application uses must also be portable.

OpenCV

OpenCV is an open source computer vision library. The library comes with both C and C++ API. It is cross platform. Along with a large amount of machine vision algorithms OpenCV naturally have its own image and matrix format and provides the necessary manipulating functions for those formats. OpenCV lets the programmer access the low level functions which performs linear algebra operations on matrices. This includes for instance solving linear least square systems, finding eigenvalues etc which are useful also outside the machine vision discipline.

6.2 Overall Design Principle

Software frameworks exists partly because they help achieving a task with less code. Programming against a framework often puts the programmer's "best-practices" on test because

integrating with the framework requires throughout knowledge about the framework and about the special features that programming language provide.

A general design principle is to divide into separate parts which have a defined interface to other parts. An actual implementation of this concept is called object oriented programming and it is the main properties of C++ and Qt based programming. The design principle of this application is to divide it into appropriate classes and utilize the most out of Qt but not at the expense of lower performance. This means for instance that the OpenCV image format could be chosen as standard format instead of the equivalent Qt image format due to performance aspects even when interoperability with the framework would be easier with the Qt image format.

The next sections will go through the main modules and define the classes in each module and their most important interface towards the rest of the system.

6.3 Video Capture

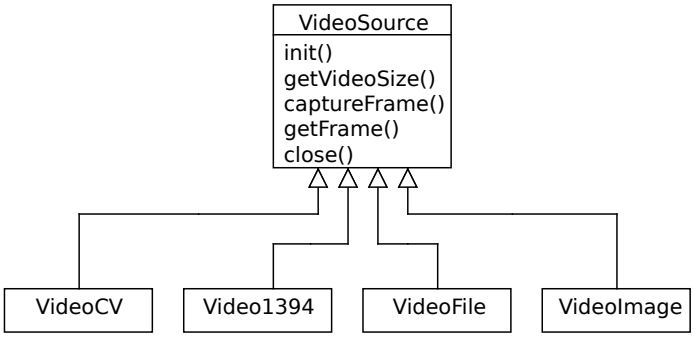


Figure 6.1: The interface class `VideoSource` and its descendant classes.

The task of the video capture part of the application is to retrieve frames from a device or from data stored on a hard drive. It is desirable to have multiple video inputs for the application to be able to test both real-time and pre-recorded video. To make the video interface independent of the video source, an abstract class `VideoSource` is created. To support a new input device a subclass which follows the interface of the `VideoSource` must be created. The following subclasses capture video frames from various sources and wraps the lower level interfaces into that of `VideoSource`. The specific implementation is covered more in detail in chapter 7.

- `VideoCV`: OpenCV video capture.
- `Video1394`: IEEE 1394 camera using libdc1394 (covered in 7.2).
- `VideoFile`: Video file reader using FFmpeg (covered in 7.2).
- `VideoImage`: Static image used for every frame in the video stream.

6.4 Image Processing

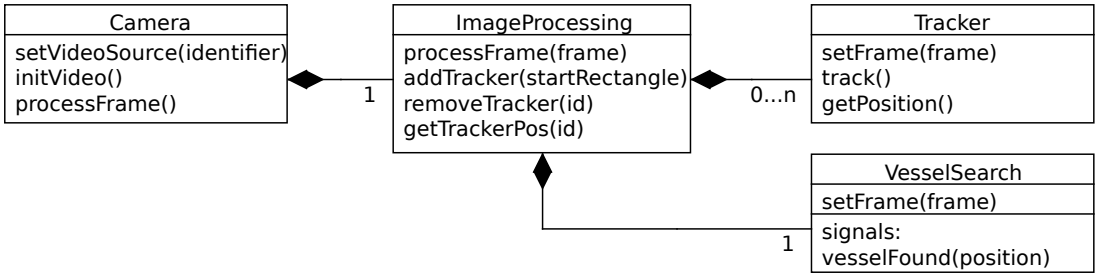


Figure 6.2: The image processing classes which work with the actual image pixels.

The image processing parts of the application deals with operations which are performed *per* camera. That is operations that work with the actual image pixels. The functionality is divided into four classes. The `Camera` class represents the instance of one camera. The camera class' main function is to convert the frames from the video source to correct internal format and forward them to the `ImageProcesing` class. It also keeps tracks of the camera matrix associated with the camera.

The `ImageProcesing` class controls any number of `Tracker` classes where each instance tracks a single point in the video. Tracker instances may be removed automatically by the `ImageProcessing` class if a `Tracker` instance does not pass the error detection tests from 4.2. The tracking initialization algorithm from 4.1 is implemented in the `VesselSearch` class. The `ImageProcessing` class forward the video images to the `VesselSearch` class when told so by external request.

Tracker class

An instance of the `Tracker` class tracks one point in the video stream using the cross-correlation method from 4.2. The tracked points are added by external request through the `ImageProcessing` class by supplying an initial start rectangle. The start rectangle defines the target representation which is captured from the current video image. The template stays unchanged throughout the trackers life (until it is deleted).

VesselSearch class

This is the tracking initialization algorithm described in 4.1. A single search is started by external request. When or if the vessel is found the relevant modules gets notified.

6.5 Position Estimator

This module takes care of the actual position estimation based on the tracked pixel values as described in chapter 5. This module keeps track of multiple groups of corresponding points in different cameras by the help of two structures `TrackerGroup` and `TrackerRef`.

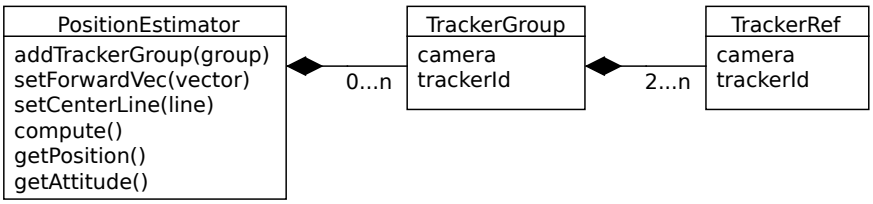


Figure 6.3: The position estimator class and its internal structures.

6.6 Tracking Control

By the term "tracking control" means the scheduling and decision making part of the application. The ultimate task for this module is to control the other modules such that vessel measurements are available as much of the time as possible. To simplify the design and more easily make working system the design is based on the tracking data from two cameras only. The two cameras must be placed like a stereo vision couple as described in 4.4. From now on let the left camera be called the *primary camera* and the right camera the *secondary camera*.

State machine design

The tracking control module follows a state machine based design and the chosen states for the state machine are based on the state of tracking.

The initial state is the state when there are no information yet found from the vessel position. Let this state be called `SEARCHING` as it means that system is searching for a vessel. In this state the `VesselSearch` algorithm is started immediately in the *primary camera* to find the vessel. When the vessel is found this marks the transition to a new state called `FINDING_AFT`. Since only the port and starboard sides of the vessel are found from the `VesselSearch` algorithm, the aft edge must be found by other means. The fact that trackers which are located *on* the vessel are more likely to be stable than the ones placed on the sea, can be a good test to find an approximation of the aft edge.

The `VesselSearch` algorithm gives out polygons counting from aft to stern direction and port to starboard. By adding trackers in the low range of polygons some trackers are likely to hit the sea behind the aft edge and others are likely to hit the vessel close to the aft edge. The trackers trying to track the sea will most likely fail after a short period of time because of too low correlation match. The trackers that are left are the ones located on the vessel. Finding the aftest tracker of those is done by finding the tracker which is inside the polygon with the

lowest number. All other tracking points except the aftest tracking point are removed as they are only used to find the aft edge.

Next, correspondence to the aftest point must be found in the *secondary camera* such that it can be verified that the aftest tracking point is not something that is only visible in one camera. This is important as the 3-D position of the aftest point is used later.

When the correspondence for the aftest tracking point is found this marks the transitions to a state called `ADDING_TRACKERS` which is suppose to create a tracking constellation which provides good 6 DOF tracking. This is again done with the use of the polygons from the `VesselSearch` class. This means that vessel search algorithm must be run to add new tracking points. The additional points are added with the aftest tracker and its surrounding polygon as a reference. This is to have better control of where the trackers will be located. For instance not placing them behind the aftest tracker. This means that if the aftest tracker fails in this state the state machine reverts back to `SEARCHING`. A simple algorithm makes sure that the trackers makes up for a good constellation by trying to establish trackers on both port and starboard sides and both in the aft and front areas of the cargo deck. In this state correspondence in the second camera is added continuously to the added trackers once they have proven to be stable for some a predefined time.

When enough trackers *with* correspondence has been added in the `ADDING_TRACKERS` 6 DOF estimation is possible given that an estimate of the body origin and the body attitude has been given to the `PositionEstimator`. If that is true this will lead to a transition to the `IN_TRACKING` state. In this state 6 DOF motion is estimated each frame by the `PositionEstimator`. Trackers are still added in this state to improve tracking in case trackers fails. All this is done while the position estimator provides continuous measurements, but if the system looses too many trackers, the state reverts back to `ADDING_TRACKERS`. In this state the aftest tracker is not used as reference for adding new trackers anymore as this would make the whole tracking fail in case the aftest tracker would fail. Instead a virtual point which was saved when the 3-D position of the aftest tracker was known is used. The 3-D point is projected on to the primary camera as a virtual tracker whenever the position of the original aftest tracker in the video image needs to be known. If less than three points are available in this state the state machine reverts back to `SEARCHING` as 6 DOF estimation is not possible and the vessel position is unknown.

Figure 6.4 show the states and the most important transitions and below follows summary of the most important actions which are performed in each state.

- `SEARCHING`: On entry all trackers are removed. `VesselSearch` algorithm is run to find vessel. Multiple trackers added to the aft.
- `FINDING_AFT`: All except the aftest trackers is removed. Tries to find stereo correspondence for the tracker.
- `ADDING_TRACKERS`: Trackers are added using the aftest tracker as reference point.
- `IN_TRACKING`: Trackers are added as needed to ensure good measurements. `PositionEstimator` gives continuous measurements.

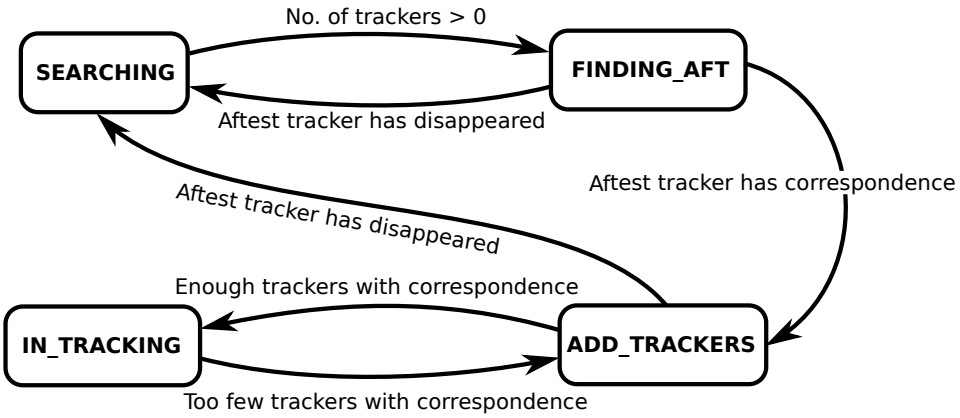


Figure 6.4: The states and most of the transitions.

The logic of the state machine are implemented in the class `TrackingManager`. The class needs access to all the `Camera` instances for tracking information and access to the `PositionEstimator` instance for knowing the 3D positions. The logic functions are run with the function `check()` for every frame.

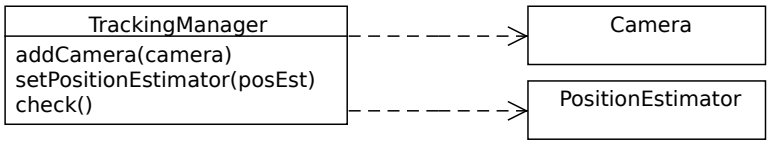


Figure 6.5: The `TrackingManager` class and the classes it needs access to.

6.7 Concurrent Computing

By dividing the application into multiple threads the operating system's scheduler will automatically assign threads to available cores. The performance benefit of using threads depends on the parallelization possibilities of the target hardware (number of cores, hyper-threading etc.) and whether it is possible to parallelize the computations. The computations in this application can be analyzed on a per-frame basis. The reason for this is that tracking data from each camera must be captured at the same time. Tracking data from the previous frame cannot not be used together with data from the current frame. Any threads must *at least* be synchronized at each frame. The interesting part to analyze is then the intra-frame computations.

Figure 6.6 depicts a one-frame capture and the suggested solution using threads. Each box symbols a thread. The circular object symbols a synchronization point for which all threads much reach before the execution can continue. From left to right there are first n video sources with n tracking threads working in parallel. No information needs to be exchanged between the tracker threads. To compute 3-D positions from the tracked positions in the video frame, information from all camera are needed because the computations uses corresponding points in *all* cameras. The execution must wait for all the trackers threads to finish. This is explained

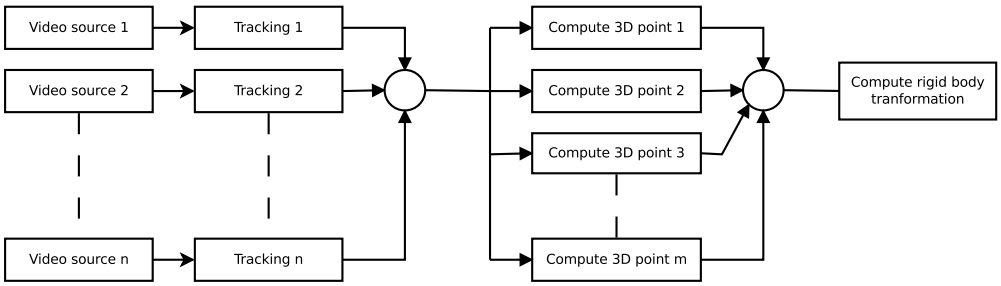


Figure 6.6: A one-frame computation using threads. Boxes represent code executing in a thread. Circles are synchronization points.

by a synchronization point in the figure. Let there be m correspondent point groups. Then there will be m threads which computes the 3-D position from the pixel positions. Finally all the 3-D positions are needed when the rigid body transformation is computed, requiring another synchronization point. There are computations inside the actual tracking algorithm which can be parallelized, like the cross-correlation algorithm which are computed for each tracker, but this is not taken into account in this analysis.

Threads

A slightly simpler design than the one described in the previous paragraph will be used as basis for the implementation. The simplification is that only a single thread computes all the 3-D positions instead of having one thread for each 3D position to be computed. There will still be one thread per camera such that the tracking algorithms can be run in parallel.

In Qt the main thread that starts the program is also the UI thread. If the image processing algorithms were to be controlled from UI thread, user interaction could stall the algorithms and the UI responsiveness would be dependent on the state of tracking. This is not desirable, therefore a separate controlling thread called `CameraManager` is needed.

The `CameraManager` thread makes sure that all cameras are in sync and processes frames taken at the same time. The position estimation and tracking managing is also run from this thread to make sure that the position is computed exactly every frame. The relation between these classes are shown in figure 6.7. The thread support is provided through Qt's `QThread` class such that the `CameraManager` class and the `Camera` inherits this class.

The dotted lines in figure 6.7 mark dependencies. First the `CameraManager` class of course needs to know about the cameras and it needs to know about the `TrackingManager` class and about the `PositionEstimator` class such that code in those classes can be executed by the `CameraManager`. Additionally both the `TrackingManager` and the `PositionEstimator` class needs to know tracking information from all the cameras. The order the classes are called will be: `processFrame()` for every `Camera` instance, `compute()` in `PositionEstimator` and finally `check()` in `TrackingManager`.

The summarize the application will be consisted of the following threads:

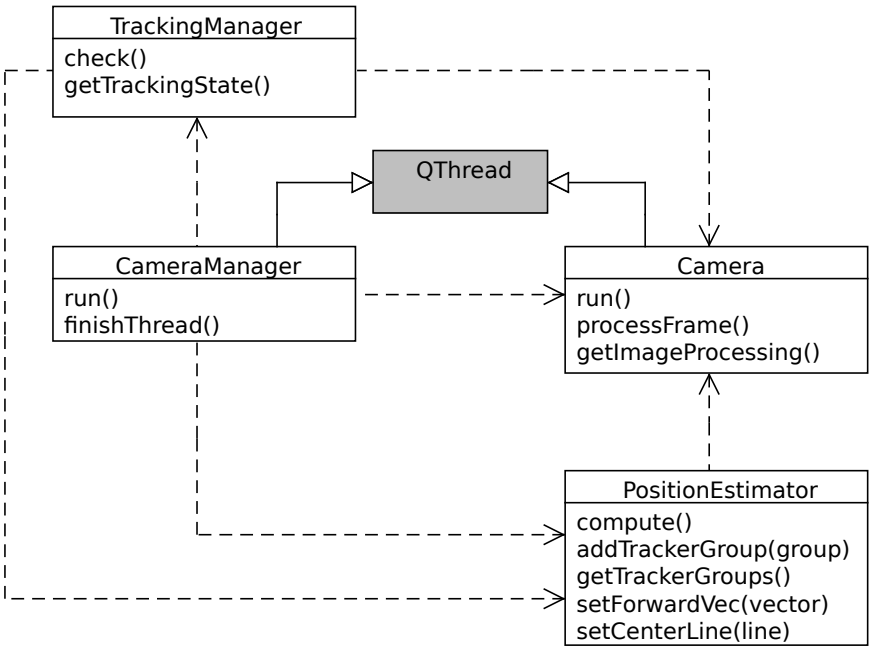


Figure 6.7: The main processing classes showing the relevant public methods.

- Main/UI thread
- `CameraManager` thread
- `Camera` thread (one per camera)

6.8 Video Display

Displaying general graphics in Qt is very easy and implementing custom graphics requires minor effort in the object oriented world of Qt and C++. Two video displays are shown for each camera to visually help debugging the image processing algorithms. One is the source video from the cameras, the other is a filtered version which shows either the tracker rectangles (ROI and patch) or the Canny edge detection output and the Hough lines.

All graphical items, including the custom video item, are placed in a canvas area which is called `QGraphicsScene` and the canvas is shown in a widget which is called `QGraphicsView`. Items which belong together are grouped using `QGraphicsItemGroup`. Figure 6.8 shows the graphical layout and figure 6.9 shows the class hierarchy.

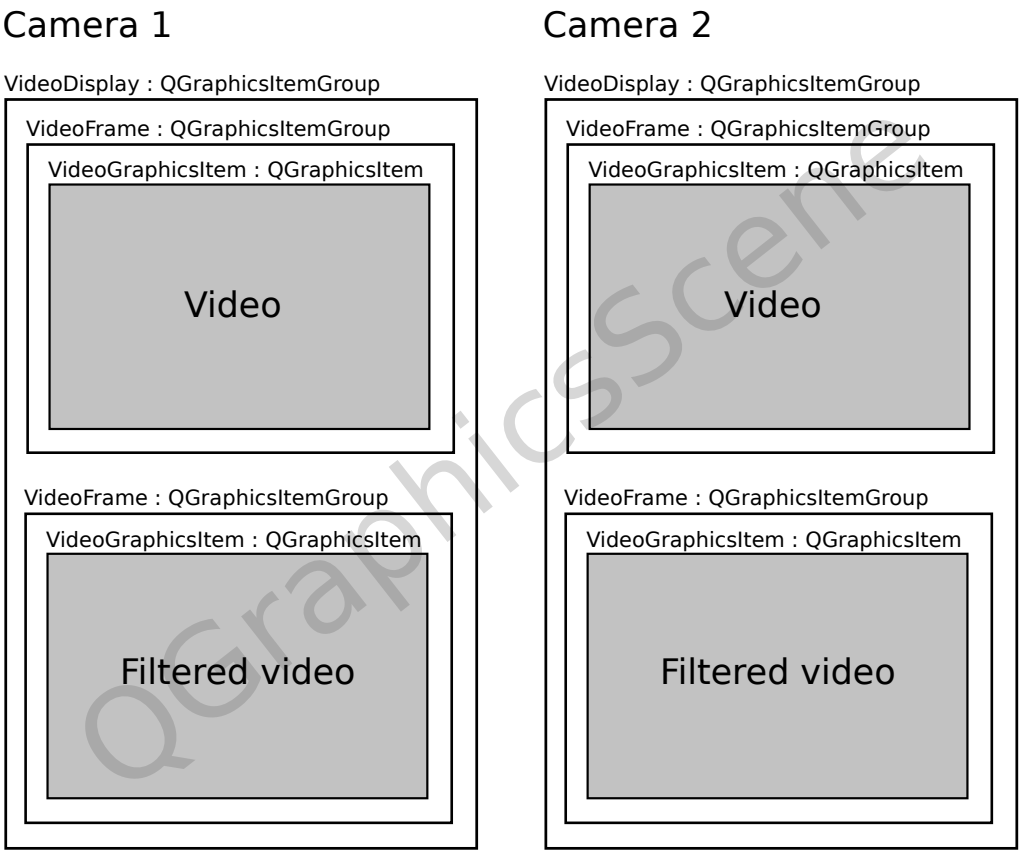


Figure 6.8: The GUI hierarchy.

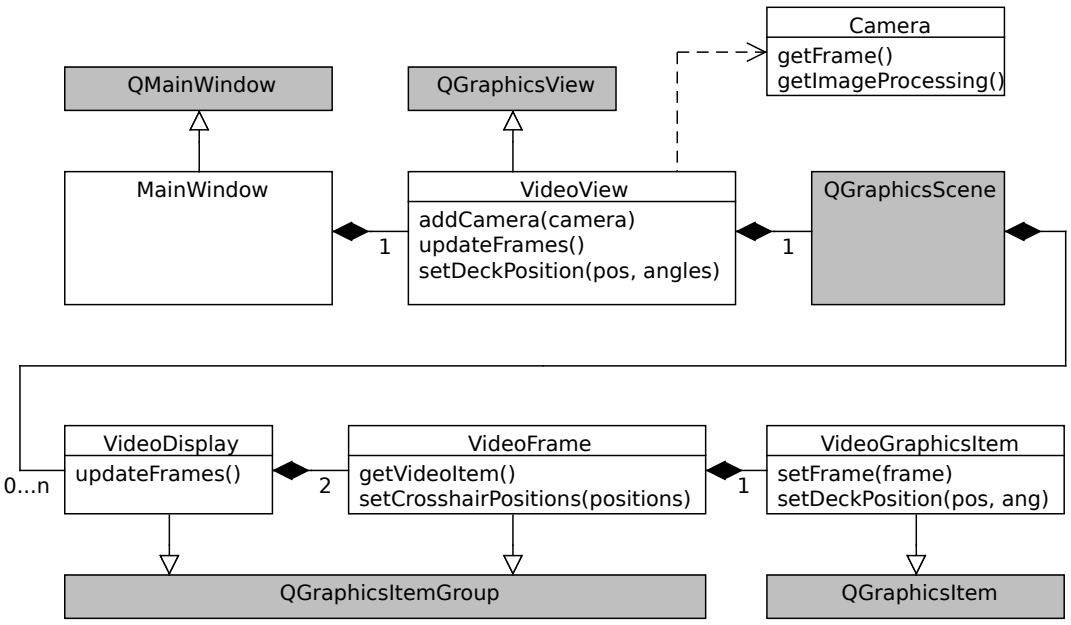


Figure 6.9: The UI hierarchy for the video displaying part.

Chapter 7

Software Implementation

The software implementation is used in the testing scenarios presented in chapter 8. This chapter presents tools used to implement the application explains implementation specific considerations.

7.1 Development Tools

Qt comes bundled with an integrated development environment (IDE) called QtCreator which is basically an advanced text editor specialized for writing Qt application. It includes among other things a graphical tool for creating UI and a graphical front end for the GNU debugger. As with all most textual programming languages one is not restricted to a particular editor, but the small add-on features which Qt adds to C++ requires an extra step in the build process and that is automatically taken care of by QtCreator. Additionally, creating UIs using only text editor is possible, but not very efficient.

The primary development took place on a PC with Ubuntu Linux version 9.10 (also known as "Karmic Koala"). The reason for choosing a Linux based operating system instead of Microsoft Windows is that the libdc1394 library does not yet support Windows. The alternative would be to buy drives from a commercial vendor which was considered a more cumbersome solution at the time.

7.2 Library Utilization

This section describes where the different libraries are used in the application.

Figure 7.1: The logos of the different frameworks and libraries used for implementing the application.

Qt

Except being the application and UI framework, Qt of course provides access to independent classes which some operation very compactly written in the source code. The `QVector2D` and `QVector3D` are used in `PositionEstimator` class wherever 3-D vectors needs to be represented. `QVector2D` is used extensively in the `VesselSearch` for constructing the vessel polygons.

In plain C, traditional arrays either have constant length or must be dynamically allocated by the programmer. Qt provides a `QVector`-class (not to be mixed with `QVector2D` and `QVector3D`) which can hold an array of any type of object by the use of C++ templates. This puts the dynamic memory allocation problem out of the way. This class is used almost every where in the application.

OpenCV

The main video frame format for the application is chosen to be the OpenCV `IplImage` format. The use of this format makes conversion between other formats very easy. And since many of the image processing algorithms are done using the OpenCV library, no conversion is needed.

OpenCV provides its own capture interface. On Windows this interface can capture any DirectShow device connected, which typically includes at least any connected web-camera. This feature was used in the early stage of testing when USB cameras were used. The OpenCV capture interface is used in the wrapper interface class `VideoCV`.

The OpenCV library is used extensively in the `Tracking` class for the cross-correlation based tracking. OpenCV provides a function `cvMatchTemplate(...)` which can use different block matching techniques including cross-correlation. For the sub-pixel accuracy computation interpolating the ROI is done using `cvResize(...)`. To find good tracking points the function `cvGoodFeaturesToTrack(...)` is used.

In the `VesselSearch` class OpenCV is used for the canny edge detection (`cvCanny(...)`) and for the Hough-transform (`cvHoughLines2(...)`).

OpenCV provides functions for performing linear algebra operations on matrices. In the `PositionEstimator` class `cvSolve(...)` is used to compute 3D-positions and `cvSVD(...)` which performs singular value decomposition is used to compute rigid body transformation.

OpenCV provides an API for storing matrices. It is used when the user explicitly wants load camera matrices from `.xml`- or `.yml`- files.

libdc1394

libdc1394 is an API to access firewire cameras which adhere to the IIDC (Instrumentation & Industrial Digital Camera) standard. Since IIDC cameras follows a strict standard they do not need a vendor specific driver to function. The library does only support Linux and Mac. This library is used in the wrapper class `Video1394` and in the `Sys1394` class.

FFmpeg

FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video. The interesting part for this application is the `libavcodec` and `libavformat` libraries which together provides functionality for reading recorded video streams in almost any format. FFmpeg is used in the wrapper class `VideoFile`.

7.3 Multithreading in Qt

Qt provides extensive thread functionality. In Qt the main thread which starts the application is always the GUI thread. Additional threads are created by extending the `QThread` class. Any code which are executed from the `run()` procedure are executed in that thread. Functions in a `QThread`-enabled class called from another thread is executed by the calling class and not by the thread which the `run()`-function is running in.

Qt of course provides cross-platform thread synchronization primitives such as mutual exclusion, semaphores and wait conditions. Such primitives are used in every class which can be accessed from multiple threads where the simultaneous access could lead to a race conditions (unpredictable result). Table 7.1 shows which threads that must be thread aware.

The rest of the classes involved in tracking and position estimation are typically either accessed by only one thread or is shielded from multiple access by another class. For instance the `TrackingManager` is only accessed by the `CameraManager`-class except some small calls from the UI-thread which does not pose any risk. And the `Tracker`-class is shielded by the `ImageProcessing`-class.

Preventing data corruption due to simultaneous access is done with the `QMutex`-class which provides mutual exclusion functionality. The thread synchronization property needed by the `CameraManager` and `Camera` classes to synchronize the cameras, is provided through a combination of `QMutex`, `QWaitCondition` and `QSemaphore`. See the source code for details on how it works.

Thread	Accessed by
`CameraManager`	• Itself when telling each `Camera` object to start processing the next frame. • The *UI thread* when adding or removing `Camera`-objects and when starting or stopping camera capture.
`Camera`	• Itself when processing each frame. • The `CameraManager`-thread when starting and stopping this class' thread.
`ImageProcessing`	• The `Camera`-thread when it tells this object to process the current frame. • Indirectly by the `CameraManager`-thread through the `PositionEstimator`-class and through the `TrackingManager`-class which e.g. fetches tracker positions from this class.
`VesselSearch`	• Indirectly by the `Camera`-thread through `ImageProcessing`-class when the next frame should be analyzed. • The `UI`-thread when it wants to display data regarding vessel search in the UI.
`PositionEstimator`	• The `CameraManager`-thread when it tells this thread to compute the next position and attitude estimation. • The `UI`-thread when it wants to display position data in the UI.

Table 7.1: Classes which are accessed from more than one thread and which threads they are accessed from.

7.4 Video Display

The use of multiple high resolution cameras requires copying and processing of large amounts of data per time. Without efficiency in mind it is very easy to create bottlenecks in the system. This section explains two concerns that must be addressed in an implementation. One is handling of video data within the application and other is displaying video on screen efficiently.

Video frame handling

OpenCV, Qt, OpenGL and libdc1349 all define their own image format and converting between them is necessary and must so be done wisely. Most image formats used for video are based on C-`structs` which have various properties and a pointer to a memory area where the actual image pixels are located. There exists various formats of the pixel memory. The important part is to avoid copying the complete video frame data using `memcpy(...)` whenever possible as this is a performance bottleneck when dealing with large image sizes.

The OpenCV image format `IplImage` is chosen as the main image format for the application such that no conversion is needed when using OpenCV image processing functions. The default pixel format of OpenCV is "24-bit BGR" which means that pixels are stored with one byte per channel in the order blue, green, red. When reading video files with FFmpeg, conversion is usually needed anyway since different video streams use different pixel formats. In this case the conversion is done internally in FFmpeg with the "sws_scaler"-sub library of FFmpeg such that the destination format becomes "24-bit BGR" from the beginning. Creating an OpenCV image is used *without* copying the pixel data, but by using the original pixel memory location as source. This is done in the following way:

```
1 // Create frame header without allocating pixel storage
2 IplImage * frame = cvCreateImageHeader(cvSize(videoWidth, videoHeight), ↩
      IPL_DEPTH_8U, 3);
3 // Set the frame header to point to the same data as the FFmpeg frame
4 // pFrameBGR->data[0] is the FFmpeg-frame's pixel data
5 frame->imageData = (char *) pFrameBGR->data[0];
```

Conversion between the other image formats is done in a similar way. More advanced format checking must be done if the conversion should work in all cases, but only a minimum of code was implemented to make the conversion work. Since the IIDC cameras used in this project are gray scale it is irrelevant whether the pixel format is RGB or BGR since the values are equal, this opts for a simpler conversion or no conversion at all.

Accelerated video display

Displaying high resolution video streams while at the same time rescaling it to fit inside an application window could easily pose a performance bottleneck. This could result in that the

CPU processing cycles which were meant to be used by image processing algorithms could be used to display the video image only. The problem of displaying high resolution video is today of course already solved by the consumer multimedia PC industry. The problem is a matter of choosing an appropriate method for displaying the video on screen. Some widely used technologies are presented in the table below:

Name	**Description**
DirectX	An API for the Windows operating system for handling video display and many other multimedia related functionality.
XVideo	A video output extension for the X Window System. The X Window System is a software application which provides graphics almost exclusively used on Linux systems.
SDL	A cross-platform API for displaying video and other multimedia related functionality.
QtOpenGL	A module which comes bundled with Qt for displaying graphics using the OpenGL API.

It is a natural choice to use the QtOpenGL module since Qt is already chosen as application framework. Using an included package requires minimum work to integrate the video display into to the rest of the application UI. It may be a surprise to use OpenGL for showing video, but in fact many video players use OpenGL for displaying video. OpenGL is a cross-platform API which makes it possible to perform operation such as resizing of the video on the graphics processing unit which ease the load on the CPU.

The video display is implemented using the OpenGL "Pixel Buffer Object" feature which amongst other things uses direct memory access to transfer the pixel data from system memory to the graphics card. The actual implementation is very API intricate and further details are found in the source code.

Augmented Reality Cargo Deck

A virtual cargo deck was projected on top of the video image. This visual feature shows that the system is tracking the movement of the vessel. The cargo deck was rendered using OpenGL which already is used to render the video. The main challenge is to match the OpenGL world to match the world defined by the camera matrix. OpenGL internally use two matrices to define the mapping between x,y and z and screen coordinates. This is similar to the camera matrix defined in chapter 3, but the conversion is not straightforward. To do the conversion a function called `argConvGLcpara2(...)` was borrowed with some modifications from a software library called "ARToolKit" (Augmented Reality Toolkit) (ARToolKit 2010). The library is free for non-commercial use.

7.5 OTC Specific Implementation

OTC is an acronym for "Offshore Technology Conference" and is the world's foremost event for the development of offshore resources in the fields of drilling, exploration, production, and environmental protection. OTC is held annually at Reliant Center in Houston (OTC 2010). A part of this project was to create a demo application to be displayed at OTC representing proof-of-concept of new technology from National Oilwell Varco. This section covers work that were specific for the OTC exhibit.

Purpose

In the scope of OTC this application represents new technology which is to be considered either as an add-on feature to already existing offshore cranes or as a feature for new cranes delivered by NOV. The purpose of the demo application is to visually proof that the movement of a supply vessel is captured by cameras and that the position is measured in real-time. The advantage of knowing this information is explained in 1.1.

OTC user interface

When dealing with crane operations the heave movement is the most important measurement. Special functionality was implemented to compute derivative heave data. The main functionality was the following:

- Dividing the cargo deck into a grid of 11 $\times$ 5 squares.
- Plotting the heave measurement for a user selectable square.
- Storing the heave measurements for the last 40 seconds for every square.
- Estimating the peak-to-peak measurement for each square.
- Visualizing the estimated peak-to-peak for every square using a color coded grid.

The position of any point on the cargo deck can be found from the position of the cargo deck and the roll, pitch and yaw relative to the n-frame. Position of an arbitrary position on the cargo deck is given by

$$\mathbf{p} = \mathbf{p}_{nb}^{n} + \mathbf{R}_{b}^{n}\mathbf{p}_{sq}^{b} \tag{7.1}$$

where $\mathbf{p}_{sq}^{b}$ is a vector describing the position on the deck relative to the center of the deck. In this application it is pointing to the center of any of the 11 $\times$ 5 squares which are defined.

Estimating peak-to-peak heave

For estimating the peak-to-peak amplitude many approaches exists. For example it is possible to register multiple minimum and maximum points and compute the difference between them for some time period. A simpler method without much logic was used and it will be briefly presented here.

Two values are computed. The first value is constrained such that it is always equal or larger than the heave measurement. If the heave measurement is less than the previous value, the value follows the heave measurement with a slow first order response. The same computation applies for the second variable, but it is constrained such that it is always equal or less than the heave measurement. The estimated peak is found by the difference between the maximum and minimum value. An example estimation is found in figure 7.2. The algorithm i similar to how a single phase rectifier behave.

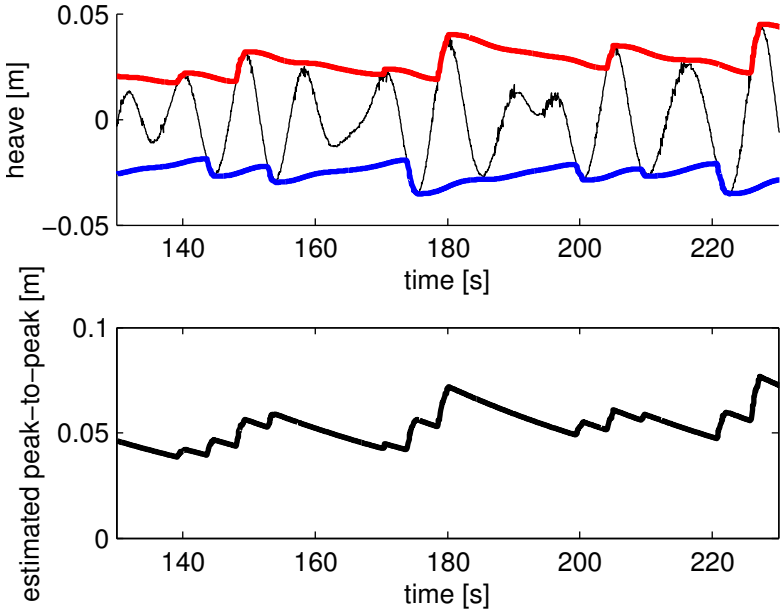


Figure 7.2: The workings of the peak-to-peak estimation method. The lower plot is the difference between the red and the blue curve in the upper plot.

Placement of Body Origin

Two simplifications were done in the placement of body frame:

- Since the yaw angle could not change and was known in advance, it was forced to a specific value.
- A special "check image" was used to position the body origin. It was based on cross-correlation by searching for the "winch only" template and position the cargo deck relatively.

These two simplification improved the appearance of the augmented reality cargo deck.

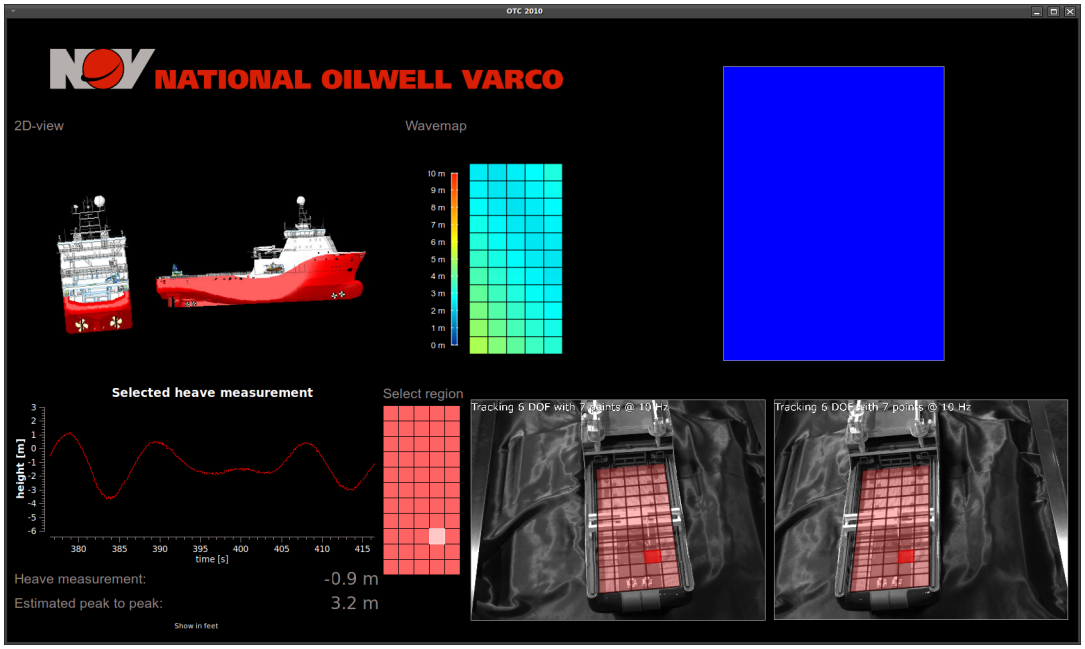


Figure 7.3: The final UI used at OTC. The blue rectangle originally showed the third camera (which was mounted in a 90° angle), but the screen shot was taken at a later occasion with only two video streams available. The heave values were multiplied by 100 such that it would match the real vessel.

Chapter 8

Experimental Setup

An important part of this project is the verifying of algorithms through the software implementation and a hardware setup. This chapter covers the purchased hardware which was used during the experiments. Additionally the complete experimental setup is described an important part is the motion platform which was built to simulate waves.

8.1 Application Hardware

This is the hardware that application uses when it is running. It consists of the processing unit and the cameras. State of the art hardware was chosen to make sure that lack of processing power or camera resolution would be a performance issue.

Processing Unit

Architecture	Intel x86
CPU	Intel Core i7 960/3.2 GHz (quad core)
Memory	12 GB
Video Card	Asus Radeon HD 5770
IEEE 1394b interface	2 x FireBoard-800

Figure 8.1: Processing unit specifications.

Manufacturer	Sony
Model	XCDU100E
Sensor chip	Grayscale, 1600×1200 pixels
Frame rate	15 Hz at 1600×1200 pixels
Color depth	8 bit/16 bit
Interface	IEEE 1394b (Firewire 800 Mbit/s)

Table 8.1: Camera specifications.

The processing unit is a high performance Intel-PC with two IEEE 1394b interface cards. Two cards are needed because the data rate for three cameras exceed the 800 MBit/s limit on one card. The third camera are not used in machine vision algorithms, but only for visual display.

The processing unit was installed with the chosen Ubuntu version and the necessary libraries and the application software were installed.

Cameras and lenses

Figure 8.2: "Sony XCD-U100" camera fitted with a "Fujinon HF16HA-1B" lens. Also showing the two digital connectors at the back allowing daisy-chaining of multiple cameras.

The cameras are of industrial type with uncompressed transfer of image data to the host computer. Gray scale cameras was chosen because the color information is really not needed and the picture sharpness of gray scale camera tend to be somewhat better since only one sensor element is needed per pixel. The most important factor when choosing lens is the focal length. Higher number means more zoom as illustrated by figure 3.1. The focal length was chosen such that the cameras captures at least the whole cargo deck at the target distance.

The cameras were run at their maximum resolution of 1600 × 1200 pixels and maximum frame rate of 15 Hz.

Manufacturer	Fujinon
Model	HF16HA-1B
Focal length	16 mm
Focal/Iris control	Manual

Table 8.2: Lens specifications.

8.2 Scenario

The experimental setup consists of two cameras, the processing unit, a motion platform and a scale model of a supply vessel. The distances in the scenario was placed such that it approximately is a scaled down version of the supply vessel rig scenario. Two cameras were used and they were mounted approximately 30 cm apart. The distance from the cameras to the motion platform was between 2 and 3 meters.

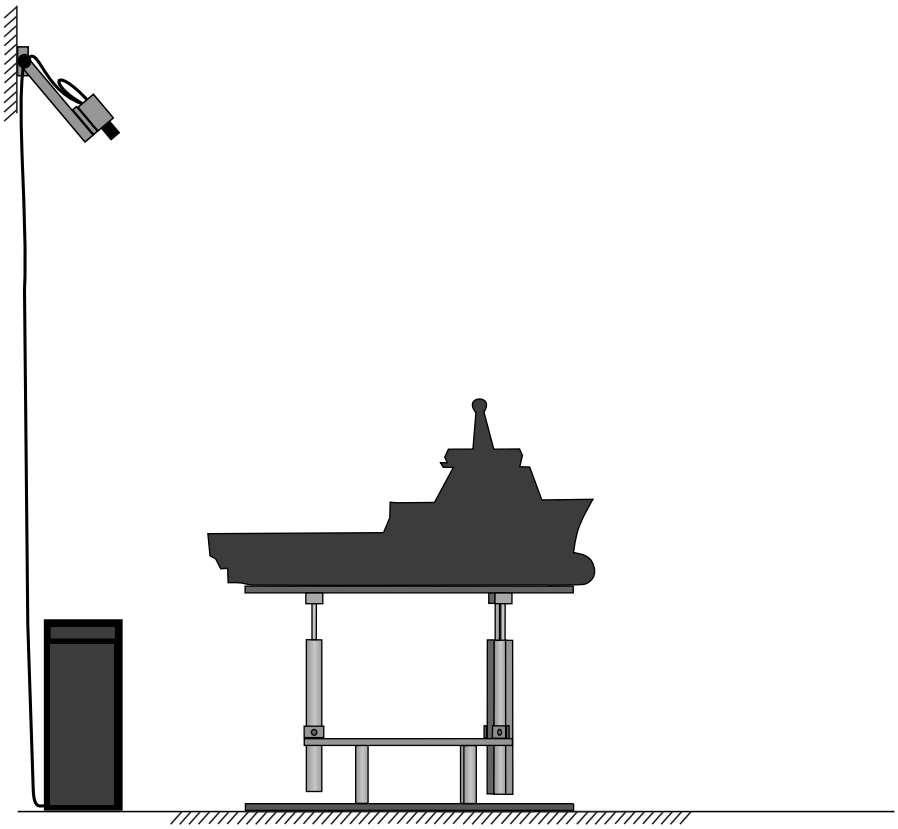


Figure 8.3: The scenario used in experimental results.

Scale Vessel Model

The tracked object is an accurate scale model of a real supply and anchor handling vessel from Siem Offshore Inc. The vessel name is "Siem Emerald". The model was rented from the

Figure 8.4: Scale model of Siem Emerald.

Norwegian Siem offices by National Oilwell Varco for use in this project.

Motion platform

The motion platform is a 3 DOF platform (roll, pitch and heave) designed by National Oilwell Varco and built by the mechanics at department of Engineering Cybernetics, NTNU. It consists of a base with three linear actuators with stepper motors and encoders. The actuators motor controls and power supply are manufactured by Festo. The electric drive makes the actuator ideal for following arbitrary smooth movements like waves.

The actuators are evenly spaced around a circle such that the angle between them are 120°. The actuator bottom mount is a hinge bearing and the actuator end point is mounted to the motion platform (the top plate) with ball bearing. The stepper motor control is performed by the supplied motor control box.

To simulate realistic wave movements, a supply vessel model from the Marine Systems Simulator (MSS) library for Simulink/Matlab (Marine Systems Simulator 2010) was used. In particular the "DP Force RAO model - zero speed" with the parameters "ShipX: Supply Vessel (L = 82.80 m)" from MSS Hydro was used. This setting suits well for the Siem Emerald as it is 91 meters long. The zero speed DP model also suits well considering the crane operation scenario.

Multiple wave/current scenarios was simulated and recorded, but only one data set is used in chapter 9. The data set used has 6 meters significant wave height, with the default settings except: "Torsethaugen" wave spectrum, zero current and wave direction straight ahead. The significant wave height was chosen based on typical limit where a crane operation must be canceled due to heave seas.

The center of rotation is approximately at center of the motion platform and the vessel scale model is located on top of the platform. This does not correspond to the center of orientation used by the vessel model in MSS. This mismatch is neglected since using the vessel model is only a way to get realistic wave movement to some extent. The yaw, x and y measurements are also neglected since the platform can only simulate pitch, roll and heave.

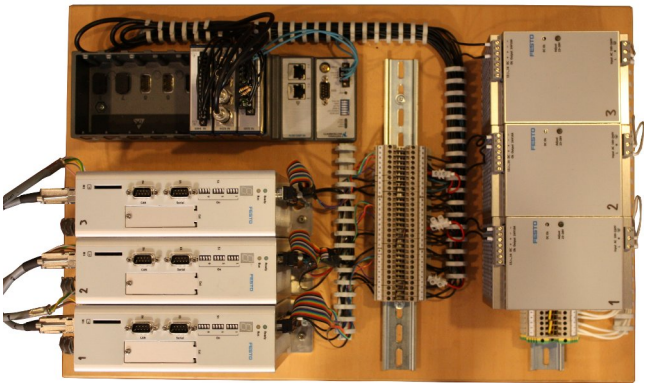


Figure 8.5: The devices involved in the actuator control. Built and programmed by Pål Jacob Nessjøen.

The overall control of the actuator positions is done using a National Instruments CompactRIO system which is programmed and deployed by Pål Jacob Nessjøen. The roll, pitch and heave data from MSS is converted to appropriate actuator positions in real-time by the CompactRIO module. Due to various limitations and uncertainties in the control loop of the actuators it is not certain that the actuators follows what would be the perfect pitch, roll and heave from the data set. Therefore the *actual* actuator positions are measured by the built in encoders and saved for later analysis. The real roll and pitch angles are computed afterwards using the forward kinematics of the motion platform which is derived in A.

Figure 8.6: The motion platform designed by National Oilwell Varco and built by mechanics at Department of Engineering Cybernetics, NTNU.

Scale	1:100
Length	91 cm
Width	22 cm

Table 8.3: Essensial scale model data.

8.3 Camera Calibration

A cube was used as calibration rig. The cube was placed at the back of the supply vessels deck. Before calibration the motion platform and vessel was placed with all the actuators half way extended. This corresponds to the mean values of roll, pitch and heave. The coordinates of the cube corners was chosen such that the x-axis points forward, the y-axis to starboard and the z-axis down. This makes it easy to compare the vessel movement measured by the camera to platform movement because the coordinate system of the platform is chosen the same way.

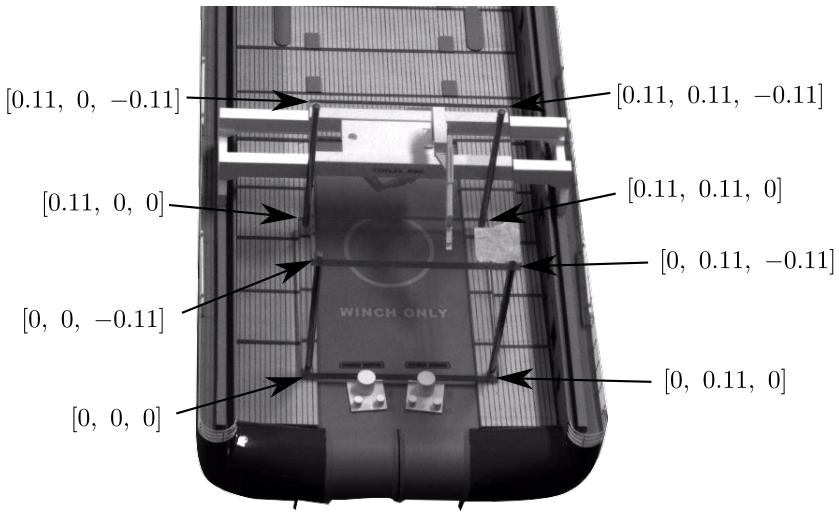


Figure 8.7: Camera calibration cube placed on cargo deck.

8.4 OTC Demo setup

The companies attending OTC are designated to separate areas called booths. This project was assigned to a small area of the NOV booth. The setup is the same as described in 8.2, except that an additional third camera was used to display an overview of the vessel. No tracking information was used from this camera. It was only used for augmented reality.

Actions were taken to make the setup to look more visually appealing than the original experimental setup. This included covering the motion platform with a steel box with the NOV logo on it and creating a silk seabed to cover the top of the box. Additionally a TV-screen was used to display the camera video and the visualization of the measurements. A specifically designed UI was displayed at this screen. The main UI was not displayed at all except at a hidden administration screen. A separate touch screen computer was used to control the motion platform. It was important that this system was separated from the machine vision system to make it clear that the measurements were computed from the camera videos only.

Figure 8.8: The stand at OTC showing the motion platform, the TV-screen and the PC to control the wave platform.

Chapter 9

Experimental Results

The experimental results focuses on the measurement accuracy of the application when used in the setting described in the previous chapter. Primarily the position and attitude measurements from the application are compared to the real data. The real data is found by using the stored actuator measurements and computing the motion according to the derivations in A. Aspects regarding tracking stability is also discussed.

9.1 Source Data

The video analysis in this chapter was performed off-line to easier test different schemes with the same source. The data was recorded at OTC as this proved to be a successful setup. The experimental setup explained in 8 is still valid. The data set for the motion platform was run in loop and the video capture was started approximately 30 seconds before the start of a new loop of the data set. Approximately 400 seconds of video was recorded at 15 frames per second. The syncing of video measurements and platform measurements was done based on a manual timing light in the video which indicates when the platform data set starts. This is not accurate enough to measure latency, but it is used to roughly align the measured and the motion platform measurements.

The data sets used in this chapter are given in the following table:

	Contents	**Sample rate**
Mesaured platform movement	Position, attitude	25 Hz
Video mesaurement, regular	Position, attitude, number of points, tracking state	15 Hz
Video mesaurement, 1/3 sub-pixel	Position, attitude, number of points, tracking state	15 Hz
Video mesaurement, 1/6 sub-pixel	Position, attitude, number of points, tracking state	15 Hz

9.2 Measurement Results

In the following sections the first data set *without* sub-pixel tracking is compared unless otherwise specified.

Figure 9.1 shows the comparison for the first 250 seconds position part of the data set. The plots also shows how many 3-D point correspondences which are tracked at any instant. During the initialization phase the measurements are far off, but stabilizes when the default number of 3-D points is reached which is six in this case.

The measurements position are correlated with the real position, but clearly there are offsets. Since the offsets are constant it is likely to believe that the application is not measuring the correct point. The "real" position is chosen to be the center of the cargo deck. The position chosen by the application may easily be off center and can result in both scale and offset errors.

Figure 9.2 shows the attitude measurement given by roll, pitch and yaw. The first thing to notice is that the roll measurement is a lot more noisy than the pitch measurement. This is because the point spread in longitudinal direction is much larger than in the lateral direction. Simply because the vessel is longer than it is wide. Noisy points which are close together will have a much greater influence on the computed angle than points far apart.

Surprisingly the real yaw angle is not zero although the motion platform was designed to be a roll, pitch and heave platform only. This is due to the exact kinematic model derived in A.1 which takes the small actuator angle deflections into account. However, it is not much correlation between the real and measured yaw values. The measured yaw value fluctuates between around $\pm 1°$. It is likely to believe that this fluctuations is due to to a body frame chosen by the application which is not perfectly aligned with the vessel longitudinal/lateral directions. Misaligned body frame will not measure the desired angles perfectly, but rather a composite of the two other rotations as well.

The angle measurements have little offsets partly because of the filtering algorithm described in 5.4.

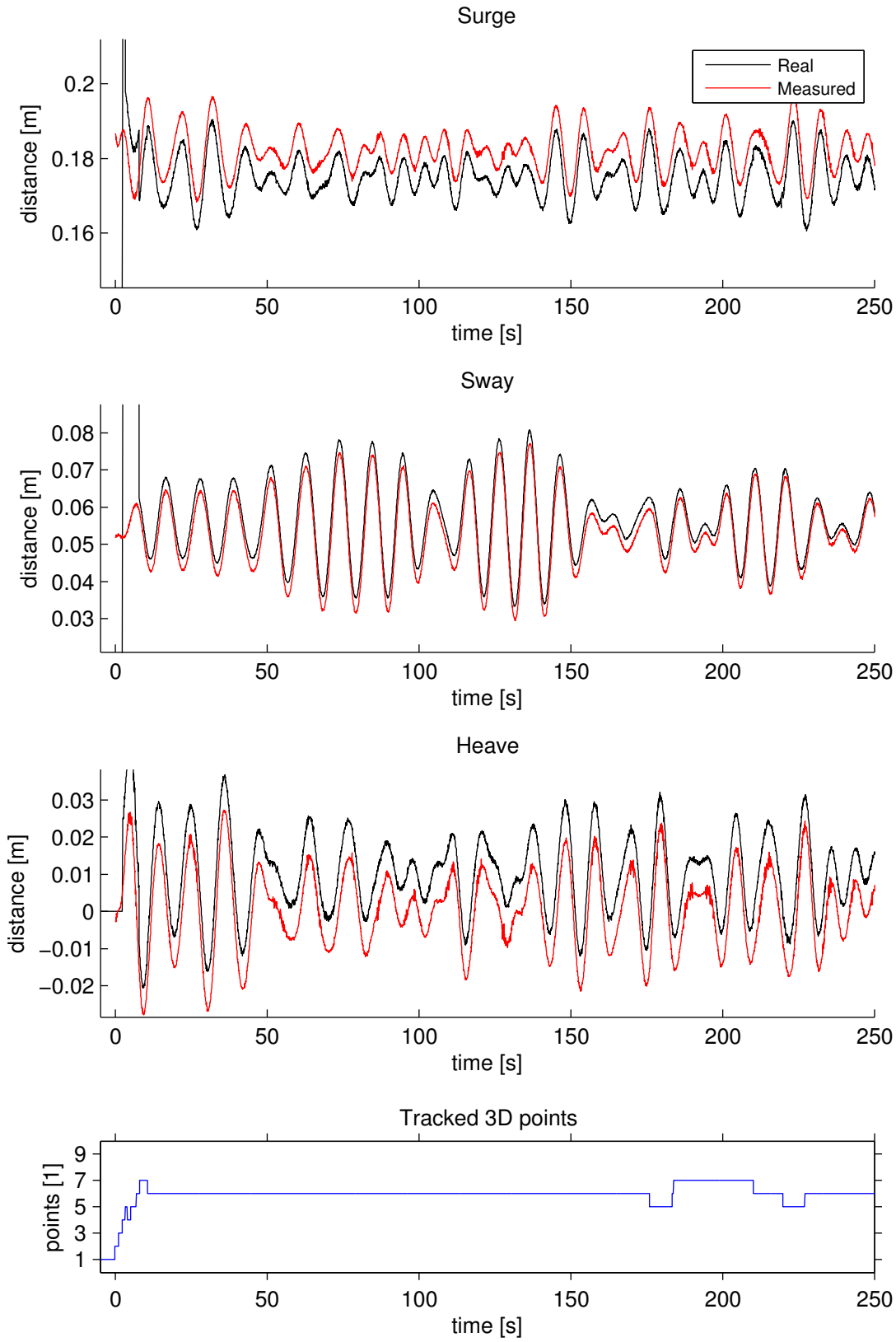


Figure 9.1: The surge (x), sway (x) and heave (x) positions of the center of the cargo deck. Measured by the application (black) and computed from the motion platform movement (red). Number of tracking points is also showed (blue).

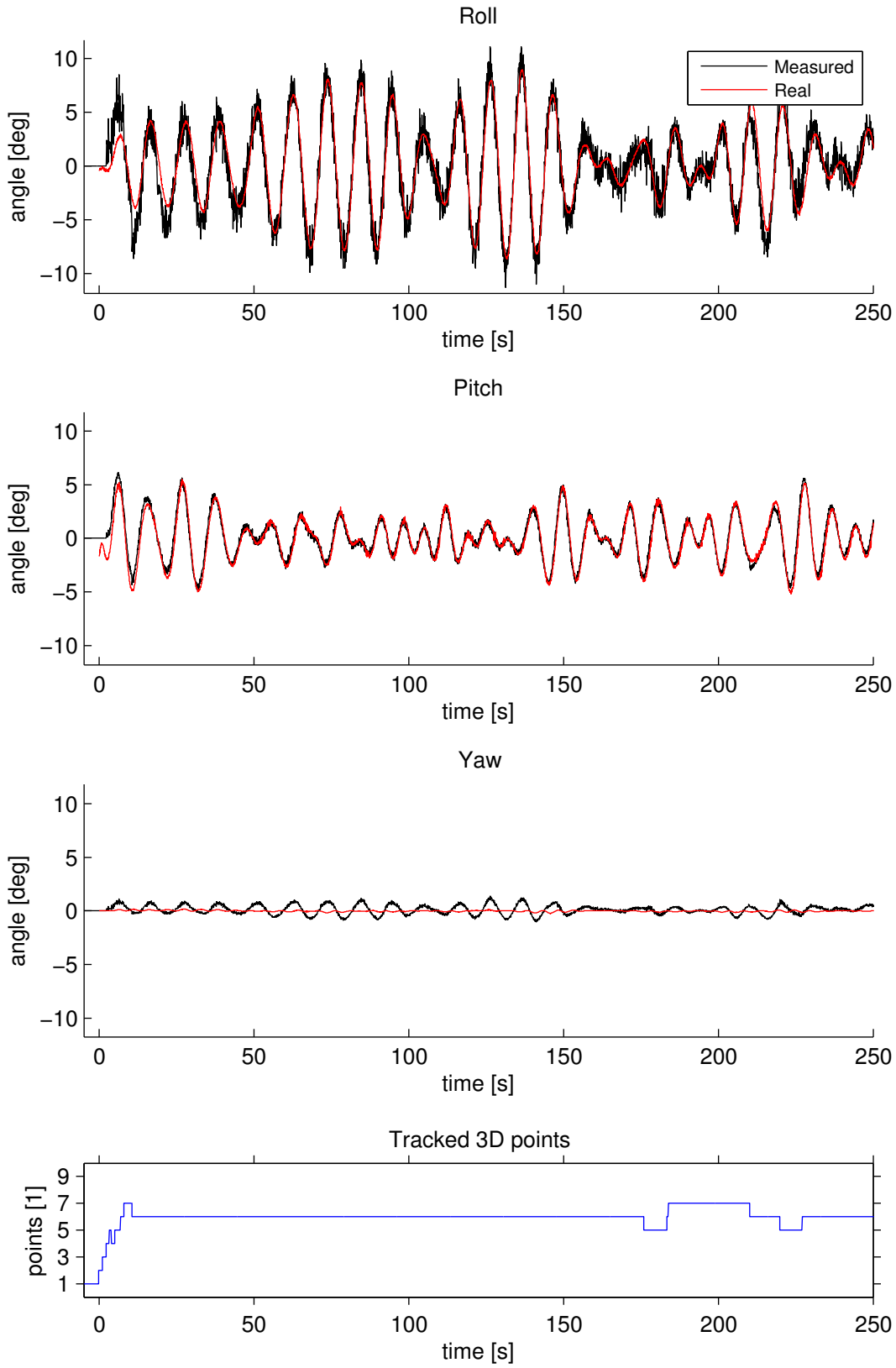


Figure 9.2: The roll (ψ), pitch (θ) and roll (ϕ) angles measured by the application (black) and computed from the motion platform movement (red). Number of tracking points is also showed (blue).

Loss of tracking point

In figure 9.3 a situation where a tracking point is lost and the roll measure clearly gets affected, but after about 20 seconds it gets back on track due to the mean filtering.

The removal of tracking point is based on two error checks: too large acceleration or too low correlation. The acceleration check computes the discrete acceleration and naturally needs at least three frames. This means that when the acceleration is too large the point may already have moved and affected the rigid body estimation permanently. It is possible to compensate for this, but this would add more latency in the system.

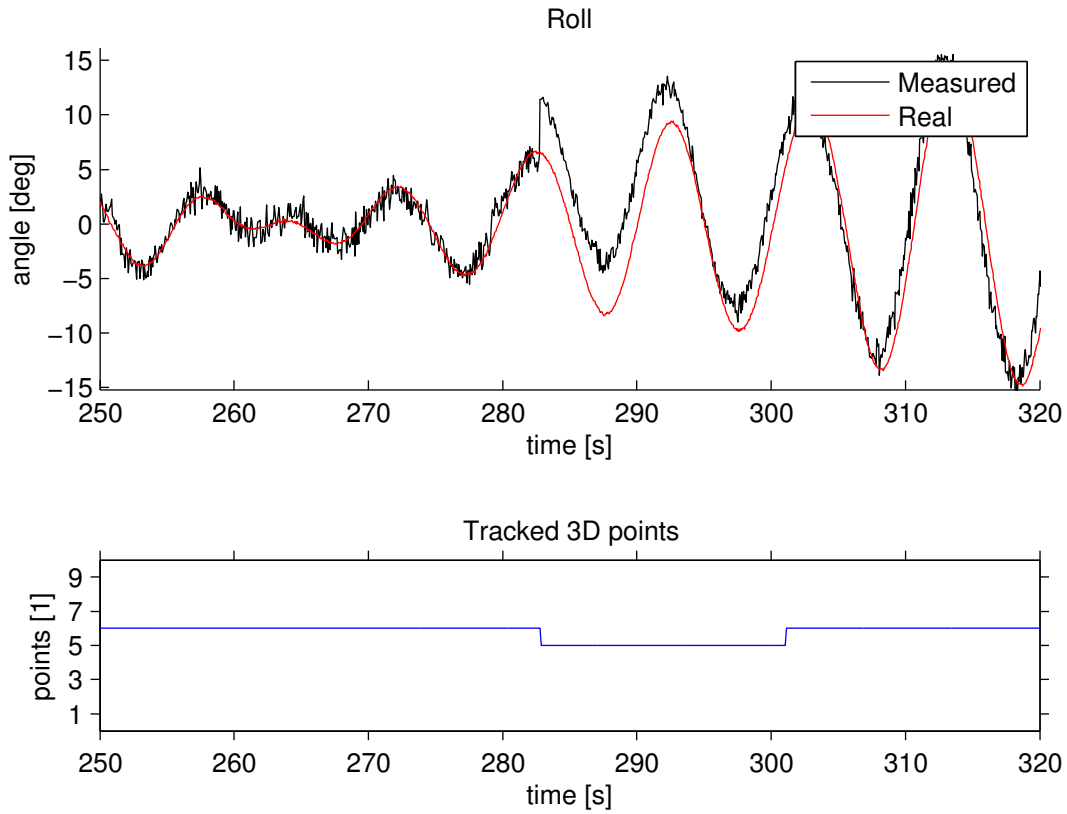


Figure 9.3: Measurements where a tracking point drifts before it is removed.

Sub-pixel tracking

Both 1/3 pixel 1/6 pixel tracking was tested. Figure 9.4 shows the comparison between regular tracking and subpixel tracking. The roll measure was chosen since it is often the most noisy measurement and it is therefore easy to see improvements. Already with 1/3th pixel tracking the roll measurement is greatly improved.

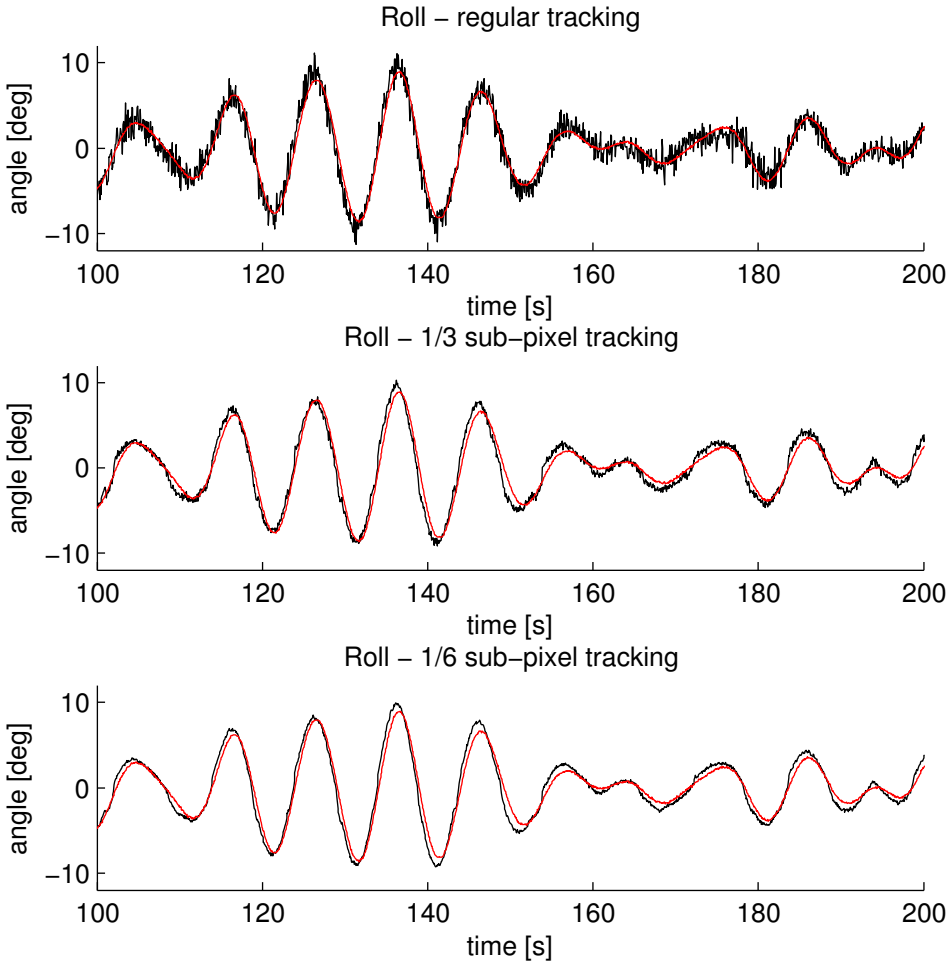


Figure 9.4: Roll measure comparison for regular and sub-pixel tracking.

	Regular	1/3 Sub-pixel
Front port	79.51%	98.9%
Front starboard	60.86%	95.22%
Aft port	68.75%	97.73%
Aft starboard	79.88%	99.58%

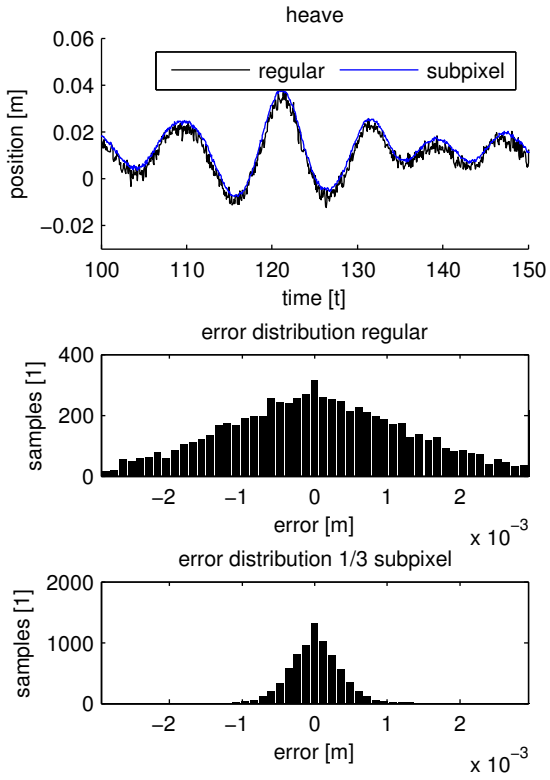
Table 9.1: Amount of samples that are within a threshold of 0.001 m for each corner and for each tracking type.

9.3 Heave Precision Analysis

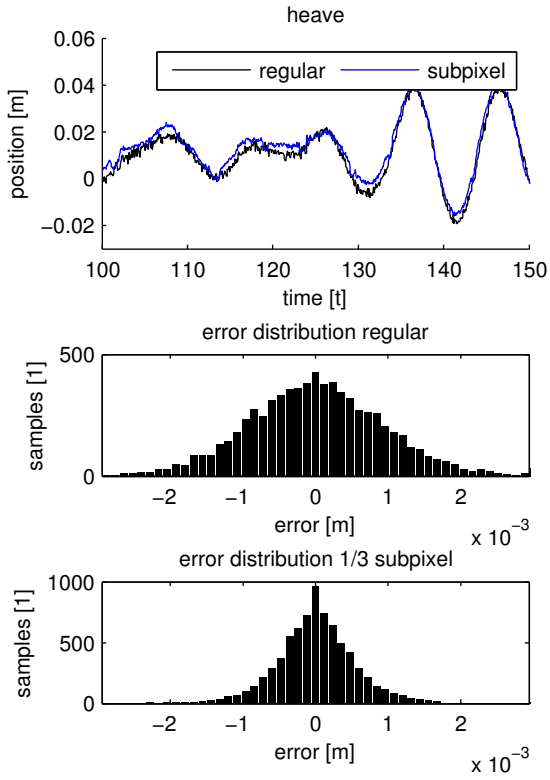
There are different ways to define precision. A common approach is to relate it to the probability distribution of the signal. In this section an accuracy a will be defined as the limit where 95% of the measurements are the mean value $\pm$ the accuracy a.

To find the precision of the measurement it is in this case not appropriate to subtract the corresponding motion of the motion platform due to the geometric uncertainties described in 9.2. Additionally the measurements from the actuators on the motion platform itself also have measurement noise which will influence the result with undesired factors. Another means of finding the accuracy is to assume that the measurement consists of a low-frequent part which is the vessel motion and a high frequent part which is measurement noise. To analyze the high-frequent part a high pass filter which filters away the vessel motion is used. It is assumed that frequencies above 1 Hz is noise.

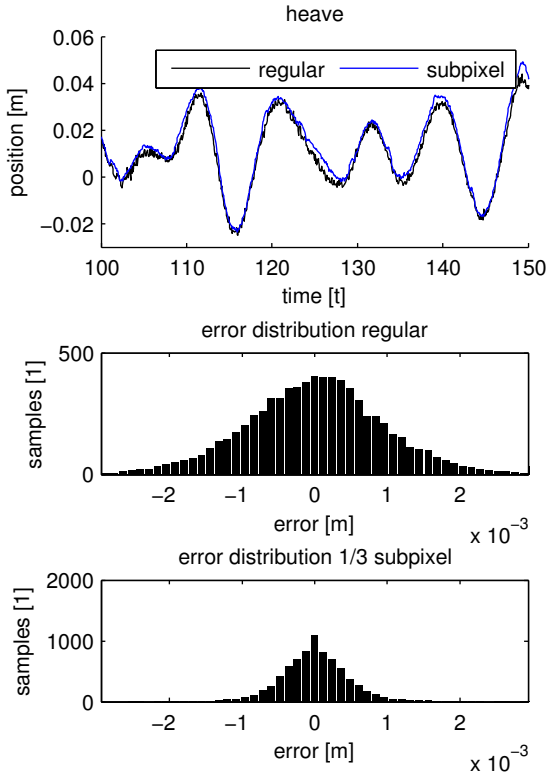
To catch the worst case measurements all the four corners of the cargo deck are analyzed. Figure 9.5 shows a small part of the heave measurement just to show the measurement noise with and without sub-pixel tracking for each cargo deck corner. The 1/3 pixel tracking is chosen for the sub-pixel variant. The histograms shows the signal distribution after the low frequency parts has been filtered away. Additionally the figure displays two histograms for the complete time series of approximately 400 seconds for sub-pixel and regular tracking. Table 9.1 shows the accuracy in terms of number of samples which are within 0.001 meters on each side of the mean. This number is chosen because it corresponds to an accuracy of 0.1 meters for the real vessel which is the requirement specified in 2.2. Both regular tracking and sub-pixel tracking with 1/3th of a pixel precision is showed in the table. It is clear that sub-pixel tracking improves the measurements quite drastically.



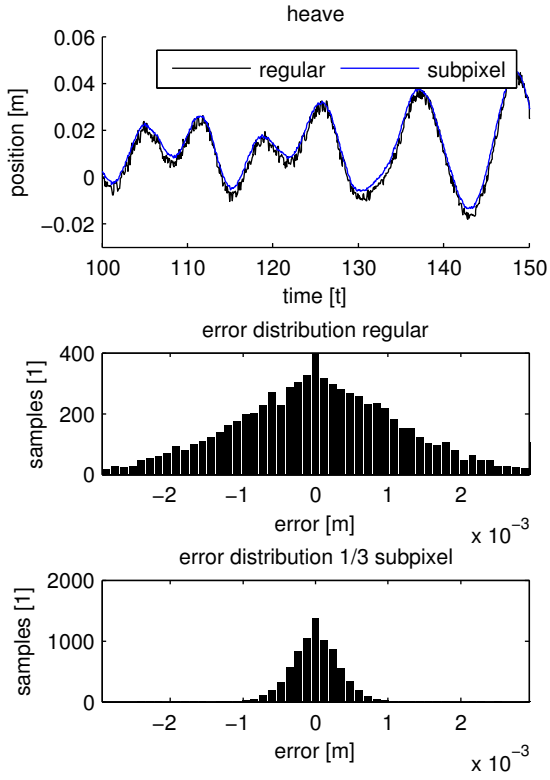
(a) Front port cargo deck corner.



(b) Front starboard cargo deck corner.



(c) Aft port cargo deck corner.



(d) Aft starboard cargo deck corner.

Figure 9.5: The heave measure and error distribution with and without sub-pixel tracking for each corner of the cargo deck.

9.4 Real-time performance

The system was run in real-time at OTC for four days with very little down time. The experimental results are based on off-line processing. This of course leads to another result than with real-time tracking, but the difference lies mainly in frame rate. Real-time processing leads to varying frame rate whereas off line processing may process without any frame dropping. Sub-pixel tracking was not used ad OTC because regular tracking with slight low-pass filtering gave sufficient result. The frame rate was seldom below 10 Hz.

When dealing with real-time systems, non-deterministic behavior usually lies in the scheduling of processes done by the operating system. The internal communication and data transfer in the application is known. It is when external devices needs to be accessed that unknown latency may appear. In machine vision systems the external devices are cameras.

A rough measure was done to find the latency from the real world to the camera display on screen. The measure showed approximately 0.5 second delay. It was not investigated further what this delay could be a result of. It is suspected that it is related to the buffering scheme of the libdc1394-library. The library uses an internal ring-buffer for each camera to store the frames. If the user application does not read the ring buffer fast enough the buffer fills up an frames get dropped. This typically happens every time the vessel search algorithm is run. Slight frame dropping is not really a problem, but because the `CameraMananger`-thread restricts the application from exceeding the frame rate of 15 Hz, the application will never catch up the lag caused by the earlier slowness. The result is that the buffer will always be filled and there will be a constant lag depending on the buffer size.

This is a design flaw of the application which is very easily fixed by increasing or removing the processing rate of 15 Hz. This way the application can catch up when there is little work to do. If the limit is removed the maximum processing rate is only limited by the camera acquisition rate or the computing power.

9.5 Vessel Independence

The system was created with vessel independence in mind, except the few requirements stated in 2.1. The system was not tested with other ship models so it is hard to give proof of vessel independence. The three most critical algorithms are the tracking initialization algorithm which needs to know the with of the vessel. Secondly the polygons which are used for aiding the selection of tracking points must be placed in regions which there exist good features to track on the vessel. And finally the stereo correspondence algorithm is dependent of cameras that are accurately calibrated and that the cameras' views are not too different. However, the tests experience of the Siem Emerald model showed that in most cases it is just a matter of tuning scalar variables to make everything work.

Chapter 10

Conclusions

A prototype system has been built an implemented for real-time performance. The 6 DOF motion of a supply ship in medium and high seas was successfully measured by the machine vision system. The accuracy of the measurements is characterized by offsets, but are generally very correlated with the real positions. It is discussed that these errors are due to misplacement of the body coordinate frame onto the vessel. This misalignment pose no practical measurement error since the operator selects the position to measure on the vessel by visual confirmation. The system proved to sustain tracking throughout the complete data set of ten minutes with six meter significant wave height. The system has therefore succeeded in keeping tracking for more than one container lift.

Sub-pixel tracking clearly leads to better tracking, but the implemented sub-pixel algorithm requires much more computation than regular tracking. Without improving the algorithm or the multi-threading properties of single point tracking, sub-pixel *may* not be possible to run in real-time. The accuracy of 1/3 sub-pixel tracking shows that the accuracy is within the requirement of ± 0.1 meter which is stated in chapter 2. Real-time observations showed that the sample rate was never below 10 Hz during regular tracking. Further investigation must be done to find out if this limit is maintained with an accuracy of ± 0.1 meter.

The experiments show that the vessel is automatically found by the custom derived line search algorithm. The loss of tracking points may distort the tracking for a short period of time and if the tracking is lost it is regained after a short period of time. The vessel independence feature has been discussed. It cannot be concluded that the system is completely vessel independent from the experimental results.

10.1 Future Work

There are a lot of aspects of the implemented application which can be improved. The most vulnerable algorithm is the tracking initialization part. Testing multiple vessels in the experimental setup was not done, and the vessel independence is one of the main features which

needs improvement. This is a complicated algorithm which must be intelligent to succeed. An alternative to sub-pixel tracking is model based filtering which also improves the accuracy.

The most important future work is the following:

- Model based filtering to improve accuracy.
- More intelligent tracking initialization.
- More advanced error detection.

Appendix A

Motion Platform Kinematics

Here the forward kinematics of the motion platform will be derived. This model is used to find the position and attitude of the motion platform given the three actuator lengths. Two coordinates frames are defined. The actuator frame $\{a\}$ is located at the center of the actuator plate. The x_a-axis points through actuator 1's hinge mount, the z_a-axis is pointing up, normal to the bottom plate and the y_a-axis such that a right handed coordinate frame is formed. The second coordinate system is attached to the motion platform with the origin at the center of plate. The x_p-axis is pointing from actuator 1's end point towards the center. z_p pointing down, normal to the plate and y_p such that a right handed coordinate frame is formed. Figure A.1 shows the defined coordinate systems.

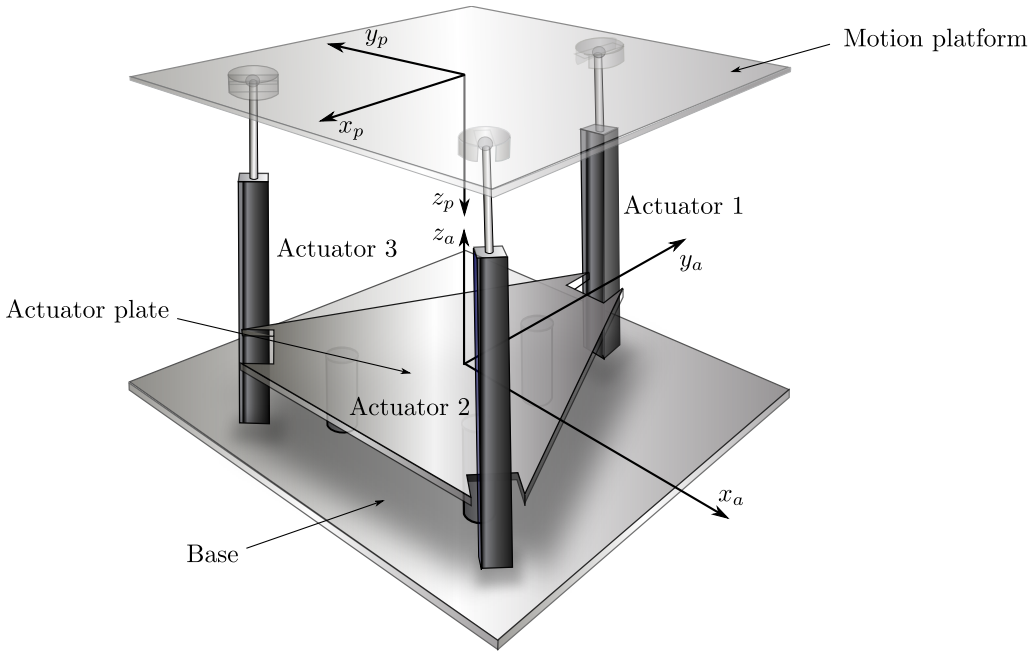


Figure A.1: The motion platform and its two coordinate frames.

The following derivation will find the relation between the $\{a\}$ frame and the $\{p\}$ frame. First

an expression for the position of the end points of the actuator $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ are derived.

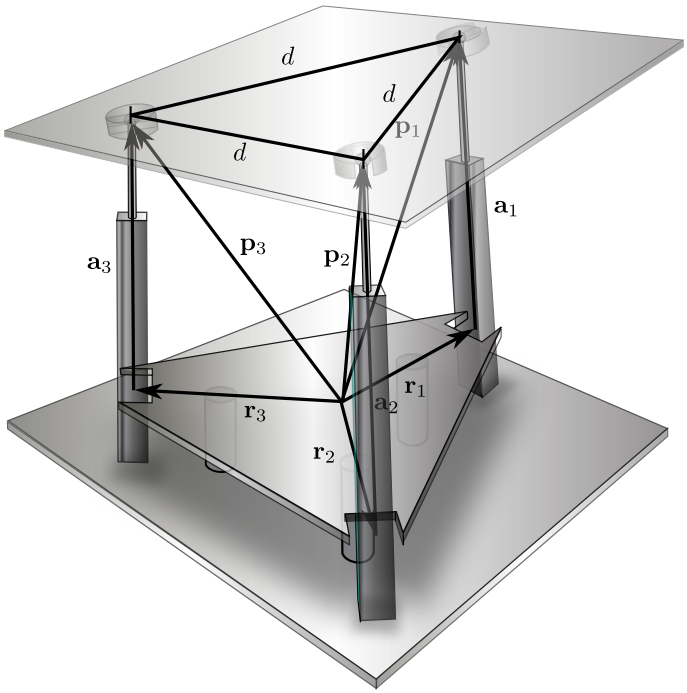


Figure A.2: The location of the vectors describing the fixed actuator hinge position $\mathbf{r}_i$ and actuator orientation and length $\mathbf{a}_i$.

Since each actuator is hinged to the actuator plate its working area is planar. The position of the actuator tip can therefore be done in a two dimensions only. Figure A.3 shows the working geometrics of an actuator in its planar working area. The hinge position of actuator i is determined by the vector $\mathbf{r}_i$ and its length is defined by the actuator hinge radius. Finally the actuator end point is determined by the angle of the actuator itself and its length.

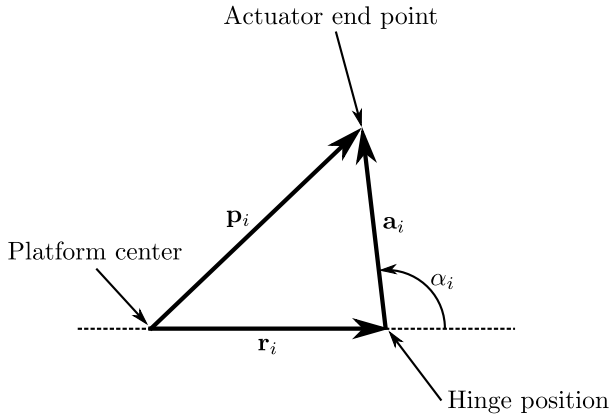


Figure A.3: Vectors for actuator top plate position i.

Actuator 1's working plane coincides with the yz-plane and has therefore the simplest expres-

sion of the actuator end point. It is given by

$$\mathbf{p}_1 = \begin{bmatrix} 0 \\ r + a_1 \cos\alpha_1 \\ a_1 \sin\alpha_1 \end{bmatrix} \tag{A.1}$$

The two other actuators are the same as actuator 1, except it must be rotated around the z-axis until such that the original yz-plane coincides with the working area of the actuator.

$$\mathbf{p}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ r + a_2 \cos\alpha_2 \\ a_2 \sin\alpha_2 \end{bmatrix} = \begin{bmatrix} \sin(\theta_2)(r + a_2 \cos\alpha_2) \\ \cos(\theta_2)(r + a_2 \cos\alpha_2) \\ a_2 \sin\alpha_2 \end{bmatrix} \tag{A.2}$$

where $\theta_2 = \frac{2}{3}\pi$ (120°)

$$\mathbf{p}_3 = \begin{bmatrix} \sin(\theta_3)(r + a_3 \cos\alpha_3) \\ \cos(\theta_3)(r + a_3 \cos\alpha_3) \\ a_3 \sin\alpha_3 \end{bmatrix} \tag{A.3}$$

where $\theta_3 = \frac{4}{3}\pi$ (240°)

A.1 Constraints

The constraint in this system is the motion platform which is attached to the actuator end points. There is only one possible position and orientation of the top plate for a given set actuator lengths. Fortunately the solution is not overdetermined meaning that actuators could work against each other. This will be shown mathematically later, however it is very intuitively correct that it must be like that. The motion plate defines constraints on the *distances* between every actuator end point. The constraints can be described by

$$\| \mathbf{p}_1 - \mathbf{p}_2 \|_2 = d \tag{A.4}$$
$$\| \mathbf{p}_1 - \mathbf{p}_3 \|_2 = d \tag{A.5}$$
$$\| \mathbf{p}_2 - \mathbf{p}_3 \|_2 = d \tag{A.6}$$

where d is the distance between two arbitrary actuator end points and is equal for all combination due to the symmetric mount position on the top plate. The unknowns in these equations are the three actuator angles α_i for $i \in \{1, 2, 3\}$.

Writing out (A.4) yields

$$\begin{aligned}(\mathrm{s}\theta_2(r+a_2\mathrm{c}\alpha_2))^2+(r+a_1\mathrm{c}\alpha_1-\mathrm{c}\theta_2(r+a_2\mathrm{c}\alpha_2))^2\\ +(a_1\mathrm{s}\alpha_1-a_2\mathrm{s}\alpha_2)^2=d^2\end{aligned} \tag{A.7}$$

$$\begin{aligned}\mathrm{s}^2\theta_2(r+a_2\mathrm{c}\alpha_2)^2+(r+a_1\mathrm{c}\alpha_1)^2-2(r+a_1\mathrm{c}\alpha_1)\mathrm{c}\theta_2(r+a_2\mathrm{c}\alpha_2)\\ +\mathrm{c}^2\theta_2(r+a_2\mathrm{c}\alpha_2)^2+(a_1\mathrm{s}\alpha_1-a_2\mathrm{s}\alpha_2)^2=d^2\end{aligned} \tag{A.8}$$

$$\begin{aligned}\overbrace{(\mathrm{s}^2\theta_2+\mathrm{c}^2\theta_2)}^{=1}(r+a_2\mathrm{c}\alpha_2)^2+(r+a_1\mathrm{c}\alpha_1)^2\\ -2(r+a_1\mathrm{c}\alpha_1)\mathrm{c}\theta_2(r+a_2\mathrm{c}\alpha_2)+(a_1\mathrm{s}\alpha_1-a_2\mathrm{s}\alpha_2)^2=d^2\end{aligned} \tag{A.9}$$

$$\begin{aligned}r^2+2ra_2\mathrm{c}\alpha_2+a_2\mathrm{s}^2\alpha_2+a_2^2\mathrm{c}^2\alpha_2+r^2+2ra_1\mathrm{c}\alpha_1+a_1^2\mathrm{c}^2\alpha_1\\ -2(r+a_1\mathrm{c}\alpha_1)\mathrm{c}\theta_2(r+a_2\mathrm{c}\alpha_2)+a_1^2\mathrm{s}^2\alpha_1-2a_1\mathrm{s}\alpha_1a_2\mathrm{s}\alpha_2+a_2^2\mathrm{s}^2a_2=d^2\end{aligned} \tag{A.10}$$

$$\begin{aligned}r^2+2ra_2\mathrm{c}\alpha_2+a_2^2\mathrm{s}^2\alpha_2+\overbrace{(\mathrm{s}^2\alpha_2+\mathrm{c}^2\alpha_2)}^{=1}a_2^2+r^2+2ra_1\mathrm{c}\alpha_1+\overbrace{(\mathrm{s}^2\alpha_1+\mathrm{c}^2\alpha_1)}^{=1}a_1^2\\ -2rc\theta_2(r+a_2\mathrm{c}\alpha_2)-2a_1\mathrm{c}\alpha_1\mathrm{c}\theta_2(r+a_2\mathrm{c}\alpha_2)-2a_1\mathrm{s}\alpha_1a_2\mathrm{s}\alpha_2=d^2\end{aligned} \tag{A.11}$$

Finally the terms are grouped by the unknowns α_1 and α_2

$$\begin{aligned}\mathrm{c}\alpha_1(2ra_1-2ra_1\mathrm{c}\theta_2)+\mathrm{c}\alpha_2(2ra_2-2ra_2\mathrm{c}\theta_2)+\mathrm{c}\alpha_1\mathrm{c}\alpha_2(-2a_1a_2\mathrm{c}\theta_2)\\ +\mathrm{s}\alpha_1\mathrm{s}\alpha_2(-2a_1a_2)+2r^2+a_1^2+a_2^2-2r^2\mathrm{c}\theta_2=d^2\end{aligned} \tag{A.12}$$

Realizing that $\mathrm{c}\theta_2=\cos(\frac{2}{3}\pi)=-0.5$ makes for some more simplification

$$\begin{aligned}\cos\alpha_1(3ra_1)+\cos\alpha_2(3ra_2)+\cos\alpha_1\cos\alpha_2(a_1a_2)+\sin\alpha_1\sin\alpha_2(-2a_1a_2)\\ +3r^2+a_1^2+a_2^2-d^2=0\end{aligned} \tag{A.13}$$

Writing out (A.5) follow the same steps as writing out A.4. Although a_2, α_2 and $\cos\theta_2$ must be replaced with a_3, α_3 and $\cos\theta_3$. The numeric value for $\cos\theta_3$ is the same as $\cos\theta_2$ because $\cos\theta_3=\cos(\frac{4}{3}\pi)=\cos(\frac{2}{3}\pi)=\cos\theta_2$ is the same.

Equation (A.5) becomes

$$\begin{aligned}\cos\alpha_1(3ra_1)+\cos\alpha_3(3ra_3)+\cos\alpha_1\cos\alpha_3(a_1a_3)+\sin\alpha_1\sin\alpha_3(-2a_1a_3)\\ +3r^2+a_1^2+a_3^2-d^2=0\end{aligned} \tag{A.14}$$

Writing out (A.6) is little different

$$\begin{aligned}
&\mathrm{s}^2\theta_2(r+a_2\mathrm{c}\alpha_2)^2 - 2\mathrm{s}\theta_2\mathrm{s}\theta_3(r+a_2\mathrm{c}\alpha_2)(r+a_3\mathrm{c}\alpha_3) + \mathrm{s}^2\theta_3(r+a_3\mathrm{c}\alpha_3)^2 \\
&+\mathrm{c}^2\theta_2(r+a_2\mathrm{c}\alpha_2)^2 - 2\mathrm{c}\theta_2\mathrm{c}\theta_3(r+a_2\mathrm{c}\alpha_2)(r+a_3\mathrm{c}\alpha_3) + \mathrm{c}^2\theta_3(r+a_3\mathrm{c}\alpha_3)^2 \\
&+a_2^2\mathrm{s}^2\alpha_2 - 2a_2\mathrm{s}\alpha_2 a_3\mathrm{s}\alpha_3 + a_3^2\mathrm{s}^2\alpha_3 = d^2
\end{aligned} \tag{A.15}$$

The term $(2\mathrm{s}\theta_2\mathrm{s}\theta_3 + 2\mathrm{c}\theta_2\mathrm{c}\theta_3) = -1$ when inserted for the values for θ_2 and θ_3

$$\begin{aligned}
&\overbrace{(\mathrm{s}^2\theta_2+\mathrm{c}^2\theta_2)}^{=1}(r+a_2\mathrm{c}\alpha_2)^2 - \overbrace{(2\mathrm{s}\theta_2\mathrm{s}\theta_3+2\mathrm{c}\theta_2\mathrm{c}\theta_3)}^{=-1}(r+a_2\mathrm{c}\alpha_2)(r+a_3\mathrm{c}\alpha_3) \\
&+\overbrace{(\mathrm{s}^2\theta_3+\mathrm{c}^2\theta_3)}^{=1}(r+a_3\mathrm{c}\alpha_3)^2 + a_2^2\mathrm{s}^2\alpha_2 - 2a_2\mathrm{s}\alpha_2 a_3\mathrm{s}\alpha_3 + a_3^2\mathrm{s}^2\alpha_3 = d^2
\end{aligned} \tag{A.16}$$

$$\begin{aligned}
&r^2 + 2ra_2\mathrm{c}\alpha_2 + a_2^2\mathrm{c}^2\alpha_2 + (r+a_2\mathrm{c}\alpha_2)(r+a_3\mathrm{c}\alpha_3) + r^2 + 2ra_3\mathrm{c}\alpha_3 + a_3^2\mathrm{c}^2\alpha_3 \\
&+a_2^2\mathrm{s}^2\alpha_2 - 2a_2\mathrm{s}\alpha_2 a_3\mathrm{s}\alpha_3 + a_3^2\mathrm{s}^2\alpha_3 = d^2
\end{aligned} \tag{A.17}$$

$$\begin{aligned}
&r^2 + 2ra_2\mathrm{c}\alpha_2 + \overbrace{(\mathrm{s}^2\alpha_2+\mathrm{c}^2\alpha_2)}^{=1}a_2^2 + (r+a_2\mathrm{c}\alpha_2)(r+a_3\mathrm{c}\alpha_3) + r^2 + 2ra_3\mathrm{c}\alpha_3 \\
&+\overbrace{(\mathrm{s}^2\alpha_3+\mathrm{c}^2\alpha_3)}^{=1}a_3^2 - 2a_2\mathrm{s}\alpha_2 a_3\mathrm{s}\alpha_3 = d^2
\end{aligned} \tag{A.18}$$

$$\begin{aligned}
&r^2 + 2ra_2\mathrm{c}\alpha_2 + a_2^2 + r^2 + ra_3\mathrm{c}\alpha_3 + ra_2\mathrm{c}\alpha_2 + a_2\mathrm{c}\alpha_2 a_3\mathrm{c}\alpha_3 + r^2 \\
&+2ra_3\mathrm{c}\alpha_3 + a_3^2 - 2a_2\mathrm{s}\alpha_2 a_3\mathrm{s}\alpha_3 = d^2
\end{aligned} \tag{A.19}$$

The terms are grouped by the unknowns α_2 and α_3

$$\begin{aligned}
&\cos\alpha_2(3ra_2) + \cos\alpha_3(3ra_3) + \cos\alpha_2\cos\alpha_3(a_2a_3) + \sin\alpha_2\sin\alpha_3(-2a_2a_3) \\
&3r^2 + a_2^2 + a_3^2 - d^2 = 0
\end{aligned} \tag{A.20}$$

Not surprisingly (A.20) ends up in the same form as the two previous equations due to the symmetry of the motion platform. The three equations (A.4) (A.5) and (A.6) determines the three unknowns α_1, α_2 and α_3. Because of the trigonometric properties of the equations it is difficult or impossible to solve them analytically. It can however easily be solved numerically using the function `fsolve` in Matlab.

A.2 Platform Attitude and Position

Suppose the unknowns α_1, α_2 and α_3 are found somehow. Then the actuator end point positions in the $\{f\}$-frame $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ can easily be found using (A.1), (A.2) and (A.3). Together they define the position and orientation of the motion platform.

What is desirable to find is the rotation matrix which describes the attitude of the platform. An orthonormal basis describing the $\{p\}$-frame relative to the $\{a\}$-frame is constructed using (A.21).

$$\begin{aligned} \mathbf{y}_p &= \frac{\mathbf{p}_3 - \mathbf{p}_2}{\| \mathbf{p}_3 - \mathbf{p}_2 \|_2} \\ \mathbf{z}_p &= \frac{\mathbf{y}_p \times (\mathbf{p}_1 - \mathbf{p}_3)}{\| \mathbf{y}_p \times (\mathbf{p}_1 - \mathbf{p}_3) \|_2} \\ \mathbf{x}_p &= \mathbf{y}_p \times \mathbf{z}_p \end{aligned} \tag{A.21}$$

The rotation matrix from the $\{a\}$-frame to the $\{p\}$-frame is described by (A.22)

$$\mathbf{R}_a^p = \begin{bmatrix} \mathbf{x}_p & \mathbf{y}_p & \mathbf{z}_p \end{bmatrix} \tag{A.22}$$

The position of the $\{p\}$-frame is the same as the center of the motion platform. The motion platform are described by three actuator end points which have the same distance from the center, are evenly distributed and are (necessarily) coplanar. Then the center of the motion platform can be found by simply computing the centroid of the points.

$$\mathbf{p}_{ao}^a = \frac{1}{3}(\mathbf{p}_1 + \mathbf{p}_2 + \mathbf{p}_3) \tag{A.23}$$

Appendix B

Application Notes

B.1 Class Descriptions

`CalibrationDialog`: Dialog class for listing calibration points and computing camera matrix.

`Camera`: Camera processing thread class.

`CameraControls`: Widget class for controlling camera specific properties.

`CameraManager`: Holds the main image processing loop. Keeps everything synchronized.

`Export`: Functionality for exporting measurments to Matlab mat-files.

`FrameWriter`: Video export class. Writes every frame as an independent JPEG file.

`GlWidget`: Small class for setting up the OpenGL Wrangler Library.

`HoughSVG`: Class for exporting Hough lines in vector format.

`VideoImage`: VideoSource class for using static images as video source.

`ImageProcessing`: Main image processing algorithms.

`MainWindow`: Main window.

`MapSelector`: Widget class for selecting cargo deck square.

`Parameters`: Widget class for adjusting parameters.

`Plot`: Widget class for plotting position using QwtPlot.

`PositionEstimator`: Position and attitude computations.

`Presentation`: The OTC presentation UI.

`SourceSelector`: Dialog for user selectable video sources (OpenCV capture or video file).

`Sys1394`: One of two wrapper classes for the `libdc1394` library.

`Tracker`: Class for tracking single point.

`TrackingManager`: Over control of tracking and position estimation modules.

`Util`: Utility class for angle to rotation matrix computation, line clipping algorithms etc.

`VesselSearch`: Tracking initialization algorithm.

`Video1394`: One of two wrapper classes for the `libdc1394` library.

`VideoCV`: Wrapper class for OpenCV capture.

`VideoDisplay`: Graphics group organizing the video display for one camera.

`VideoFile`: Wrapper for FFmpeg video capture.

`VideoFrame`: Graphics group organizing the video display for one video stream.

`VideoGraphicsItem`: OpenGL video and augmented reality display class.

`VideoScene`: Class for handling user the overall video display of multiple cameras.

`VideoSource`: Abstract class representing a video source.

`WaveEstimator`: Algorithm for estimating heave amplitude for each square on the cargo deck.

`WaveMap`: Graphics class displaying heave amplitude using color coded cargo deck with gradient legend.

B.2 Used Libraries

- Qt (version 4.6.2)
- OpenCV (version 2.0.0a)
- Qwt (version 5.2.1-SVN)
- FFmpeg (version 0.5.1)
- The OpenGL Extension Wrangler Library (version 1.5.3)
- libdc1394 (version 2.1.2, Linux only)
- Matlab External C interface (version R2010a)

B.3 Known Bugs

- Removing of camera sources while system is tracking leads to crash.
- Video file looping is not reliable.
- The frame rate drops gradually after 1-2 hours of operation.
- When zooming in very much in the video display area, the OpenGL video texture is misaligned with the designated position.
- The parameter adjustment tab is not working at all.

Appendix C

Digital Attachments

Three top level directories are found.

C.1 Matlab

In this directory there is one `m`-file for generating each figure 9.1, 9.2, 9.3, 9.4 and 9.5.

The following sub directories are found:

Motion platform measurements: Code and measurement data for computing motion platform position and attitude.

Cargo deck corners comparison: Code for computing cargo deck corners using platform measurement, regular video tracking and 1/3 sub-pixel tracking.

Video measurements: The three video measurements from table 9.1

C.2 Src

The source code for the application.

C.3 References

Digital versions of some of the referenced articles.

References

ARToolKit (2010). Artoolkit, `http://www.hitl.washington.edu/artoolkit/`.

Arun, K. S., Huang, T. S. & Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets, *IEEE Trans. Pattern Anal. Mach. Intell.* **9**(5): 698–700.

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV*, O'Reilly Media Inc., Sebastopol, CA, USA.

Canny, J. (1986). A computational approach to edge detection, *IEEE Trans. Pattern Anal. Mach. Intell.* **8**(6): 679–698.

Comaniciu, D. & Meer, P. (1999). Mean shift analysis and applications, *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* **2**: 1197–1203.

Duda, R. O. & Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures, *Commun. ACM* **15**(1): 11–15.

Edwards, C. H. & Penney, D. E. (2005). *Elementary Linear Algebra*, Prentice-Hall, Inc.

Eggert, D. W., Lorusso, A. & Fisher, R. B. (1997). Estimating 3-d rigid body transformations: a comparison of four major algorithms, *Mach. Vision Appl.* **9**(5-6): 272–290.

Fischler, M. A. & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, *Communications of the ACM* **24**(6): 381–395.

Kanatani, K. (1994). Analysis of 3-d rotation fitting, *IEEE Trans. Pattern Anal. Mach. Intell.* **16**(5): 543–549.

Ma, Y., Soatto, S., Kosecka, J. & Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*, SpringerVerlag.

Marine Systems Simulator (2010). Marine systems simulator, `http://www.marinecontrol.org/download.html`.

Mathworks (2010). Calling matlab software from c and fortran programs, `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/f38569.html`.

OTC (2010). Offshore technology conference, `http://www.otcnet.org/`.

Shi, J. & Tomasi, C. (1994). Good features to track, *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pp. 593–600.

Skoglund, J. & Felsberg, M. (2006). Evaluation of subpixel tracking algorithms, *ISVC06*, pp. II: 374–382.

SNAME (1950). Nomenclature for treating the motion of a submerged body through a fluid, *Technical and Research Bulletin* .

Thelin, J. (2007). *Foundations of Qt Development*, Springer-Verlag New York, Inc.

Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations, *Computer Vision, IEEE International Conference on* **1**: 666–673 vol.1.