NTNU
Norwegian University of
Science and Technology

Object Tracking for Fine-Tuning of Robot Positions

Tore Brekke

Master of Science in Engineering Cybernetics
Submission date: June 2009
Supervisor: Alexey Pavlov, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

The use of robots for inspection and close contact operations is dependent on having an accurate model to be able to position the robot. In many complex applications, a model may not be 100% aligned with the real world since the objects that one wishes to inspect or interact with are not fixed to the same frame as the robot. There is a need for complementary methods to automatically achieve dynamical accurate positioning of the robot in relation to an object.

The goal of the project is to develop object tracking algorithms based on video capturing devices, for automatic control of a robot. The algorithm should dynamically position the robot in relation to an object. The idea is to recognize an object or equipment and position the robot with high accuracy relative to this object or equipment. The 'marker' of the object or equipment must be designed to be recognized in six degrees (x, y, z, q1, q2, q3, w). Based on available vision libraries to identify objects and position in relation to this object, it is possible to define the required move for the robot from its current position to the new position.

The task will be developed stepwise. An example of an approach is:
1. The first step should be to determine the position and the rotation of a given, simple, fixed object, in three dimensions.
2. The second step should be to determine the position and the rotation of a given, simple moving object, in three dimensions, and to predict the position and orientation of the object given a time horizon.
3. The third step is to use the predicted position and rotation of the object to track the object with a robot, given a response time for the robot.
4.If possible, extend the complexity of the object.

Thesis work outline:
Literature search/ background research

- Build up basic knowledge in tracking algorithms, robot modeling and programming, image recognition and 3D modeling. Investigate prior work in the area of the thesis. Find adequate image processing library to perform the image recognition.

Basic 3D modeling

- Build up a model off-line to be used for the image recognition.

Algorithm development

- Develop the algorithm to track the object according to the steps given above.

Test and validation

- If possible, validate the algorithm on a simple and clearly defined object in ABB's robot lab in Oslo.

Abstract

In many complex applications an accurate model of the plant is not known. Consequently, complementary methods are needed to automatically achieve accurate dynamical positioning of a robot in relation to its surroundings. This thesis describes the development of a control strategy on vision-based object tracking for a robot manipulator.

To ensure necessary robustness we assume that four distinct, sircular shapes are visible on the face of the object to inspect. Based on this information, along with knowledge of the camera parameters, the position and the orientation of the object are estimated. The developed system relies on the use of an open-source vision library, ViSP.

A Kalman filter is used to predict future states of the moving object, in order to reduce tracking errors introduced by the response time of the system.

Preface

This work is the result of a 30 credit master thesis carried out during the final semester of the Master of Engineering Cybernetics education at the Norwegian University of Science and Technology, NTNU. This thesis presents a scheme for estimating position and orientation of an object in real-time using a camera, as well as available open-source image processing libraries. In particular, the Visual Servoing Platform, ViSP, developed by the INRIA Lagadic team, has been used. The thesis also suggests a strategy for tracking a moving object with a robot manipulator.

I would like to thank my supervisor at NTNU, Alexey Pavlov, for his guidance as well as intersting conversations. I would also like to thank ABB, and especially my co-supervisor Johan Gunnar, for suggesting an intersting and challenging topic for my master thesis. Thanks to Aksel Transeth and Terje Mugaas, who works at Sintef. A big thank also goes to the fellow students at my office - Anders Garmo, Christian Sesseng, Eivind Lindeberg, Lars Andreas Wennersberg, Morten Dinhoff Pedersen, Per Aaslid and Terje Kvangardsnes. Many thanks also to my parents Sylvia and Asbjørn Brekke.

Last, but certainly not least, the biggest of thanks to the love of my life Inger Lise Skorge Aasen and to my daughter Nathalie Aasen Brekke who have been more patient, and understanding than what can be expected.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The term Robot first appeared in the play "Rossum's Universal Robots", written by the Czech playwright Karel Capek in 1920 - robota being the Czech word for work. Work, is the keyword in this context. Perhaps the greatest potential benefit to humanity that robots can provide, is to alleviate the need for humans to regularly perform tasks in dangerous environments, such as underground mines, underwater, in space, and in in hazardous industrial environments such as an offshore oil or gas rig [2].

Replacing humans with robots has an enormous potential economically as well. A robot is capable of working non-stop, and it will do exactly what it has been told to do at all times. Furthermore, if we replace humans entirely from the plant, many of the safety measures required to protect human beings from getting injured, are no longer necessary. This may help lowering costs considerably.

In the early 1980's robot manipulators were characterized as being the ultimate solution to automated manufacturing. Early predictions were that future factories would be operated by few, if any, human operators. Today, it is evident that these predictions were somewhat naive, and total human redundancy in the industrial process is still far from being a reality. One simple, but important reason is the fact that robotics is difficult or, somewhat equivalently, that humans are very good at what they do[26]. Robots still have many limitations, and successful implementations are typically highly dependent on static, predictable environments. The higher the degree of uncertainties involved in a task, the harder it will be to solve.

In order to use robots for inspection and close contact operations it is necessary to have a precise model of the workspace. However, in many complex applications a given model will only partially coincide with the real plant. For this reason it is not possible to position the robot based solely on the knowledge of the model. Complementary methods are needed to automatically achieve accurate dynamical positioning of the robot in relation to its surroundings. This thesis describes the development of a control strategy on vision-based object tracking for a robot manipulator.

Computer vision is concerned with processing images and video with the purpose of extracting useful information. An example is the task of locating an object of interest within the field of view, and estimating the object's relative position. One of the greatest challenges in this field of science, is to process the data fast enough. For this reason, historically, computer vision has been focus-

ing on off-line systems. Dealing with with real-time robot navigation, on the other hand, fast response times are required for making the system work satisfactorily. Consequently, for a given robotic task, it will be necessary to simplify and filter out as much information as possible, in order to obtain on-line performance.

The robot, with a camera attached to it, is assumed to be located in the vicinity of the object that one wishes to investigate or interact with, such that the object itself is within the cameras field of view. To ensure a necessary degree of robustness, it is assumed that four distinct, sircular shapes are visible on the face of the object. The center of these dots are extracted as image features and coupled with their corresponding points in a model of the object. The object model merely describes how these four points are related to each other in the objects own coordinate system. This information, along with parameters describing key properties of the camera used, is sufficient to estimate the position and orientation of the object relative to the robot with the camera attached to it. The developed system relies on the use of an open-source vision library, ViSP.

Assuming a non-static object, different models describing potential motion of this object are presented. The underlying assumption is that the object exhibits smooth motion that can be described as the sum of one more sinusoidals. The time needed to process an image and estimate the position and orientation of the object of interest, along with the response time of the robot suggests that a tracking scheme that is based on directing the robot relative to the currently estimated position will suffer from this lag. The error in position will be proportional to the total lag of the system and to the motion exhibited by the object of interest. Introducing a Kalman filter, we can estimate future states of the system, based on a state space model. With an adequate implementation, tracking errors can be significantly reduced by using feed-forward from the estimated future states.

1.2 Thesis Outline

In this thesis a scheme for estimating position and orientation of an object relative to a reference frame is presented. The emphasize is on making the procedure robust such that it will be applicable in industrial settings. Furthermore, a Kalman filter is designed and used in order to reduce tracking errors imposed by the control system's response time.

Vision-based control of robot manipulators requires a substantial amount of knowledge in many different fields. The author tries to explain the most important definitions and methods that are used. In section 2, basic concepts like rigid motions, homogeneuous transformations, and standard robot modeling are given. Section 3 focuses on principles from computer vision. It describes what is necessary to relate pixel coordinates to xyz world coordinates, for a point in a camera's field of view. In section 4, further theory on vision-based control is given. Section 5 describes the algorithm that is used to estimate the positon and orientation of the goal object. In section 6 we describe how to extract features from an image and how to track these features efficiently in an image sequence. Section 7 focuses mainly on state prediction using a Kalman filter. In section 8, the experimental implementations are described. In section 9, the experimental results are discussed. Section 10 presents concluding remarks, and in section 11 future work is discussed.

2 Describing Motions and Modelling Robots

2.1 Introduction

In order to plan and control an objects motion relative to its environment, it is necessary to have an effective way of describing the spatial relations between the object and its surroundings. In the first part of this chapter, fundamental principles from rigid body kinematics are presented. By assigning different coordinate frames to different parts of a system, it is possible to relate these by so-called homogeneous transformations. These transformations represent a convenient tool in describing motion.

In the second part of this chapter, basic theory from robot modeling is given.

2.1.1 Rigid Motions and Homogeneous Transformations

We need to have a way of representing position and orientation of the various parts of a system in relation to each other. The first step is often to establish a fixed coordinate system, a world or base frame, as to which any other part of the system is referenced.

The translational part of a transformation between different frames is simply described by a three dimensional vector, defining the relative translation in x, y, z coordinates. The rotational part of the transformation is in its generality described by a 3x3 rotation matrix, with a total of nine elements. The rotation matrix satisfies some convenient properties, which can be exploited when deriving kinematic equations. The rotation matrix R between two frames a and b is denoted R_b^a and is said to belong to $SO(3)$, that is, the special orthogonal group of order 3

$$SO(3) = \{R | R \in \mathbb{R}^{3x3}, R \text{ is orthogonal and det } R = 1\} \quad (1)$$

Despite the number of elements in a rotation matrix, a rigid body has at most three rotational degrees DOF, and therefore, at most three parameters are needed to specify the orientation. Consequently, there are several ways to parameterize rotation. Euler angles, roll-pitch-yaw angles[1] and axis/angle representation are examples of such parameterizations [9].

2.1.2 Euler Angles

Euler angles, as mentioned above, is a way of parameterizing a rotation matrix. We specify the complete rotation from one frame to another as the product of three successive rotations about pre-defined principle axes.

[1] It should be noted that the notion of roll, pitch and yaw in robotics terminology refers to rotations around z, y and x axes, respectively.

A common choice is to use the zyx-convention. By this convention the rotation matrix is defined as

$$R_{zyx} = R_{z,\phi}R_{y,\theta}R_{x,\psi} \tag{2}$$

$$= \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi c_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix} \tag{4}$$

By this convention, we typically assume that the rotations are made relative to the axes in a fixed frame.

Assume that we have a rotation matrix, and that we would like to extract the Euler angles that parametrize this matrix, in this case by the z-y-x convention. If we denote the elements of the rotation matrix R by r_{ij}, we can find the corresponding angles ϕ, θ, ψ as

$$\begin{aligned} \phi &= atan2(r_{32}, r_{33}) \\ \theta &= -asin(r_{31}) \\ \psi &= atan2(r_{21}, r_{11}) \end{aligned} \tag{5}$$

Rigid Motions

A rigid motion is a pure translation together with a pure rotation, defined as an ordered pair (d, R), where $d \in \mathbb{R}^3$, and $R \in SO(3)$. The group of all rigid motions is known as the special euclidean group, denoted by $SE(3)$.

Define R_1^0 as the rotation matrix that determines the orientation of a frame o_1 with axes (x_1, y_1, z_1), with respect to frame o_0 with axes (x_0, y_0, z_0). Furthermore, let d be the vector from the origin of frame o_0 to o_1. Suppose that the point p i rigidly attached to coordinate frame o_1, with local coordinates p^1. Consequently, we can express the coordinates of p with respect to frame o_0 as

$$p^0 = R_1^0 p^1 + d^0 \tag{6}$$

If we introduce a third coordinate frame o_2, and let d_2 be the vector from the origin of o_1 to the origin of o_2, and if we assume that the point p instead is attached to frame o_2, with local coordinates p^2, we can determine its coordinates relative to frame o_0 as

$$p^0 = R_1^0 p^1 + d_1^0 \tag{7}$$

in which

$$p^1 = R_2^1 p^2 + d_2^1 \tag{8}$$

By inserting equation (8) into equation (7) we obtain

$$p^0 = R_1^0 R_2^1 p^2 + R_1^0 d_2^1 + d_1^0 \tag{9}$$

or equivalently

$$p^0 = R_2^0 p^2 + d_2^0 \tag{10}$$

From equations (9) and (10) we have the relationships

$$R_2^0 = R_1^0 R_2^1 \tag{11}$$

and

$$d_2^0 = d_1^0 + R_1^0 d_2^1 \tag{12}$$

Homogeneous Transformations

If we consider a series of such rigid motions as described above, the expressions leading to equation (9), soon become quite big. Thus, a more compact way of representing the transformation, is by arranging the involved expressions in a matrix. Considering equations (11) and (12) we see that we have the following relationship

$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

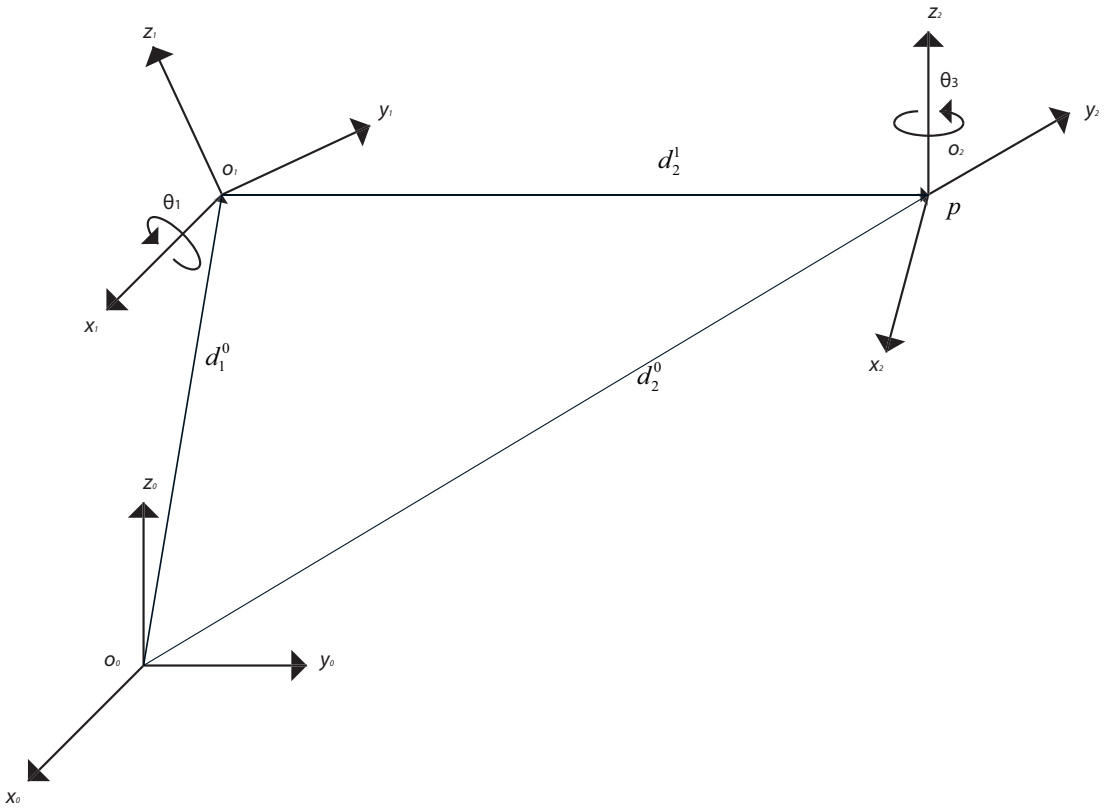

Figure 1: Rigid Motions

in which 0 denotes the vector $(0, 0, 0)$. Hence, rigid motions can be compactly described by matrices of the general form

$$H = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}, \; R \in SO(3), \; d \in \mathbb{R}^3 \tag{14}$$

A transformation matrix on this form is called a homogeneous transformation.

2.2 Robot Modelling

In robotics, an initial problem is to derive a model that relates joint values to link positions, and vice versa. A thorough introductory textbook on robotics in general, which the writer frequently refers to throughout this report, is [26].

A robot manipulator is composed of links connected by joints. Each joint represents the interconnection between two links, and is typically either of prismatic or revolute character. A prismatic joint enables relative linear motion between adjacent links, whereas a revolute joint enables relative rotation between two links. The system composed of links and joints forms a so-called kinematic chain.

The first step of robot modelling can be to establish a fixed coordinate system located at the base of the manipulator, and proceed by attaching frames to each successive link. The origins of these frames will be related through homogeneous transformations as described earlier this chapter. By multiplying successive homogeneous transformation matrices we can calculate the position and orientation of a specific frame.

The configuration of a manipulator specifies the location of every point on the manipulator. Consequently, the set of all possible configurations makes up the manipulators configuration space. Considering a manipulator composed of rigid links connected by revolute or prismatic joints, it suffices to know the joint angles for revolute joints, and joint offsets for prismatic joints in order to determine the location of any point on the manipulator. Accordingly, a convenient way of describing a manipulators configuration is by its joint values.

The number of joints determines the number of degrees of freedom (DOF). A rigid object in three dimensional space will have six DOF. Three of these correspond to its relative position, and the last three relates to the objects orientation. This means that for a manipulator to be able to obtain arbitrary position and orientation, it will need to have at least six DOF. A manipulator having more than six DOF is said to be kinematically redundant.

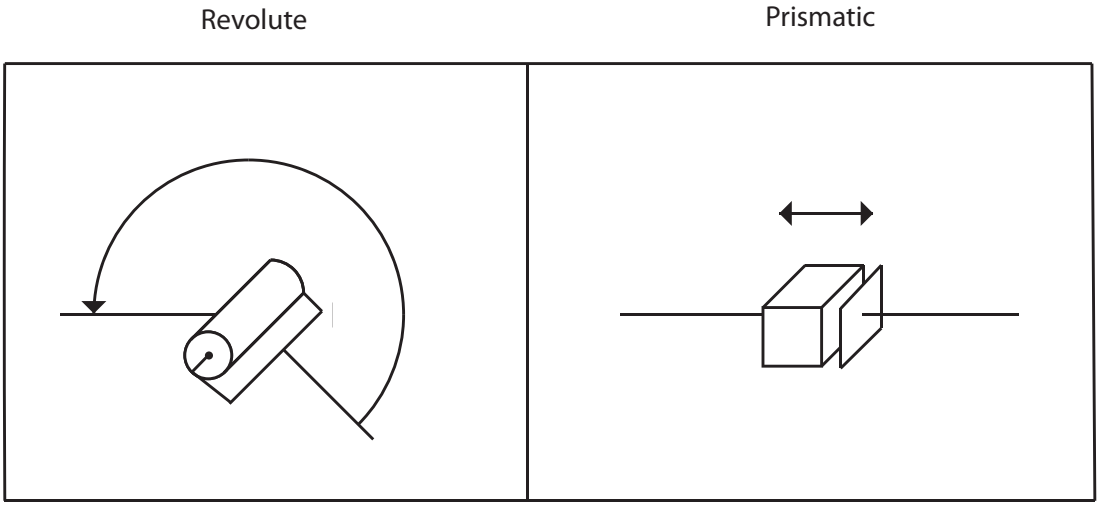

Figure 2: Symbolic representation of robot joints

2.2.1 Forward Kinematics

The forward kinematics problem can be stated as follows. Given the joint variables of the robot, find the position and orientation of the end effector. However, if the robot to be analyzed has a large number of links, finding the forward kinematics may become a complex problem. For this reason, it is very convenient to

apply a convention that simplifies matters.

An initial problem is how to select the reference frames of the robot. The Denavit-Hartenberg (DH) convention offers a simple way of achieving this. [2] In the DH convention, each homogeneous transformation A_i is represented as a product of four basic transformations

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \tag{15}$$

$$= \begin{bmatrix} c_{\theta i} & -s_{\theta i} & 0 & 0 \\ s_{\theta i} & c_{\theta i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha i} & -s_{\alpha i} & 0 \\ 0 & s_{\alpha i} & s_{\alpha i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_{\theta i} & -s_{\theta i} c_{\alpha i} & s_{\theta i} s_{\alpha i} & a_i c_{\theta i} \\ s_{\theta i} & c_{\theta i} c_{\alpha i} & -c_{\theta i} s_{\alpha i} & a_i s_{\theta i} \\ 0 & s_{\alpha i} & c_{\alpha i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The quantities θ_i, a_i, d_i and α_i are parameters associated with link i and joint i, and are commonly known as link length, link twist, link offset and joint angle, respectively. The short forms s_θ, c_θ denotes $\sin(\theta)$ and $\cos(\theta)$ respectively. When deriving the transformations between links, we assume that each joint only has one DOF. Hence, in each homogeneous transformation between adjacent links, only one variable will appear, the other parameters are constants. This may seem like a constraint as to when it is appropriate to apply the DH convention, but in fact it is not. For links with more than one DOF, we merely have to decompose it into simpler parts, where each resulting part, only possesses one DOF. It is worth mentioning that a link may very well have zero length.

For a given matrix A_i, the variable parameter will be either θ_i, or d_i, depending on whether the specific joint is revolute, or prismatic. While, in general, any homogeneous transformation may be represented using six parameters, corresponding to translation and rotation, the above transformation is described by only four parameters θ_i, a_i, d_i and α_i. The consequence, is that we can not assign the related frames as we see fit; on the contrary, we have to obey by some simple rules.

DH1: Axis x_{i+1} must be assigned such that it is perpendicular to axis x_i.

DH2: Axis x_{i+1} must be assigned such that it intersects axis z_i

This principle is illustrated in Figure 3

[2] To be precise, it should be noted that the method described, refers to the classical DH convention, as apposed to the less known, modified DH convention [7].

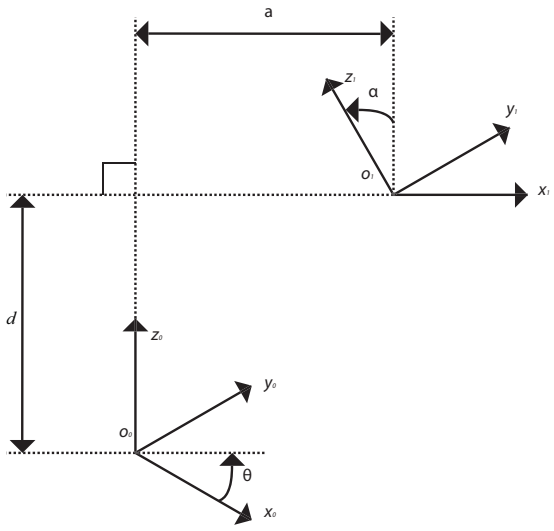

Figure 3: Coordinate frames assigned by DH convention

Consider a kinematic chain, consisting of several successive frames. The transformations between adjacent frames are denoted by A_i, such that the transformation between frame 0 and 1 becomes A_1, and the transformation between frame 1 and 2 becomes A_2. The total transformation from frame 0 to frame 2 is denoted by T_2^0.

2.2.2 Inverse Kinematics

By forward kinematics we can determine the end-effector position given the manipulator joint variables. Inverse kinematics is the problem of solving for the joint variables given the end-effectors position and orientation in space. Formally, we can express the problem as follows. Given a 4×4 homogeneous transformation

$$H = \begin{bmatrix} R & o \\ 0 & 1 \end{bmatrix} \in SE(3) \tag{16}$$

find a solution of the equation

$$T_n^0(q_1, ..., q_n) = H \tag{17}$$

where T_n^0 is the homogeneous transformation from base to end-effector frame, and H represents the desired position and orientation of the end-effector. Equation 17 results in twelve[3] nontrivial, non-linear equations in n unknown variables.

[3]Since the bottom row of T_n^0 and H are $(0, 0, 0, 1)$, 4 of the 16 entries are trivial.

The forward kinematics problem will always have a unique solution. The inverse kinematics problem, on the contrary, may or may not have a solution. The complexity of the problem increases with the number of nonzero DH parameters. Even if a solution exists, it may or may not be unique. In addition, since the forward kinematic equations in general are complicated nonlinear functions of the joint variables, solutions can be difficult to obtain even when they do exist.

Although the general problem of solving the inverse kinematics is difficult, for manipulators having six joints with the last three joint axes intersecting at a single point, it turns out that it is possible to decouple the problem into two simpler problems, namely finding the inverse position kinematics and finding the inverse orientation kinematics. As a matter of fact, many manipulators are designed such that this property is fulfilled, partly because it makes inverse kinematics easier to solve[26].

2.3 Velocity Kinematics

We will adopt the following notations for the linear and angular velocities, decomposed in a certain reference frame.

$$\begin{aligned} v_j^i &= \text{linear velocity of frame } j \text{ decomposed in frame } i \\ w_{k,j}^i &= \text{angular velocity of frame } j \text{ with respect to frame } k \\ &\quad \text{decomposed in frame i} \end{aligned}$$

The differential equation for the rotation matrix between two frames can be described as

$$\dot{R}_j^i = R_j^i S(\omega_{i,j}^i) \tag{18}$$

where $S(\omega_{i,j}^i)$ is the skew symmetric matrix formed by the the vector of angular velocities $\omega_{i,j}^i$.

Consider a point p that is rigidly attached to a moving frame j. Suppose that frame j rotates and translates relative to a frame i. The coordinates of p decomposed in frame i are given by

$$p^i = R_j^i(t) p^j \tag{19}$$

The velocity of p relative to frame i is

$$\dot{p}^i = \dot{R}_j^i(t) p^j + R_j^i(t) \dot{p}^j \tag{20}$$

$$= S(\omega_{i,j}^i) R_j^i(t) p^j + R_j^i(t) \dot{p}^j \tag{21}$$

$$= S(\omega_{i,j}^i) p^i + R_j^i(t) \dot{p}^j = \omega^i \times p^i + R_j^i(t) \dot{p}^j \tag{22}$$

where $\omega^i \times p^i$ is the tangential velocity in terms of the familiar vector cross product.

3 Computer Vision

The problem of inferring 3-D information of a scene from a set of 2-D images has a long history in computer vision. By its very nature, this problem falls into the category of so-called inverse problems, which are prone to be ill-conditioned and difficult to solve in their generality unless additional assumptions are imposed. While, in general, the task of selecting a correct mathematical model can be elusive, simple choices of representation can be made by exploiting geometric primitives such as points, lines, curves, surfaces, and volumes. [27]

3.1 The Geometry of Image Formation

Computer vision starts with the acquisition of images. A digital image is a two-dimensional brightness array in which its elements are called pixels. The term pixel is derived from the two words picture and element. In the case of a color image, its RGB (red, blue, green) values represent three such brightness arrays. In the case of a greyscale image, there is only one.

Formally, the image is a map I defined on a compact region Ω of a two-dimensional surface, taking values in the positive real numbers. For a camera, Ω is a planar, rectangular region occupied by the sensing elements. [4]

$$I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}_+; (x, y) \mapsto I(x, y) \tag{23}$$

By using a lens with focal length λ, light is focused onto a two-dimensional array of sensing elements. The value of each pixel[5] is determined by the intensity of the light that is radiated onto the corresponding element.

The image plane is defined as the plane containing the sensing array. The axes x_c and y_c form a basis for the image plane, and are taken to be parallel to the horizontal and vertical axes (respectively) of the image. The axis z_c is perpendicular to the image plane and aligned with the optical axis of the lens.

The origin of the camera frame is known as the center of projection and is located at a distance λ behind the image plane. The intersection between the image plane and the optical axis, is known as the principal point. Consequently, a point in the image plane will have coordinates (u, v, λ).

3.2 Imaging Through a Pinhole

The pinhole camera model is an idealization in which the lens is regarded to be an ideal, infinitesimally small pinhole, situated at the focal center of lens. With this assumption, the process of image formation is basically reduced to tracing

[4] Typically composed of charged couple device (CCD) sensors, although other types exist and are being used as well.

[5] or values, if we are considering a color image

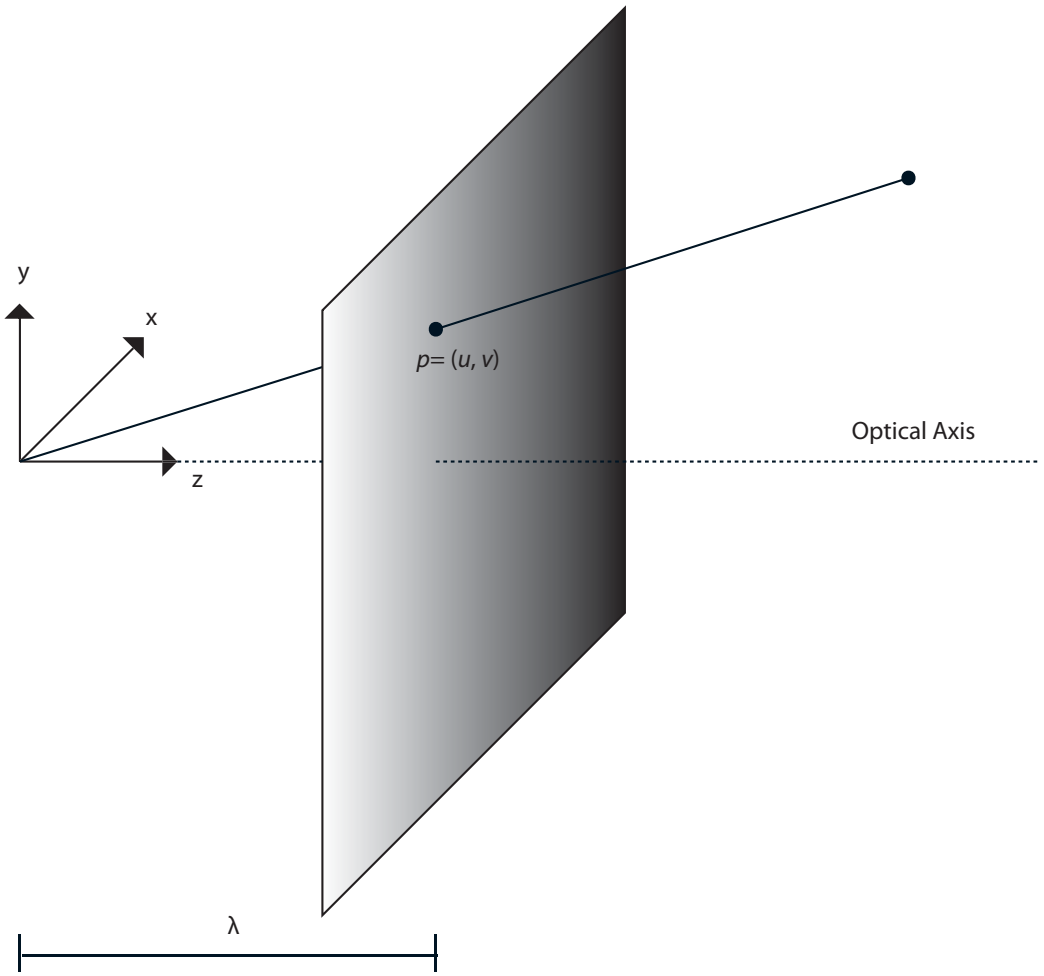

Figure 4: Camera Coordinate Frame

rays from points on objects to pixels. In reality, the pinhole will have a finite size and each point in the image plane will collect light from a cone of rays sustaining a finite, solid angle. Hence, this idealized and extremely simple model of the imaging geometry will not strictly apply. However, due to its simplicity, the pinhole model is very convenient to use, and in many cases it provides an acceptable approximation of the imaging process [12].

Suppose that P is a point in the real world with coordinates (x, y, z) relative to the camera frame. If we denote p as the projection of P onto the image plane, p will have the coordinates (u, v, λ). Assuming that we apply the pinhole model, the points P, p and the origin of the camera frame will be collinear. Consequently, the relationship between the two points can be written as

$$k \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ \lambda \end{bmatrix} \tag{24}$$

By solving for k as $k = \lambda/z$, and substituting into 24 we obtain

$$u = \lambda\frac{x}{z}, v = \lambda\frac{y}{z} \tag{25}$$

which is known as the equations for perspective projection[18].

3.3 The Image Plane and Sensor Array

Since the image is a discrete formation of pixels, it is necessary to relate pixel coordinates to image plane coordinates. We denote pixel coordinates by (r, c), and the coordinates of the principle point in the pixel array plane by (o_r, o_c). Furthermore, we denote the horizontal and vertical dimensions of a pixel by s_x and s_y respectively. The origin of the pixel array is commonly chosen to be located at a corner point, instead of at the image center. Also, equation (25) was derived with respect to the model illustrated in Figure 4. In this model the center of projection was placed behind the image plane, in order to simplify the model. As a result of this, the horizontal and vertical axes of the pixel array coordinate system usually point in opposite directions from the horizontal and vertical axes of the camera coordinate frame. Consequently, the relations between image plane coordinates and pixel array coordinates become

$$-\frac{u}{s_x} = (r - o_r), \quad -\frac{v}{s_y} = (c - o_c) \tag{26}$$

This is an approximation, as the coordinates (r, c) will be integers.

3.4 Camera Calibration

Camera calibration is concerned with determining the parameters necessary to relate pixel coordinates (r, c) to world coordinates (x, y, z) of a point in the cameras field of view. Knowing the position and orientation of the camera frame relative to the world frame we have the relation

$$x^w = R_c^w x^c + O_c^w$$

and conversely

$$x^c = R_w^c(x^w - O_w^c)$$

In order to simplify notation, we define

$$R = R_c^w x^c, \; T = -R_w^c O_c^w$$

such that

$$x^c = Rx^w + T$$

R and T are known as the extrinsic camera parameters.

From equations (26) and (25) we obtain the relations

$$r = -\frac{\lambda}{s_x}\frac{x}{z} + o_r,\ c = -\frac{\lambda}{s_y}\frac{y}{z} + o_c \tag{27}$$

which define a mapping from 3D world coordinates to pixel coordinates. It is not necessary to know both λ, s_x and s_y explicitly in order to determine (r, c). It suffices to know the ratios

$$r = \frac{\lambda}{s_x}\ c = \frac{\lambda}{s_y}$$

The parameters f_x, f_y, o_r and o_c are known as the intrinsic parameters, and they are constant for a given camera.

3.5 Determining Camera Parameters

The parameters o_r and o_r are fairly easy to determine. This can be done by using the principle of vanishing points. The vanishing points for three mutually orthogonal sets of parallel lines in the world will project onto three lines in the image, defining a triangle. The orthocenter of this triangle will be the image principal point, with coordinates (o_r, o_c). In practice, this can be achieved by placing a cube in the cameras field of view and locating its edges in the image. Then, we find the intersections of the image lines corresponding to each set of parallel lines. Finally, we compute the orthocenter for the resulting triangle.

The next step is to acquire a data set $D = \{r_i, c_i, x_i, y_i, z_i\}$, for $i = 1 \cdots N$. Here, x_i, y_i, z_i are the world coordinates of the projected point with image pixel coordinates r_i, c_i. Subsequently, we can form a set of linear equations in which the unknowns in our equations correspond to the camera parameters that we seek to determine. In general, we define the extrinsic parameters of the camera as

$$R = \begin{bmatrix} r_{11} & r_{12}r_{13} \\ r_{21} & r_{22}r_{23} \\ r_{31} & r_{32}r_{33} \end{bmatrix},\ T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

Hence, the coordinates of a point in the world with respect to the camera frame are

$$x^c = r_{11}x + r_{12}y + r_{13}z + T_x \tag{28}$$
$$y^c = r_{21}x + r_{22}y + r_{23}z + T_y \tag{29}$$
$$z^c = r_{31}x + r_{32}y + r_{33}z + T_z \tag{30}$$

Substituting these expressions into (27) we obtain

$$r - o_r = -f_x \frac{x_c}{z_c} = -f_x \frac{r_{11}x + r_{12}y + r_{13}z + T_x}{r_{31}x0r_{32}y + r_{33}z + T_z} \tag{31}$$

$$c - o_c = -f_y \frac{y^c}{z^c} = -f_y \frac{r_{21}x + r_{22}y + r_{23}z + T_y}{r_{31}x0r_{32}y + r_{33}z + T_z} \tag{32}$$

Because the principal point (o_r, o_c) is known, for simplicity, in the remaining we will use the coordinate transformation

$$r \leftarrow r - o_r, \qquad\qquad c \leftarrow c - o_c$$

By solving equations (31) and (32) for z^c and equating them, we can express the above projection equations as functions of the unknown variables r_{ij}, T_x, T_y, T_z, f_x, f_y. For data D_i we obtain

$$r_i f_y(r_{21}x_i + r_{22}y_i + r_{23}z_i + T_y) = c_i f_x(r_{11}x_i + r_{12}y_i + r_{13}z_i + T_x)$$

which, by defining $\alpha = f_x/f_y$, can be rewritten as

$$r_i r_{21}x_i + r_i r_{22}y_i + r_i r_{23}z_i + r_i T_y - \alpha c_i r_{11}x_i - \alpha r_{12}y_i - \alpha r_{13}z_i - \alpha T_x = 0 \tag{33}$$

Now, let N be the number of samples in D. We can arrange N equations as in (33) in a matrix

$$Ax = 0 \tag{34}$$

in which x denotes the vector

$$x = \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \\ T_y \\ \alpha r_{11} \\ \alpha r_{12} \\ \alpha r_{13} \\ \alpha T_x \end{bmatrix} \tag{35}$$

For any solution $\bar{x} = [\bar{x}_1, ..., \bar{x}_8]^T$ of the equation(34), we can only conclude that this solution is a scalar multiple kx of the desired solution x. However, at this stage it is possible to exploit properties of R being a rotation matrix[6]. Specifically, we know that

$$(\bar{x}_1^2 + \bar{x}_2^2 + \bar{x}_3^2)^{\frac{1}{2}} = (k^2(r_{21}^2 + r_{22}^2 + r_{23}^2))^{\frac{1}{2}} = |k| \tag{36}$$

[6]Remember the definition of special orthogonal groups 1

and

$$(\bar{x}_5^2 + \bar{x}_6^2 + \bar{x}_7^2)^{\frac{1}{2}} = (\alpha^2 k^2 (r_{21}^2 + r_{22}^2 + r_{23}^2))^{\frac{1}{2}} = \alpha |k| \tag{37}$$

The scalar value, $\alpha > 0$, and it remains to decide the sign of k. By equation (27)[7] we conclude that $rx^c < 0$, and by this reason k is chosen such that $r(r_{11}x + r_{12}yr_{13}z + T_x) < 0$. Proceeding, we then need to determine T_x, f_x, f_y. We know that the third column of R can simply be described as the vector cross product of columns one and two. Furthermore, we know that $\alpha = f_x/f_y$. Hence, it suffices to determine T_z and f_x. Recall that

$$r = f_x \frac{x^c}{z^c} = -f_x \frac{r_{11}x + r_{12}y + r_{13}z + T_x}{r_{31}x0r_{32}y + r_{33}z + T_z} \tag{38}$$

By proceeding in a manner analogous to that of finding the first eight parameters, (38) can be written as

$$r(r_{31}x + r_{32}yr_{33}z + T_z) = -f_x(r_{11}x + r_{21}y + r_{13}z + T_x) \tag{39}$$

This resulting equation is readily solved for T_z and f_x.

3.6 Camera Parameters from Experimental Data

Several different cameras have been used throughout this work. First, a QuickCam web camera from Logitech was used. This is a simple camera which transmits its data to a computer via an USB port. Later on, this camera was replaced by a FireWire camera from CAMTEK. Choosing FireWire instead of USB should enhance the data transmission rate. In addition, it generates less noise in the image than USB camera does. In connection with the implementations at the robotics labs at ABB and Sintef, camera calibration was used to identify the camera parameters for the cameras that were used there.

To determine the camera parameters a camera calibration toolbox[1] for Matlab was used. This toolbox is really a program operated by means of a graphical user interface. Below is a short summary of the procedure.

The first thing that needed to be done was to print out a checkerboard pattern consisting of black and white squares of equal, known sizes. Then, a series of pictures were taken of this checkerboard pattern from different positions and orientations. Once this was done, the pictures needed to be loaded into memory by the Matlab program. In the following the pictures were treated one by one, specifying the size of the squares, and clicking on the extreme corners of the patterns by means of the computer mouse. When all the pictures had been processed this way, the program performed several optimization algorithms. The camera parameters, with specified uncertainties, were then determined.

[7] Recall the transformation $r \leftarrow r - o_r$.

In addition, the program enabled the user to take new pictures of the checkerboard pattern. After clicking on the extreme corners, the system would then output information regarding the position and orientation of the camera relative to the checkerboard pattern, corresponding to the moment when the picture was taken.

The identified camera parameters for the cameras from Logitech and CAMTEK are summarized in Table 1 and Table 2, respectively[8]. The camera parameters for the robot cameras used at ABB and at Sintef can be found in Table 3 and Table 4. In addition to the numbers listed here, the program also determined the uncertainties associated with the parameters, along with several other parameters not mentioned here.

Focal length λ	833.28	823.31
Principal point (o_r, o_c)	467.75	346.41

Table 1: Camera Parameters for Quickcam

Focal length λ	381.02	382.46
Principal point (o_r, o_c)	159.50	119.50

Table 2: Camera Parameters for CAMTEK

Focal length λ	774.01	777.90
Principal point (o_r, o_c)	308.75	211.93

Table 3: Camera Parameters for ABBs robot camera

Focal length λ	759.19	770.08
Principal point (o_r, o_c)	338.66	242.15

Table 4: Camera Parameters for Sintefs robot camera

3.7

In the preceding sections, some aspects of computer vision have been addressed. Certainly, computer vision in general covers a huge amount of different theoretical and practical fields and the presented material cover only a small fraction of it. A textbook that offers an in-depth look into computer vision is [19].

[8]The reason that there are two numbers for λ is that the focal length here is actually given in pixels

4 Vision-Based Control

The problem faced in vision-based control is that of extracting a relevant and robust set of parameters from an image and use these parameters to control the motion of the manipulator in real time[26].

Over the last decades, a number of different approaches have been suggested for the task of vision-based control. Their differences arise from aspects such as how camera(s) and manipulator are configured, in what way image data are being used, how the coordinate systems are chosen and so on.

Visual sensing and manipulation can be combined in an open-loop kind of way, "looking" then moving. Consequently, the results depend on the accuracy of the visual sensor and the robot end-effector. By using a visual-feedback control loop, the performance of the system can be greatly improved. Closed-loop position control for a robot end-effector is commonly referred to as visual servoing[15]. In general, visual servoing comprises principles from a vast number of areas including high-speed image processing, kinematics, dynamics, control theory and real-time computing. The overall goal is to control a robot to manipulate its environment using vision.

4.1 Configuration Issues

When designing a vision-based control system, a number of different questions must be addressed before the actual system can be constructed. A fundamental point to consider is the camera configuration. Amongst other things, this involves deciding on how many cameras to incorporate, where to locate it (or them), and the type of camera(s) to use[9].

Location of the camera(s) is basically a question of whether to attach the camera(s) to the robot, or to place it (or them) somewhere around the workspace. These two possibilities are referred to as fixed camera and eye-in-hand configurations, respectively[18]. In the latter, there exists a known, often constant relationship between the pose of the camera and the pose of the end-effector. The two approaches are shown in Figure 5.

An advantage with fixed camera, is that the view is kept constant when robot moves. This means that there exists a fixed relation between the workspace and the camera. However, a disadvantage with this approach is that the cameras field of view may be blocked as the robot manoeuvers, which would be unacceptable. Furthermore, if the workspace of the manipulator covers a large area, many cameras may be needed in order to cover it all.

In an eye-in-hand configuration, the camera is typically mounted above the manipulators wrist, such that possible wrist movements will not propagate onto

[9] An important difference being whether a camera with a zoom lens, or a lens with fixed focal length should be used.

the camera. A downside to with this system is that the geometry between camera and workspace is in constant change as the manipulator moves.

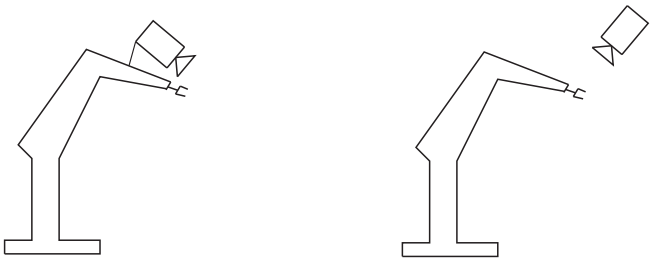

Figure 5: Eye-in-hand configuration vs fixed camera

Single camera vision systems are most commonly used, because they typically will be less expensive and simpler to build than multi-camera systems[21]. However, using two cameras in a stereo configuration [14][16][17] makes certain computer vision problems easier to handle.

4.2 Vision-Based Control Configurations

Vision-based control methods differ in the error used to compute the control law. In position-based visual servo control[10], the vision data are used to build a three dimensional representation of the surroundings. The main problem with this approach is to build the 3D model in real time. It is also highly dependent on the camera being accurately calibrated, as it tends to be sensitive with respect to calibration errors[26].

In image-based visual servo control the image data are being used to directly control the robot. An error is computed in the two dimensional image space, based on visual features such as coordinates of points or the orientation of lines. Image-based visual servoing is generally considered to be quite robust to both camera calibration errors and robot calibration errors [10]. On the other hand,

[10]Also known as 3D visual servoing

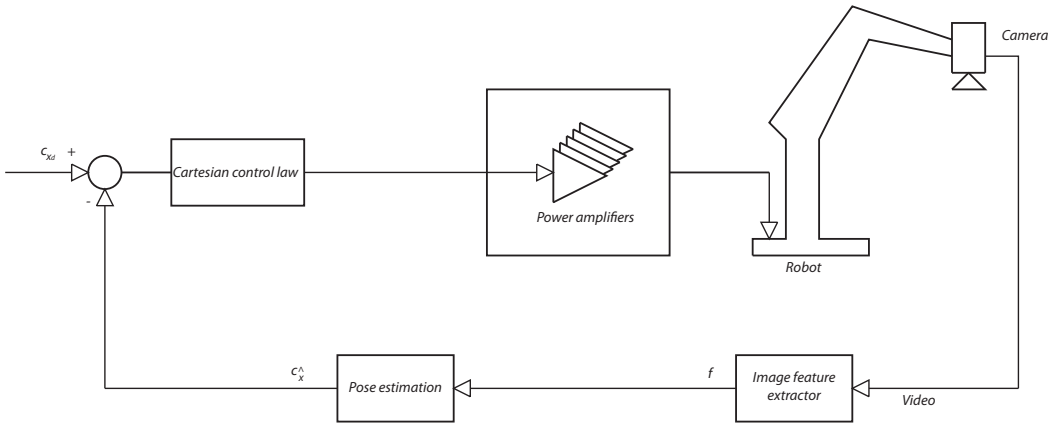

Figure 6: Position-based visual servo (PBVS) structure

convergence can be theoretically ensured only in a region around the desired position[21].

4.3 The Manipulator Jacobian

For a general n-link manipulator the Jacobian represents the instantaneous transformation between the n-dimensional vector of joint velocities $\dot{q}(t)$ and the 6-dimensional vector of linear and angular velocities of the end-effector. This transformation is then represented by a $6\text{x}n$ matrix. It can be shown[26] that

$$S(\omega_n^0) = \dot{R}_n^0 (R_n^0)^T \tag{40}$$

defines the angular velocity vector ω_n^0 of the end-effector, in which S denotes the skew-symmetrical matrix as defined in for instance [9]. Furthermore, let

$$v_n^0 = \dot{o}_n^0 \tag{41}$$

denote the linear velocity of the end-effector. We then seek expressions such that

$$v_n^0 = J_v \dot{q} \tag{42}$$
$$\omega_n^0 = J_\omega \dot{q} \tag{43}$$

in which J_v and J_ω are $3\text{x}n$ matrices. Equations (42) and (43) can be written together as

$$\xi = J\dot{q} \tag{44}$$

The $6\text{x}n$ matrix J is called the manipulator Jacobian.

4.4 The Interaction Matrix

As commented in 2.2.2 the inverse kinematics problem is hard to solve. The inverse velocity problem on the other hand, is generally much easier to solve. This can be done by simply inverting the manipulator Jacobian, assuming it is non-singular. Whereas inverse kinematics equations represent a non-linear mapping between complicated geometrical spaces, the mapping of velocities is a linear map between linear subspaces. In the same way, the relation between vectors defined by means of image features and camera velocities will be a linear mapping between linear subspaces.

Now, let $s(t)$ denote a vector of image feature values, and $\dot{s(t)}$ its rates of change. The latter is referred to as an image feature velocity. If we consider a single point, the corresponding feature vector is

$$s(t) = \begin{bmatrix} u(t) \\ v(t) \end{bmatrix}$$

If we denote the camera velocity by ξ we have

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{45}$$

where v and ω are the linear and angular velocities of the camera, respectively.

The relation between $\dot{s}$ and ξ is given by

$$\dot{s} = L(s, q)\xi \tag{46}$$

in which L is called the interaction matrix[18]. This matrix describes how image feature parameters change with respect to changing camera pose. Its explicit form relies on the features chosen. In its simplest form, the features are merely coordinates of points in an image.

4.5 The Interaction Matrix for Point Features

Consider the scenario with a moving camera observing a point with fixed position. For instance, this could involve a camera positioning itself relative to an object that is to be manipulated in some sense. At time t the position of the origin of the camera frame, relative to the fixed frame, is denoted by $o(t)$. Likewise, the orientation is given a rotation matrix $R_c^0 = R(t)$. Let P be the fixed point of interest, with $s = [u, v]^T$ being the corresponding feature vector. This is shown in Figure 7. Now, the aim is to derive the interaction matrix L, relating

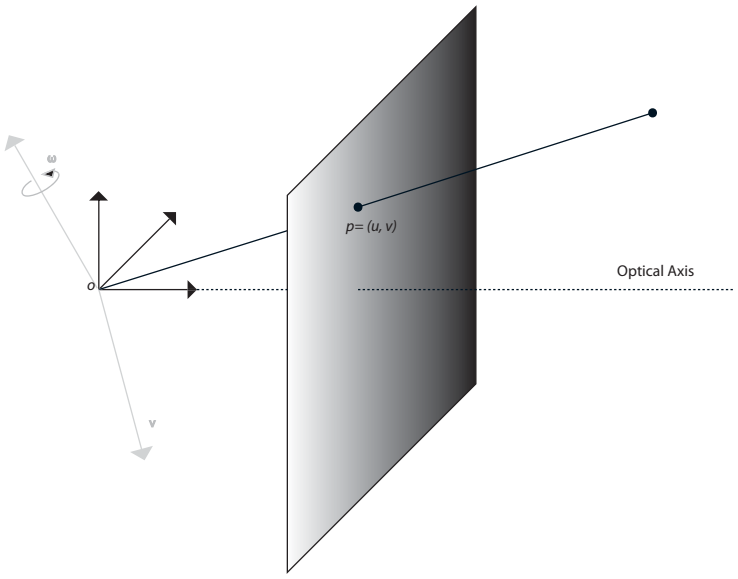

Figure 7: The camera frame moves with linear velocity v and angular velocity ω relative to a fixed point in space.

velocity of the camera ξ to the image feature velocity, $\dot{s}$

Let p^0 be the time-invariant coordinates of P relative to the world frame, and $p^c(t)$ its time-varying coordinates relative to the moving camera frame. Then by 6 we can write

$$p^0 = R(t)p^c(t) + o(t) \tag{47}$$

It follows that we can solve for the coordinates of P relative to the camera frame as

$$p^c(t) = R^T(t)[p^0 - o(t)] \tag{48}$$

Dropping the explicit reference to time, by notational convenience, the velocity of the point P relative to the moving camera frame becomes

$$\frac{d}{dt}p^c(t) = \dot{R}^T(p^0 - o) - R^T\dot{o} \tag{49}$$

From equation (40) we know that $\dot{R} = S(\omega)R$, and it follows that $\dot{R}^T = R^T S(\omega)^T = R^T S(-\omega)$, which enables us to write equation (49) as

$$\begin{aligned}\dot{R}^T(p^0 - o) - R^T\dot{o} &= R^T S(-\omega)(p^0 - o) - R^T\dot{o} \\ &= R^T S(-\omega)RR^T(p^0 - o) - R^T\dot{o} \\ &= -R^T\omega \times R^T(p^0 - o) - R^T\dot{o}\end{aligned} \tag{50}$$

From equation (48) we know that $R^T(p^0 - o(t)) = p^c$. Furthermore, since ω expresses angular velocity for the moving frame in coordinates expressed with respect to the fixed frame, we see that $R^T\omega = R_0^c\omega^0 = \omega_c$ gives the moving frames angular velocity expressed with respect to the moving frame. Also, it is clear that $R^T\dot{o} = \dot{o}^c$. By these expressions, the velocity of P relative to the moving frame is

$$\dot{p}^c = -\omega^c \times p^c - \dot{o}^c \tag{51}$$

4.6 Constructing the Interaction Matrix

To derive the interaction matrix for point features, we simply apply equation (51) and the expression for perspective projection derived in 3.2. For notational convenience the coordinates for P relative to the camera frame are defined as $p^c = [x, y, z]^T$. The coordinates for the angular velocity vector is denoted by $\omega^c = [\omega_x, \omega_y, \omega_z]^T = R^T\omega$. Furthermore, we assign $R^T\dot{o} = \dot{o}^c$. By the above conventions, equation (51) can be rewritten as

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = -\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} \times \begin{bmatrix}x\\ y\\ z\end{bmatrix} - \begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix}$$

and if we write it out

$$\dot{x} = y\omega_z - z\omega_y - v_x \tag{52}$$

$$\dot{x} = z\omega_x - x\omega_z - v_y \tag{53}$$

$$\dot{x} = x\omega_y - y\omega_x - v_z \tag{54}$$

We know that u and v represents the image coordinates of Ps projection in the image. By 25, x and y can be expressed as

$$x = \frac{uz}{\lambda}, y = \frac{vz}{\lambda} \tag{55}$$

which substituted into equations (52), (53) and (54) yields

$$\dot{x} = \frac{vz}{\lambda}\omega_z - z\omega_y - v_x \tag{56}$$
$$\dot{x} = z\omega_x - \frac{uz}{\lambda}\omega_z - v_y \tag{57}$$
$$\dot{x} = \frac{uz}{\lambda}\omega_y - \frac{vz}{\lambda}\omega_x - v_z \tag{58}$$

The equations above, describe velocity $\dot{p}^c$ by means of image coordinates u and v, the depth of the point z, and the angular and linear velocity of the camera. The next step is to derive expressions for $\dot{u}$ and $\dot{v}$ and combine these with equations (56), (57) and (58).

Clearly, these are given by

$$\dot{u} = \frac{d}{dt}\frac{\lambda x}{z} = \lambda\frac{z\dot{x} - x\dot{z}}{z^2} \tag{59}$$
$$\dot{v} = \frac{d}{dt}\frac{\lambda y}{z} = \lambda\frac{z\dot{y} - y\dot{z}}{z^2} \tag{60}$$
$$\tag{61}$$

Substituting 56 and 58 into 59 yields

$$\begin{aligned}\dot{u} &= \frac{\lambda}{z^2}\big(z[\frac{vz}{\lambda}\omega_z - z\omega_y - v_x] - \frac{uz}{\lambda}[\frac{uz}{\lambda}\omega_y - \frac{vz}{\lambda}\omega_x - v_z]\big) \\ &= -\frac{\lambda}{z}v_x + \frac{u}{z}v_z + \frac{uv}{\lambda}\omega_x - \frac{\lambda^2 + u^2}{\lambda}\omega_y + v\omega_z\end{aligned} \tag{62}$$

$$\begin{aligned}\dot{v} &= \frac{\lambda}{z^2}\big(z[-\frac{uz}{\lambda}\omega_z + z\omega_x - v_y] - \frac{vz}{\lambda}[\frac{uz}{\lambda}\omega_y - \frac{vz}{\lambda}\omega_x - v_z]\big) \\ &= -\frac{\lambda}{z}v_y + \frac{v}{z}v_z + \frac{\lambda^2 + v^2}{\lambda}\omega_x - \frac{uv}{\lambda}\omega_y - u\omega_z\end{aligned} \tag{63}$$

If we combine equations (62) and (63) we can stack them in a matrix such that

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} -\frac{\lambda}{z} & 0 & \frac{u}{z} & \frac{uv}{\lambda} & -\frac{\lambda^2+u^2}{\lambda} & v \\ 0 & -\frac{\lambda}{z} & \frac{v}{z} & \frac{\lambda^2+v^2}{\lambda} & -\frac{uv}{\lambda} & -u \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{64}$$

This matrix is the interaction matrix for a point. Moreover, the interaction matrix may be split into two parts such that

$$\dot{s} = L_v(u, v, z)v + L_\omega(u, v)\omega \tag{65}$$

where $L_v(u, v, z)$, consisting of the three first columns of L, is a function of both image coordinates and depth. $L_\omega(u, v)$ on the other hand, contains the last three rows of L and is only a function of the image coordinates, not of the depth. In real world implementations this can prove useful, when the true depth may be unknown.

Due to the fact that the camera velocity has six degrees of freedom, while at same time, only two of these, u and v, are being observed it follows that not all motions will introduce observable changes in the image. To elaborate, since $L \in \mathbb{R}^{2x6}$ it has a null space of dimension four. For the interaction matrix derived in equation 64, the null space can be shown to be spanned by the four vectors[18]

$$\begin{bmatrix} u \\ v \\ \lambda \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ u \\ v \\ \lambda \end{bmatrix}, \begin{bmatrix} uvz \\ -(u^2+\lambda^2)z \\ \lambda vz \\ -\lambda^2 \\ 0 \\ u\lambda \end{bmatrix}, \begin{bmatrix} \lambda(u^2+v^2+\lambda^2)z \\ 0 \\ -u(u^2+v^2+\lambda^2)z \\ uv\lambda \\ -(u^2+\lambda^2)z \\ u\lambda^2 \end{bmatrix} \tag{66}$$

It is trivial to augment the derived interaction matrix to incorporate several feature points. Simply define a vector of features, $s = [u_1 v_1 ... u_n v_n]^T$ and a vector of depths $z = [z_1 ... z_n]^T$. The composite interaction matrix L_c will be a function of the image coordinates of n points and n depth values,

$$\dot{s} = L_c(s, z)\xi \tag{67}$$

in which the individual matrices, corresponding to each feature point, are simply stacked on top of each other.

$$L_c(s,z) = \begin{bmatrix} L_1(u_1,v_1,z_1) \\ \vdots \\ L_n(u_n,v_n,z_n) \end{bmatrix} = \begin{bmatrix} -\frac{\lambda}{z_1} & 0 & \frac{u_1}{z_1} & \frac{u_1 v_1}{\lambda} & -\frac{\lambda^2+u_1^2}{\lambda} & v_1 \\ 0 & -\frac{\lambda}{z_1} & \frac{v_1}{z_1} & \frac{\lambda^2+v^2}{\lambda} & -\frac{u_1 v_1}{\lambda} & -u_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\frac{\lambda}{z_n} & 0 & \frac{u_n}{z_n} & \frac{u_n v_n}{\lambda} & -\frac{\lambda^2+u_n^2}{\lambda} & v_n \\ 0 & -\frac{\lambda}{z_n} & \frac{v_n}{z_n} & \frac{\lambda^2+v_n^2}{\lambda} & -\frac{u_n v_n}{\lambda} & -u_n \end{bmatrix} \tag{68}$$

4.6.1 The Interaction Matrix for Other Features

In the previous subsection the interaction matrix for a point was derived. It is also possible to consider more complex features such as lines, circles and cylinders. A general framework for computing **L** is given in [11]. Below, the interaction matrix for a straight line is given, but its derivation is omitted.

The Interaction Matrix for a Line

A straight line can be considered as the intersection of two planes

$$\begin{aligned} A_1 X + B_1 Y + C_1 Z &= 0 \\ A_2 X + B_2 Y + C_2 Z + D_2 &= 0 \end{aligned} \tag{69}$$

The corresponding equation for the projected line in the image plane is given by

$$x\cos\theta + y\sin\theta - \rho = 0 \tag{70}$$

The interaction matrix associated to $\mathbf{p}_m = (\theta, \rho)$ is given by

$$\begin{aligned} \mathbf{L}_\theta &= \begin{bmatrix} \lambda_\theta \cos\theta & \lambda_\theta \sin\theta & -\lambda_\theta \rho & \rho\cos\theta - \rho\sin\theta & -1 \end{bmatrix} \\ \mathbf{L}_\rho &= \begin{bmatrix} \lambda_\rho \cos\theta & \lambda_\rho \sin\theta & -\lambda_\rho \rho & (1+\rho^2)\sin\theta - (1+\rho^2)\cos\theta & 0 \end{bmatrix} \end{aligned} \tag{71}$$

4.7 Image-Based Control Laws

As mentioned in section 4.2, with image-based control methods the image error function is given by

$$e(t) = s(t) - s^d \tag{72}$$

in which s^d denotes the desired configuration of image features. The problem is then to find a mapping from the error function to a commanded camera motion. Henceforth, it is assumed the robot is controlled on a lower level, such that feasible, desired trajectories can be followed. A standard approach to image-based control is to compute a desired camera velocity, xi, and use this as an

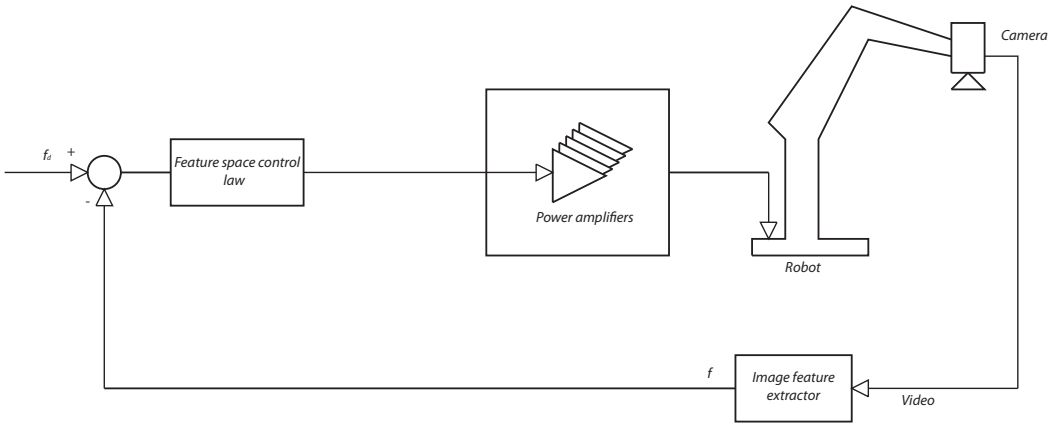

Figure 8: Image-based visual servo (IBVS) structure

input to the system. Usually, relating image feature velocities to camera velocity, is done solving equation 46 [26]. This will then give the camera velocity that produces the desired feature velocity, $\dot{s}$. If the interaction matrix is square and non-singular, the solution can be obtained simply by inverting this matrix.

Let k be the number of feature vectors and m the number of components in the camera body velocity, ξ, it follows that $L \in \mathbb{R}^{k \times m}$. If $k = m$ and L is full rank, then the solution is given by

$$\xi = L^{-1}\dot{s} \tag{73}$$

If $k < m$, the system is under constrained, and L^{-1} does not exist. This basically means that too few feature velocities are being observed to unambiguously determine the camera motion ξ. Nevertheless, a solution can be computed as

$$\xi = L^{\dagger}\dot{s} + (I_m - L^{\dagger}L)b \tag{74}$$

in which $L^{\dagger}$ is the pseudo inverse for L given by

$$L^{\dagger} = L^T(LL^T)^{-1} \tag{75}$$

,I_m denotes the mxm identity matrix, and $b \in \mathbb{R}^m$ is an arbitrary vector.

In general, when $k < m$, $(I - LL^{\dagger}) \neq 0$ and all vectors of the form $(I - LL^{\dagger})b$ lie in the null space of L, implying that those components of the camera that are unobservable lie in the null space of L. Letting $b = 0$, the value for ξ that minimizes the norm $||\dot{s} - L\xi||$ is obtained.

In the final possibility, when $k > m$ and L is full rank, the system will typically behave in an inconsistent matter, implying that more feature velocities are observed than what is required to uniquely determine the camera motion. The rank of the null space of L will be zero, and a least squares solution can be used

$$\xi = L^{\dagger}\dot{s} \tag{76}$$

where the pseudo inverse is given by

$$L^{\dagger} = (L^T L)^{-1} L^T \tag{77}$$

4.8 Proportional Control

By using Lyapunov theory a stable control system can be designed[26]. Considering the system given by 46 and the Lyapunov candidate function

$$V(t) = \frac{1}{2}||e(t)||^2 = \frac{1}{2}e^T e \tag{78}$$

$$\dot{V} = \frac{d}{dt}\frac{1}{2}e^T e = e^T \dot{e} \tag{79}$$

Hence, if a controller could be designed such that

$$\dot{e} = -\lambda e \tag{80}$$

with, $\lambda > 0$, we would achieve

$$\dot{V} = -\lambda e^T e < 0 \tag{81}$$

ensuring an exponentially stable closed-loop system, such that asymptotic stability is ensured even under small perturbations, such as small camera calibration errors.

It turns out designing such a controller in a visual servo control, often can be achieved. The derivative of the error function is

$$\dot{e} = \frac{d}{dt}(s(t) - s^d) = \dot{s(t)} = L\xi \tag{82}$$

By substituting this result into equation (80) we get

$$-\lambda e(t) = L\xi \tag{83}$$

Now, if $k = m$ and L has full rank, consequently, L^{-1} exists, leading to an exponentially stable system

$$\xi = -\lambda L^{-1} e(t) \tag{84}$$

However, when $k > m$, exponential stability can not be achieved, as the derivative $\dot{V}$ is only negative semi-definite. In reality, the true values of L or $L^{\dagger}$ will not be perfectly known, because these matrices rely on knowledge of z, that is, the depth. These will have to be estimated by the system, and consequently we will have an estimate for L, $\hat{L}$. Nevertheless, the same reasoning applies. Substituting for the estimated value, it follows that the resulting system will be stable whenever $L\hat{L}$ is positive definite. This fact helps explain why visual servoing is a robust scheme with respect to small calibration errors.

4.9 Performance of IBVS systems

The image-based control law described above offers good performance with respect to image error. However, a downside with the approach is that it can sometimes induce large camera motions causing a task to fail. The source of this problem is that the scheme only considers the image error, and in which direction it should be moving in order to cancel out this error. Consequently, the manipulator may have to perform impossible motions in order to obey the systems commands. However, so-called partitioned methods provide one way of dealing with this problem.

4.10 Partitioned Methods

As have been mentioned, image-based methods are by their nature robust to errors in calibration and sensing. Their negative sides arise from the fact that they may cause system failure when a required camera motion exceeds the systems capabilities. For instance, if the required camera motion is a large rotation about the optical axis, each feature point is caused to move in a straight line from its currents image position to its desired position. Effectively, the induced camera motion would have been a retreat along the optical axis, and for a required rotation of π the camera would retreat to $z = -\infty$ at which point det(L)=0 and the controller fails[26].

The idea of partitioned methods is to use the interaction matrix L to control only a subset of the camera degrees of freedom, and consequently employ other methods to control the remaining degrees of freedom. Equation (64) can be written as

$$\dot{s} = \begin{bmatrix} L_{v_x} & L_{v_y} & L_{v_z} & L_{\omega_x} & L_{\omega_y} & L_{\omega_z} \end{bmatrix} \xi \tag{85}$$

$$= \begin{bmatrix} L_{v_x} & L_{v_y} & L_{v_z} & L_{\omega_x} & L_{\omega_y} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_x \\ \omega_y \end{bmatrix} + \begin{bmatrix} L_{v_z} & L_{\omega_z} \end{bmatrix} \begin{bmatrix} v_z \\ \omega_z \end{bmatrix} \tag{86}$$

$$= L_{xy}\xi_{xy} + L_z\xi_z \tag{87}$$

In this equation $\dot{s}_z = L_z\xi_z$ gives the component of $\dot{s}$ causing the camera to move along and rotate about the optical axis. In this equation control is allowed to be partitioned into two separated components. If we suppose now that a control scheme has been implemented to determine the value of $\xi_{xy} = u_z$. By employing an image-based method to find ξ_{xy}, equation(87) could be solved as

$$\xi_{xy} = L_{xy}^{\dagger}(\dot{s} - L_z\xi_z) \tag{88}$$

The term $(-L_{xy}^{\dagger}L_z\xi_z)$ is the required value of ξ_{xy} to cancel the feature motion $\dot{s}_z$. Furthermore, the control $u_{xy} = \xi_{xy} = L_{xy}^{\dagger}\dot{s}$ gives the velocity along and rotation about the camera x and y axes, producing the desired $\dot{s}$ when the image feature motion resulting from ξ_z has been accounted for.

Using the same Lypunov design method as earlier, setting $\dot{e} = -\lambda e$, we obtain

$$-\lambda e = \dot{e} = \dot{s} = L_{xy}\xi_{xy} + L_z\xi_z \tag{89}$$

leading to

$$\xi_{xy} = -L_{xy}^{\dagger}(\lambda e(t) + L_z\xi_z) \tag{90}$$

The term $(\lambda e(t) + L_z\xi_z)$ can be thought of as a modified error incorporating the initial image feature error while also taking into account the feature error that will be induced by movement along and about the optical axis because of ξ_z. Now, what remains is to develop a control law, allowing the value of ξ_z to be determined. In order to find ω_z, the angle θ_{ij} from the horizontal axis of the image plane, to the directed line segment joining two feature points can be used. Specifically, the longest line segment that can be constructed from the feature points, should be used, for numerical conditioning reasons. The value $\omega_z = \gamma_{w_z}(\theta_{ij}^d - \theta_{ij})$ where θ_{ij}^d denotes the desired value, and γ_{ω_z} is some scalar gain. The apparent size of an object in the image can be used to determine v_z. If σ^2 denotes the area of some polygon, then we define v_z as

$$v_z = \gamma_{v_z} ln(\frac{\sigma^d}{\sigma}) \tag{91}$$

The apparent size is a well-suited feature as it is a scalar, rotation invariant and, it can be easily computed.

5 Pose Estimation

5.1 Introduction

Estimating the pose of an object relative to some base frame is a problem that lies at the very heart of computer vision. In many applications it is fundamental that pose computation is performed with high accuracy and, at the same time, as fast as possible.

Different types of sensors providing measurements to the system may be considered. These include, but are not limited to cameras, GPS, gyroscopes, accelerometers, lasers and ultrasonic emitter/detector combinations. Most of these do not provide the necessary information regarding the object pose directly, but rather it must be estimated.

Cameras provide large amounts of information that can be exploited in various ways. However, the fact that cameras supply only degenerate 2D information about the witnessed 3D world presents a great challenge. For this reason it is necessary to make certain assumptions about the scene. One possibility is to assume that a sufficiently accurate 3D model of the goal object is known in advance. This is the approach taken in this work.

5.2 Model-Based Object Pose

In this thesis the measurement sensor considered is a single camera and the pose is estimated from a single image. However, the pose is re-estimated as frequently as possible, to ensure the best estimate at all times.

The intrinsic camera parameters have been estimated beforehand by means of camera calibration as described in section 3. Also, a 3D model of the goal object is assumed to be known with sufficient accuracy. With this assumption we know how different parts of the target are related physically, which poses strong constraints that can be exploited in the pose computation.

5.3 Virtual Visual Servoing

The method used for estimating the object pose in this thesis is based on direct analogy between pose estimation and classical image-based visual servoing. It goes by the term Virtual Visual Servoing (VVS).

In most image-based control methods, the main computation problem is that of determining the relation between 3D coordinates of features (points, lines, ellipses, etc.) and their projections onto the image plane.

In order for these methods to succeed at their task, it is essential that the location of the considered features in the object frame are accurately known. It is also very important to be able to track the image features (in the image) from one image to the next. If the goal object is assumed to be at rest, and

if the initial computed pose is known to be satisfactorily accurate, it might in some cases suffice to compute the pose only once and perform the positioning accordingly. However, chances are that recomputing the pose, as closing in on the goal object, will improve the quality of the pose estimate, and increase the likelihood for succeeding at the positioning task.

If the goal object is non-static, that is, if the goal object is moving relative to the robot and the task to solve is a matter of tracking, then obviously, it is absolutely necessary that the visual features also are tracked in the image, and that the pose is recomputed frequently.

In virtual visual servoing, pose computation involves a full scale non-linear optimization, and the problem of computing the pose is considered as similar to 2D visual servoing[5]. In visual servoing or image-based control, a camera is controlled in relation to its surroundings as described in section 4. The task is specified as regulation of a set of visual features in an image. Defining a number of constraints in the image space, a control law minimizing the error between the current and the desired position of the visual features can be automatically built[22]. Consequently, the camera position is modified according to this control law, ultimately converging to the desired pose if all goes well. As long as the combination of visual features are properly chosen, there will be only one final camera position minimizing the error function.

5.3.1 Principle

As mentioned above, the pose computation problem can in fact be considered as analogous to visual servoing. The principle can be illustrated by the following scenario. - Suppose that an object is made up by points, and define a virtual camera with intrinsic parameters ϑ. Furthermore, let the camera pose relative to the object be described by the homogeneous matrix T_o^c. The parameters of this matrix are the extrinsic parameters, introduced in section 3. It follows that the position of the object points P^c in the camera frame is given by:

$$P^c = T_o^c P^o \tag{92}$$

and its projection in the image is given by:

$$p = pr_\vartheta(P^c) = pr_\vartheta(T_o^c P^o) \tag{93}$$

in which pr_ϑ denotes the projection model with intrinsic parameters ϑ. The extrinsic parameters are what we wish to estimate. This is done by minimizing the error between the observed features in the image and the position of those same features computed by back-projection, corresponding to equation (93). The optimization is now performed by moving the virtual camera, using a visual servoing control law. When the error is below a given threshold, minimization is succesful and the pose is available.

5.3.2 Method Description

The features extracted from the real image are denoted p_{m_d}, and the corresponding features computed by back-projection p_m. Consequently, the error that we wish to minimize is $||p_{m_d} - p_m||$. The task function e is defined by

$$e = C(p_m(r) - p_{m_d}) \tag{94}$$

The matrix C is called the combination matrix, and r denotes the camera extrinsic parameters matrix. The combination matrix C is chosen such that CL_{p_m} is full rank. The differential is given by

$$\dot{e} = \frac{\partial e}{\partial r}\frac{\partial r}{\partial t} = CL_{p_m}\xi_c \tag{95}$$

in which the matrix L_{p_m} is the interaction matrix or image jacobian relating the motion of the visual feature in the image to the camera velocity ξ_c

$$\dot{p_m} = \frac{\partial p_m}{\partial r}\frac{\partial r}{\partial t} = L_{p_m}\xi_c \tag{96}$$

Specifying an exponentially decoupled decrease of the error

$$\dot{e} = -\lambda e \tag{97}$$

in which λ is a coeffiecient tuning the decay rate, the control law can be derived. By equation (95) and 97 we see that

$$CL_{p_m}\xi_c = -\lambda e \tag{98}$$

which gives us the ideal control law

$$\xi_c = -\lambda(CL_{p_m})^{-1}e \tag{99}$$

The interaction matrix depends on the pose between the camera and the target and on the value of the visual feature. In practice, we will not know this matrix explicitly, so a model $\hat{L_{p_m}}$ is used. Hence,

$$\xi_c = -\lambda(C\hat{L_{p_m}})^{-1}e \tag{100}$$

If we insert the control law (100) into equation (95) we obtain the behaviour of the closed loop system

$$\dot{e} = -\lambda(CL_{p_m})(CL_{p_m})^{-1}e \tag{101}$$

It is important that convergence and stability are ensured, and the positivity condition

$$(CL_{p_m})(CL_{p_m})^{-1} > 0 \tag{102}$$

is enough to obtain a decay in $||e||$, implying global asymptotic stability of the system.

It is possible to consider different choices of C and $\hat{L_{p_m}}$. The dimension *k* of the visual feature vector p should be greater than 6, such that the chosen visual features are redundant. The combination matrix C must be of dimension 6 x *k*, and the most obvious solution, is to define C as the pseudo inverse of the interaction matrix.

$$C = \hat{L_{p_m}}^{\dagger} \tag{103}$$

By this choice $C\hat{L_{p_m}} = I_6$ and stability is ensured as long as $\hat{L_{p_m}}^{\dagger} L_{p_m} > 0$.

It is also necessary to choose an estimate for the interaction matrix, $\hat{L_{p_m}}$. Possible candidates include $\hat{L_{p_m}} = L_{p_m}(p_{m_d}, r_d)$, in which the the interaction matrix is computed only once, with the final values of the pose and the visual features. In fact, this is a common choice in many robotics systems. However, in the context of pose computation, r_d is not known; it is what we seek to estimate.

A second alternative is to consider the choice $\hat{L_{p_m}} = L_{p_m}(p_{m_i}, r_i)$, in which r_i is the initial pose of the virtual camera and p_i is the initial value of the visual features. In this case, the positivity condition (102) will be fulfilled only if $p_{m_i} - p_{m_d}$ is small.

Another possibility is to choose the estimate of the interaction matrix as $\hat{L_{p_m}} = L_{p_m}(p_m, r)$. By this approach, the interaction matrix is computed at each iteration, using the current values of the pose and visual features. In this case, since the values of the elements of C varies, only local stability can be assured. If the error $p_m - p_{m_d}$ is too big, convergence may not be reached. Although only local stability can be shown, this is actually the chosen solution in the implementation of VVS. However, as long as the object displacement between successive images is not too great and if at the same time p_m and r are initialized with their respective values calculated from the previous image, convergence is not an issue. In practice, experiments have shown that convergence is always reached when the camera displacement has an orientation error less than 30° on each axis. Potential problems may therefore arise only for the initial image, when the initialization has been too coarse [22].

6 Image Features Extraction and Tracking

6.1 Introduction

Computer vision, and in particularly, vision based control seek to interact with the physical world, based on input from images. In order to do so, we need to relate information encapsulated in the images with additional knowledge of the structure of scene, velocities of the camera, and so on. Extracting information from the images is not a trivial task, and it has been a research topic for a long time. The thing about images and specifically, image processing, is that a lot of data has to be handled. A typical image, in vision based control applications, has dimensions 640 times 480 pixels. These, in total, 307200 pixels, somehow describe the instantaneous scene that the camera is witnessing. A human being is immediately capable of extracting a lot of qualitative information from this image. A computer algorithm however, may need to consider each and every pixel in conjunction with a multiple number of other pixels, searching for pre-defined patterns. Even though processor speeds continue to increase with time, image processing algorithms still need to be very cleverly designed, if they should have any chance to operate in real-time.

6.2 Moving Edges

The moving edges algorithm[3] presents an efficient method for tracking edges in an image sequence. The approach uses modeling principles and likely hypothesis testing techniques. A spatiotemporal edge in a series of images is modeled as a surface patch in a three dimensional spatiotemporal space. By using a likelihood ratio test the detection of an edge is accomplished, along with an estimate of related attributes. A great advantage of the algorithm is that only point coordinates and image intensities are manipulated. No prior edge extraction is required[11]. As a result, the moving edges algorithm can be implemented with convolution efficiency, which implies real-time performance[24].

6.2.1 Method Description

A list of pixels L along the contour of the edge is considered at time t. In a normal displacement computation process, the "‘corresponding"’ pixel p_i^{t+1} of every point $p_i^t \in L$ is sought in the following image I^{t+1}. The search is performed in the direction normal to the contour. A one dimensional search interval $\frac{j}{i},\ j \in [-J, J]$, in the direction δ of the normal to the contour is chosen, as described in Figure 9. Next, a criterion corresponding to the square root of a log-likelihood

[11]Note that we are here talking about *tracking* an edge, which at some earlier point was extracted from an initial image

ratio ζ^j is computed for every point $p_i^t \in L^t$, and for every complete position Q_i^j, which for computational issues lie in the direction δ^* nearest to δ, from the set $\{0°, 45°, 90°, 135°\}$. This ratio is just the absolute sum of the convolution values, computed at p_i^t and Q_i^j, respectively, in images I^t and I^{t+1}, using a predetermined mask M_δ function of the orientation of the contour[3].

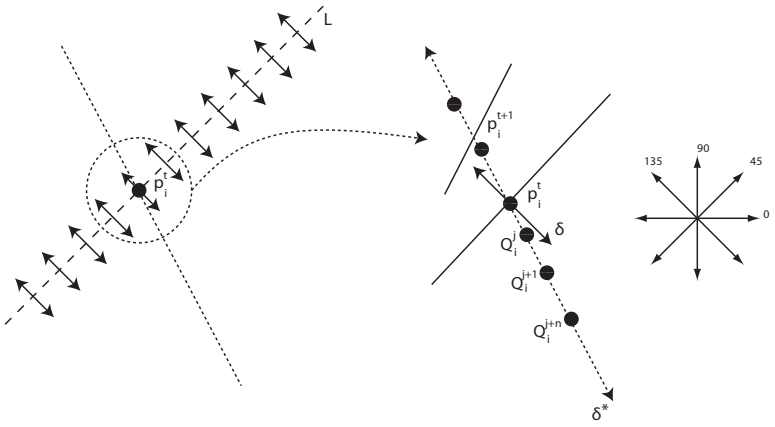

Figure 9: Determining point positions of a tracked object contour from one image to the next using the Moving Edges Algorithm.

Accordingly, the position of point p_i at time $t+1$ is found by solving

$$Q_i^{j^*} = \arg \max_{j \in [-J,J]} \zeta^j \tag{104}$$

where

$$\zeta^j = |I^t_{\nu(p_i)} * M_\delta + I^{t+1}_{\nu(Q_i^j)} * M_\delta| \tag{105}$$

, ζ^{j^*} is larger than a threshold λ, and $\nu(\cdot)$ represents the neighbourhood of the pixel in question. The pixel p_i^{t+1} satisfying equation (104) is then added to the list L^{t+1}.

We now have a list containing k pixels and their corresponding displacement components orthogonal to the object model contour $(p_i^t, d_i^\perp)_{i=1\ldots k}$. This procedure is repeated on the arrival of each new image frame, as pointed out, never actually extracting a new contour, only manipulating point coordinates. Another advantage comes from the fact the method is strictly local. Thus, any partial occlusion of the considered contour only leads to loosing some local measurements, not the actual contour itself.

6.2.2 Dot Tracking

A dot is defined by connex pixels with the same gray level. Without going into details, the underground algorithm is based on a binarisation of the image, followed by a contour detection step using the so-called Freeman chain coding to determine the characteristics of the dot such as location and size.

7 State Prediction

7.1 Introduction

Assuming temporal continuity of the motions experienced by the system, it is possible to improve the performance of the system by implementing filters and prediction. Introduced in 1960, the much celebrated Kalman Filter was instantly embraced by the automatic control community. A wealth of both basic and advanced texts on the subject exists. One extensive introductory reference is [4].

7.2 The Kalman Filter

In many applications the main reason for applying a Kalman filter is to filter out noise from the measurements, hence achieving better estimates of the states. In this case, we assume that the estimates computed by the pose computation algorithm already are fairly accurate. When the goal object is at rest, we want the the system to position the end effector at a desired pose relative to this target. In this case, the desired pose, in a world-coordinate system, does not explicitly change with time. It may change as a result of updated estimates, as the manipulator closes in on the target, but we know that the target itself will remain at rest. It is clear that a pose estimate is not instantly available, it must be calculated by our system. These calculations introduce a delay in the system, such that the measurement y is actually an estimate of the past states. In the case of a non-moving target, this time delay does not affect the systems performance much. However, when the target is moving and the flow of measurements are used for generating time-varying set points for the manipulator, it is clear that any time delay will cause the manipulator to constantly stay at least one step behind. The system will also need a certain amount of time to actually reach the set-point, introducing further delay.

If the target is moving in an unpredictable manner, the scheme presented above is the best we can do, and we will have to live with the error implied by the delay. However, if we assume that the target moves as a periodic function of time (possibly with slowly varying characteristics) we can model this motion and use it to predict the future states of our system. If the model is accurate enough, this means that we should be able to eliminate most of the error introduced by the time delay. This is the approach taken in the second part of this thesis.

7.3 Modeling the Target Motion

An undamped harmonic oscillation with frequency ω is simply described by the second order differential equation

$$\ddot{x} + \omega^2 x = 0 \tag{106}$$

which on state-space form becomes

$$\dot{X} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{107}$$

Assuming that the targets movement in one DOF can be described by a sinusoidal with frequency ω, and that the state of the end-effector is only affected by the input signal u, the state-space model describing the target and the end-effectors motion in one DOF becomes

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 \\ -\omega^2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \tag{108}$$

Recall that what we are measuring is the the relative pose between the end-effector and the target, consequently the measurement becomes

$$y = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{109}$$

7.3.1 More Complicated Target Motion

If we assume the targets motion in one DOF can not be described by a single sinusoidal, but rather as the sum of several sinusoidals with different frequencies, our model can easily be expanded to include these

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\omega_1^2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -\omega_2^2 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & & & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & -\omega_i^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{2i-1} \\ x_{2i} \\ x_{2i+1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix} u \tag{110}$$

$$y = \begin{bmatrix} 1 & 0 & 1 & 0 & \cdots & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{2i-1} \\ x_{2i} \\ x_{2i+1} \end{bmatrix} \tag{111}$$

7.4 Unknown Frequencies

In the above sections we assumed that the frequencies of the motion that we were trying to follow, were known and constant. However, we made no assumptions regarding phase or amplitude. Now, let us assume that also the frequencies are unknown. Our only assumption now is that we know how many frequency components are needed to describe the motion. This assumption is not really necessary. In case we did not know this number, we could keep augmenting our system model with more frequency components until the model was satisfactorily accurate.

Including the frequencies of the motion as variables to be estimated, we can write the model as

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= x_3^2 x_1 \\ \dot{x_3} &= 0 \\ \dot{x_4} &= x_5 \\ \dot{x_5} &= x_6^2 x_4 \\ \dot{x_6} &= 0 \\ &\vdots \\ \dot{x_{3i-2}} &= x_{3i-1} \\ \dot{x_{3i-1}} &= x_{3i}^2 x_{3i-2} \\ \dot{x_{3i}} &= 0 \\ \dot{x_{3i+1}} &= u \end{aligned} \tag{112}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_{3i-2} \\ x_{3i-1} \\ x_{3i} \\ x_{3i+1} \end{bmatrix} \tag{113}$$

Including the frequencies as variables, the model becomes non-linear, and we need to apply an extended Kalman filter. The main difference is that we need to linearize the model at each time-step. The linearized model becomes

$$\frac{\partial f}{\partial x}\Big|_{X=X_k} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -x_3^2 & 0 & -2x_3x_1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -x_6^2 & 0 & -2x_6x_4 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & & & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -x_{3i}^2 & -2x_{3i}x_{3i-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \end{bmatrix}\Bigg|_{X=X_k} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \\ \Delta x_6 \\ \vdots \\ \Delta x_{3i-2} \\ \Delta x_{3i-1} \\ \Delta x_{3i} \\ \Delta x_{3i+1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Delta u \tag{114}$$

7.5 Complete System

Knowing how the target object moves in a body fixed frame, the next step is to relate the body fixed velocities of the object to an inertial frame. The robot manipulator is typically controlled by directing the end effector, relative to an inertial base frame. Thus, knowing the motion of the target object, in the inertial frame, allows us to, at least in theory, track it with the end effector. We assume that the pose estimates of the goal object are given in the inertial frame of the manipulator. Furthermore, we assume that the orientation of the goal object is described in terms of Euler angles, by the zyx convention. Let p^i be the position of the target object, in the inertial frame. Then, the body-fixed velocity vector v_o^b decomposed in the inertial frame, can be described by

$$\dot{p}^i = R_b^i(\Theta) v_o^b \tag{115}$$

The body-fixed angular velocity vector $\omega_{i,b}^b$ and the Euler rate $\dot{\Theta}$ are related through a transformation matrix $T_\Theta(\Theta)$ as

$$\dot{\Theta} = T_\Theta(\Theta)\omega_{i,b}^b \tag{116}$$

The assumption of zyx Euler angles, implies that the transformation matrix $T_\Theta(\Theta)$ should satisfy

$$\omega_{i,b}^b = \begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} + R_{x,\phi}^T \begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + R_{x,\phi}^T R_{y,\theta}^T \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} := T_\Theta^{-1}(\Theta)\dot{\Theta} \tag{117}$$

Expanding the above equation gives us[13]

$$T_\Theta^{-1} = \begin{bmatrix}1 & 0 & -s_\theta\\0 & c_\phi & c_\theta s_\phi\\0 & -s_\phi & c_\theta c_\phi\end{bmatrix} \Longrightarrow \qquad T_\Theta(\Theta) = \begin{bmatrix}1 & s_\phi t_\theta & c_\phi t_\theta\\0 & c_\phi & -s_\phi\\0 & s_\phi/c_\theta & c_\phi/c_\theta\end{bmatrix} \tag{118}$$

Note that by using this definition, $T_\Theta(\Theta)$ is undefined for $\theta = \pm 90^\circ$. Alternatively, quaternion representation could be used, avoiding the potential problem of singularities. However, in this context, it is regarded as highly unlikely that the object that we are tracking, should be able to rotate $\pm 90^\circ$ in any direction. Hence, the use of Euler angles should not cause any difficulties.

Combining position and orientation in a pose vector

$$\eta = \begin{bmatrix}p^i\\\Theta\end{bmatrix} \tag{119}$$

and linear and angular body-fixed velocities in a velocity vector

$$\xi = \begin{bmatrix}v_o^b\\\omega_{i,b}^b\end{bmatrix} \tag{120}$$

the 6 DOF kinematic equations can be arranged as

$$\begin{bmatrix}\dot{p^i}\\\dot{\Theta}\end{bmatrix} = \begin{bmatrix}R_b^i(\Theta) & 0_{3\times3}\\0_{3\times3} & T_\Theta(\Theta)\end{bmatrix} \begin{bmatrix}v_o^b\\\omega_{i,b}^b\end{bmatrix} \Leftrightarrow \dot{\eta} = \Pi(\eta)\xi \tag{121}$$

We now know how to relate the body fixed velocities of the goal object to an inertial frame. However, since our description of the goal objects movement is given in either forms as in equation(114) or equation(110), depending on whether the involved frequencies are known in advance or not, we need to relate these expressions to equation(121). Let Λ^o be the vector formed by stacking the vectors X, Y, Z, Φ, Θ, Ψ, in the target object frame, where each vector $X, ..., \Phi$ is on the form $X = [x_1, x_2, ..., x_n]^T$. In the same way, Λ^i is the corresponding vector decomposed in the inertial frame. Furthermore, we denote Γ as the augmented Π matrix, formed by stacking each successive row of Π with multiplicity n on top of each other. If we denote row i of Π with multiplicity n

as $\Pi(i)$, such that $\Pi(i) \in \mathbb{R}^{n\times 6}$ the augmented system is described by

$$\dot{\Lambda}^i = \begin{bmatrix} \Pi(1) \\ \Pi(2) \\ \Pi(3) \\ \Pi(4) \\ \Pi(5) \\ \Pi(6) \end{bmatrix} \Lambda^o = \Gamma\Lambda^o \tag{122}$$

7.6 The Kalman Filter Equations

The robot control system at ABB allows for setting desired position of the end-effector, and either the time by which it should reach this position, or the velocity by which it should be moving. In this case, the command updating frequency is limited to between 2 and 5 Hz. It is also possible to command the robot by only specifying a desired position, without specifying velocity or time, in which case a higher command updating frequency can be accomplished, approximately 10 Hz.

Besides the specification of the robot control system, we also need to consider the estimates provided by pose computation. This whole procedure is quite computational demanding, as one can imagine. First, a new image needs to be captured, then, the image features need to be located, finally, the pose must be estimated. The time needed for these operations will depend on several factors. Obviously, the hardware/software configuration plays a major role. With a lot a processing power, time consumption should be significantly reduced. Besides this, the time needed will heavily depend on the motion experienced by system, specifically, the displacement of image features in the image, from one image to the next. A large displacement means that at larger area of the image needs to be searched, whereas small displacements have the opposite effect. Tests on the writers office setup suggests a computation time of approximately 130-140 ms for each iteration of the loop. Consequently, new estimates are available at about 7-8 Hz.

Since both measurements and control inputs are sampled in discrete time, we need to apply the discrete time Kalman filter. The equations for the discrete time Kalman filter are given in table 5

K_k	=	$P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$
$\hat{q}_k$	=	$\hat{q_k}^- + K_k(z_k - H_k \hat{q_k}^-$
P_k	=	$P_k^- + K_k H_k P_k^- - P_k^- H_k^T K_k^T + K_k(H_k P_k^- H_k^T + R_k) K_k^T$
$\hat{q}_{k+1}^-$	=	$\Phi_k \hat{q}_k + \Delta_k u_k$
P_{k+1}^-	=	$\Phi_k P_k \Phi_k^T + Q_k$

Table 5: Discrete Time Kalman Filter Update Equations

8 Implementation

'If you're going through hell, keep going' - Winston Churchill

8.1 Introduction

The practical parts of this thesis can roughly be divided into three main sections. The parts related to image processing and manipulation, that is - the features extraction, image feature tracking, and pose estimation based on the extracted features - where given the main priority for the greater part of the execution of this thesis. Without this part working, it is obvious that the overall goal of the thesis could not be fulfilled. Obviously, in order to track a moving object, the most basic part is to collect measurements that describe the spatial coordinates of this object.

In the second main section, the focus was on developing prediction schemes that would allow for the prediction of future states of the tracked object, based on the at-all-times current knowledge. This way, the control system should be able to position the manipulator, by using feed-forward from the predicted future states, ideally reducing the error in the robots coordinates relative to the target object as compared to the desired ones, to zero.

The last section was the implementation of the combined pose-estimation and tracking scheme on an industrial robot test facility.

8.2 Pose Estimation

The pose of the goal object, relative to the camera is estimated by the algorithm described in section 5.3. This algorithm basically relies on three things. First of all, we need to know how physical objects project onto images, for the specific camera in question. The parameters describing this relation are known as the intrinsic camera parameters. These are estimated through a camera calibration procedure, as described in section 3. The intrinsic camera parameters have been estimated for several different cameras, including a web camera, a Fire-Wire camera, a camera attached to the robot at the ABB lab, and a camera attached to the robot at the Sintef lab. Secondly, we need to locate reliable features in the sequence of images. The third thing that we need to know is how the image features relate to a each other in the physical world.

Different kinds of features may be considered. However, it is important that the system is capable of tracking these features from frame to frame. Possible image features include points, lines and curves. The first strategy was to consider more or less arbitrary shaped objects, without any fiducial markers, and automatically extract reliable features from these. A great deal of effort was put into this part, but the outcomes were not satisfying. Straight lines were ex-

tracted from images, and attempted tracked using the Moving Edges algorithm, described in section 6.2. Using this method, the problem with tracking straight lines that are considered separately, is that a tracked edge tends to evolve itself further and further away from the actual edge, as the camera moves, or as the lighting conditions changes. A possible solution to this problem would be to impose further constraints on the extracted edges, linking them together based upon the objects structure, rather than just considering them one by one.

Instead of further pursuing this idea, the choice was made to rather consider objects with spherical shaped markers on. Assuming that the object that we seek to track has some clearly defined dots either artificially attached to it, or that they simply exist as a natural part of the objects geometry, we can track these from frame to frame. The principle for the dot tracking is described in section 6.2. Assuming that the dots are successfully extracted from an image, we use the center of these dots as the image features that the pose computation will rely on.

For convenience, a black piece of paper with four white dots, attached to a planar box, has been used to describe the goal object. The target object is defined by a frame with origin in the common center between the dots. Figure 10 shows the orientation of this frame. The z-axis points into the picture, the x-axis points to the right and the y-axis points downwards. In this setting, the

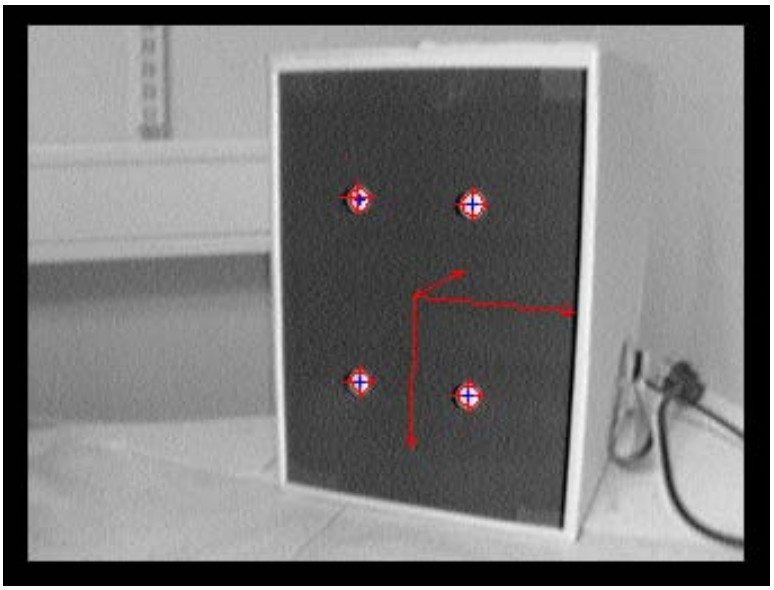

Figure 10: Object frame

algorithm described below is used to find the image features and link these features to their corresponding model features. We define the coordinates of each image feature in the object frame. Then, we define certain properties that describe the dot to search for. These include upper and lower bounds for light

Algorithm 1 Feature matching

```
define location of features in physical model
define dot ← brightness, dot ← size, dot ← ellipsoid precision
define detected dot list = {∅}
capture initial image

for entire image do
   search current area for dot
   if current search area matches dot then
      detected dot list ← dot
   end if
end for
if sizeOf( dot list ) == 4 then
   sort( dot list ) and link center of dots with features in physical model
end if
```

intensity and size, and also, to what extent the dot to search for should have an elliptical shape.

After the initial image has been captured, we search through the entire image looking for dots that match the pre-defined criteria. Four distinct dots must be found. If a dot is found, we put it in a list of discovered dots. If four dots are found we proceed to sorting the list. We are seeking the center point of each dot. These define the feature location in the image. The list is first sorted with respect to the features v values, then, by their u-values. When this is done, each feature is linked with the corresponding point in the physical model, as illustrated in Figure 11.

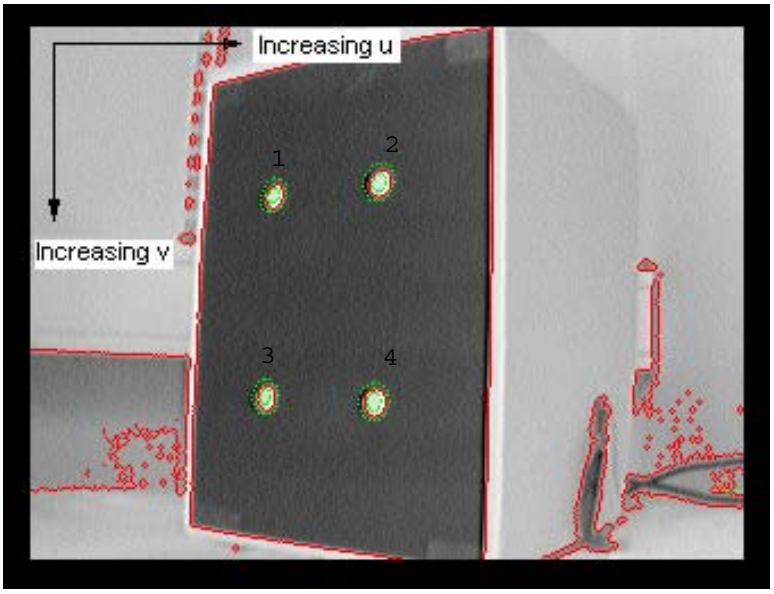

Figure 11: Dot detection and model matching

With a symmetrical object, as far as the model features are concerned, we need to make an assumption in order to guarantee that the right image feature is linked to the corresponding feature in the model. We assume that in the inital image, the camera is never rotated, relative to the object, to such an extent that the v values of the image features marked as 3 and 4 in Figure 11, exceed the v values of image features 1 and 2. However, assuming that the object that we are considering at least has a minimal degree of rigidity with respect to its surroundings, this should not be an issue.

Figure 12 shows a number of screenshots, depicting the estimated poses as illustrated by the objects' body fixed coordinate frames, seen in red. It may be hard to tell from the figure, but the center of a dot is marked by a red cross. In the dot center, we can also see a small blue dot. These blue dots represent the model features computed by back-projection, as described in section 5.3. In the centered leftmost view, the camera motion from the previous frame to the current frame has been relatively large such that a greater part of the image has had to be searched to locate the image features.

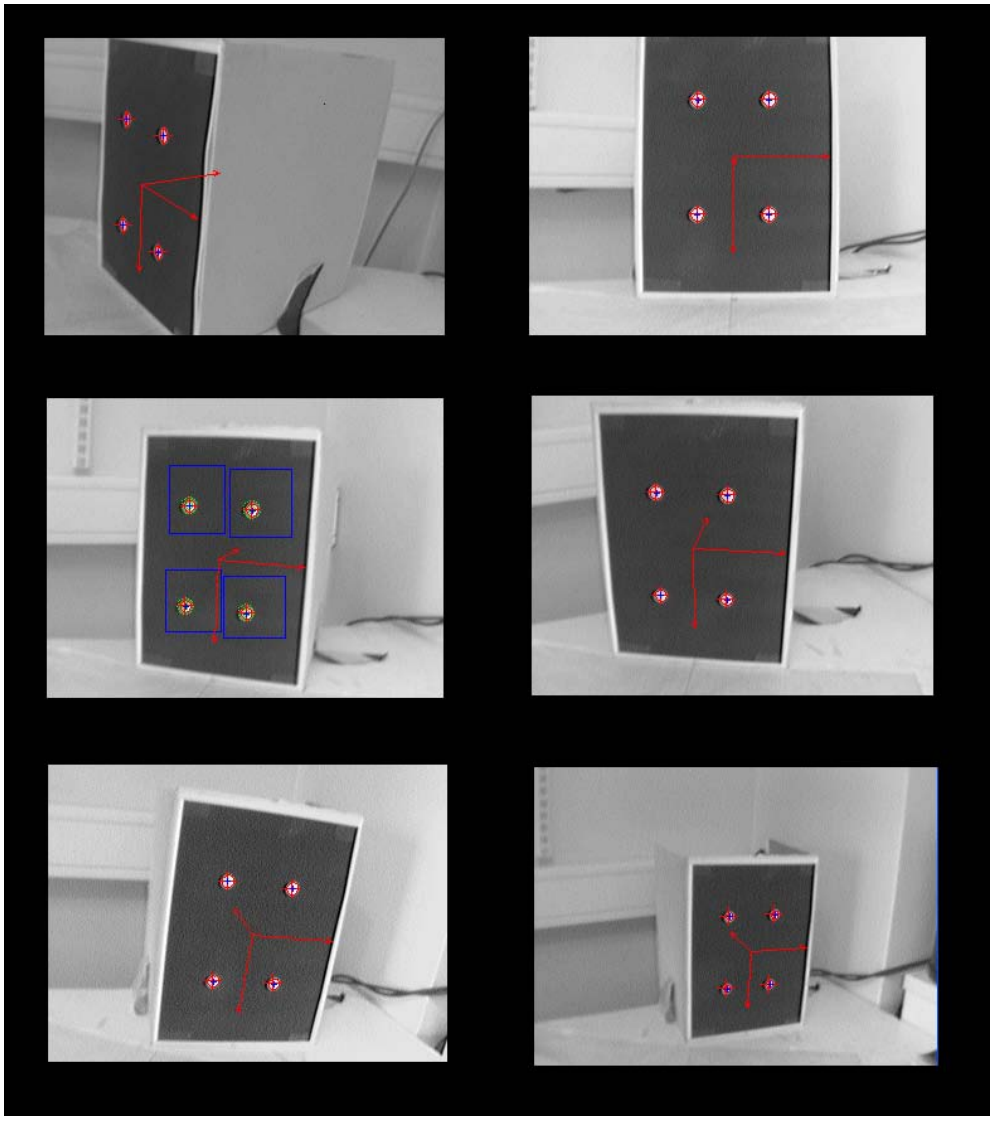

Figure 12: Computed pose visualized by the orientation and apparent size of the object's body fixed coordinate frame

8.3 State Estimation

The scheme for state estimation was developed through modeling and simulations in Matlab/Simulink. The core of this work is the prediction of motion, and is based on the models developed in section 7. Figure 13 shows a the model file describing a single DOF. As a target object is moving, we are trying to identify the motion, and and predict future states. If we are able to do so, we can use the estimates for feed-forward control of the robot.

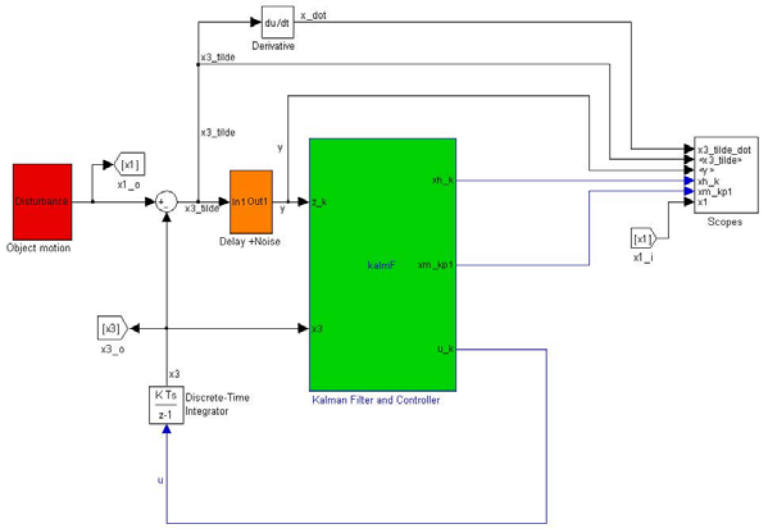

Figure 13: Model representing a single DOF

In the following figures the target object's motion is described by a single sinusoidal with known frequency. The measurements are sampled, delayed by 150 milliseconds, and colored by noise with a given power level. Figure(14) shows the error between predicted and real values of the object's position. Figure 15 shows the tracking error. The tracking error describes the difference between realised position and desired position, relative to the target object. It should be noted that tuning the regulator has not been given much considerations. The effort has been focused on the performance of the Kalman filter, and specifically.

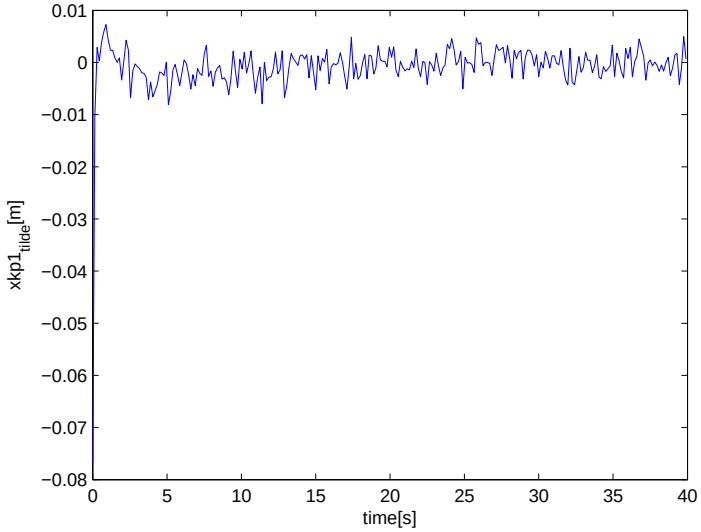

Figure 14: Prediction error. Target moves like one sine.

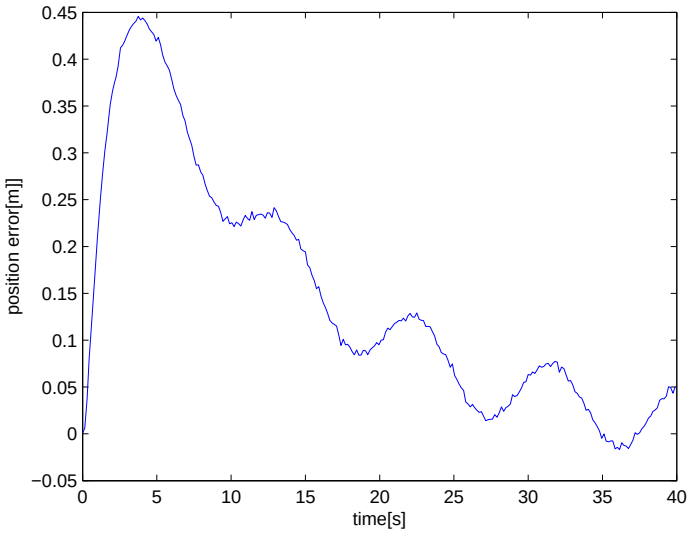

Figure 15: Tracking error. Target moves like one sine.

In the following, the target object moves as the sum of two sine waves in one DOF - $x(t) = 0.5\sin(2\pi/9) + 0.3\sin(2\pi/11)$ In this case neither the amplitudes, phase, nor frequencies of the motion is known in advance. Figure16 shows the error between estimated and real position of the target object. For the record,

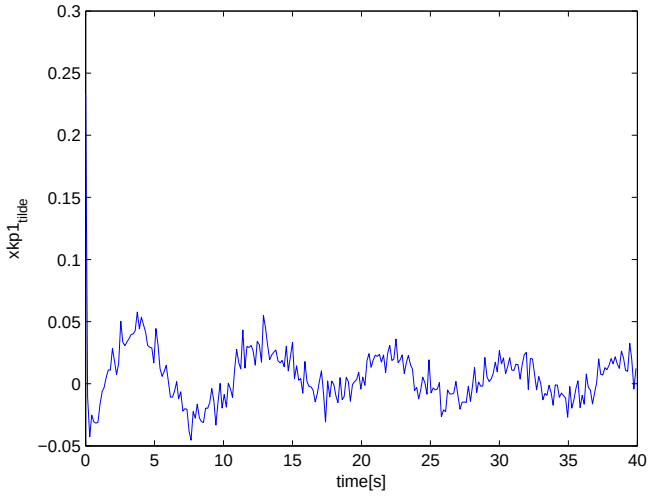

Figure 16: Prediction error. Target moves like the sum of two sine waves that are unknown for the Kalman filter.

Figure 17 shows the real motion exhibited by the target object.

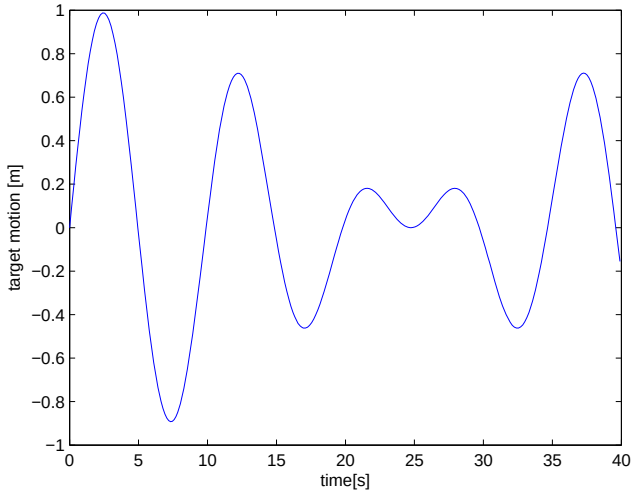

Figure 17: Target motion

8.4 Implementation at ABB

The idea was to implement a combined pose computation and tracking scheme at the ABB test facilities. The ABB setup includes a small scale process plant, a robot manipulator attached to a gantry crane, capable of moving in a rather large area, in addition to two more manipulators attached to rails. The gantry crane mounted manipulator would be used for vision-based tracking. Based on what was possible to do, we agreed on the procedure described briefly summarized in 2.

Basically, the robot manipulator would be starting out such that the goal object was within the cameras field of view. The end-effector coordinates at this initial position would define a reference frame in which any future command had to be given. Denote o^0 as the reference frame, then $T^o_{e(t)}$ defines the transformation from the origin of o to the end-effector position at time t. Furthermore, denote the desired move relative to this end-effector position as, at time t as $T^{e(t)}_{e_d(t)}$. The desired position decomposed in the reference frame becomes

$$T^o_{e_d(t)} = T^o_{e(t)} T^{e(t)}_{e_d(t)} \tag{123}$$

The robot control system requires that we express the translational part of a desired pose as x-y-z values, while orientation must be given as either Euler angles or unit Quaternion. Both are fairly easily extracted from the rotation matrix[12]. If we stick to the Euler angle zyx convention, the angles can found according to equation (5).

Figure 18 depicts how the main coordinate frames of the system are related. With a mild abuse of notation o_i denotes the inertial frame, which does not necessarily coincide with the actual inertial frame of the system as a whole, but rather it is defined as the initial frame in which all future end-effector coordinates are expressed with respect to.

Denote o as the target object frame. It follows that the transformation from the inertial frame to o is given by

$$T^i_o = T^i_e T^e_c T^c_o \tag{124}$$

The relation between the end-effector and the target object is given by

$$T^e_o = T^e_c T^c_o \tag{125}$$

Denote the desired pose of the end-effector relative to the target object as $T^e_{o\,d}$, and relative to the inertial frame, as $T^i_{e\,d}$. It follows that the desired pose of the end-effector in the inertial frame can be expressed as

$$T^i_o = T^i_e T^e_c T^c_o = T^i_{e\,d} T^e_{o\,d}$$

[12] The upper right 3×3 matrix of $T^o_{e_d(t)}$

$$\Downarrow$$

$$T^i_{e\,d} = T^i_o (T^e_{o\,d})^{-1} \tag{126}$$

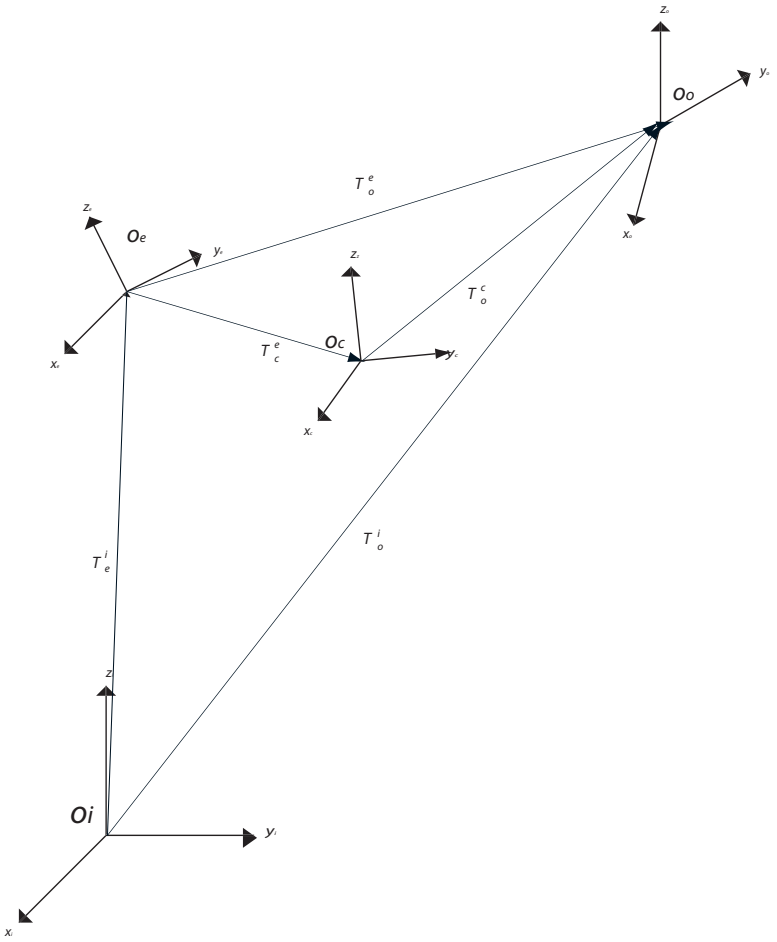

Figure 18: Relation between different frames

The author travelled to Oslo and stayed there for a week. The first problem to overcome was due to the different programming languages used by the undersigned and the developers at ABB. The writer has done all the coding, except for the Matlab/Simulink simulations, in C++. ABB, on the other hand, do most of their coding in C#. Since neither the author nor the personnel at ABB had any experience in cross language programming, different solutions where considered. One solution was to export the code from the writers system to the ABB side, by using dynamic link libraries, dlls. With lack of experience this seemed like a very time consuming job, as it concerned large amounts of data. Another solution would be for the two systems to operate separatly, reading and writing data to files. However, this did not seem like a feasible solution for a system that is supposed to be working in real-time. The third possibility, was to let the

two systems run separatly, but to communicate via TCP/IP[13].The choice fell on this option. Networking was also a but new topic for the author.

Algorithm 2 Tracking ABB

init : **read** end-effector coordinates **and define** coordinate system start
receive end-effector coordinates **and** image **from** robot control system
estimate pose **and** predict future pose
send predicted pose **to** robot control system
goto receive

We needed to figure out how to send images and robot end-effector coordinates from the ABB side to the authors side. Then, do the necessary computations and send back the desired end-effector coordinates to the robot control system. When the first part was solved, that is, successfully sending and receiving images, new problems arose. It seemed that the libraries used for image processing did not support the image formats available from the sender side. Although we at that point managed to convert the received images to the "required" format, by using another external image processing library, it turned out that there were even differences within this specific format. As a result, we were unable to process these images. The week went by and no experiments were conducted.

8.5 Implementation at Sintef

Since we did not get to do any experiments at ABB, alternative solutions were considered. Sintef division for Applied Cybernetics has got a robot lab which resembles the ABB setup. As far as the workspace is concerned, it is basically a somewhat a downscaled version of the setup found at ABB, with one robot less. However, integrating the authors system with the existing one at Sintef, a lot more problems were initially unresolved, as compared the ABB setup.

The authors system would need to communicate through TCP/IP with several other systems. One server provides images, one server provides robot joint angles and a third server can receive coordinates that are used to control the robot. The author still had to resolve the problem that remained from the attempt at ABB, namely the conversion of images to the right format whilst keeping data in computer memory.

Having solved this problem, communication with the different servers had to be resolved, as well as computing the forward kinematics of the manipulator, based on received joint angles. Figure 19 shows a principle description of the

[13]TCP/IP is an abbreviation for Transmission Control Protocol/Internet Protocol. Shortly described, it is a gathering of communication protocols to connect computers in a network.

manipulators forward kinematics, and Table 6 summarizes the DH parameters for robot.

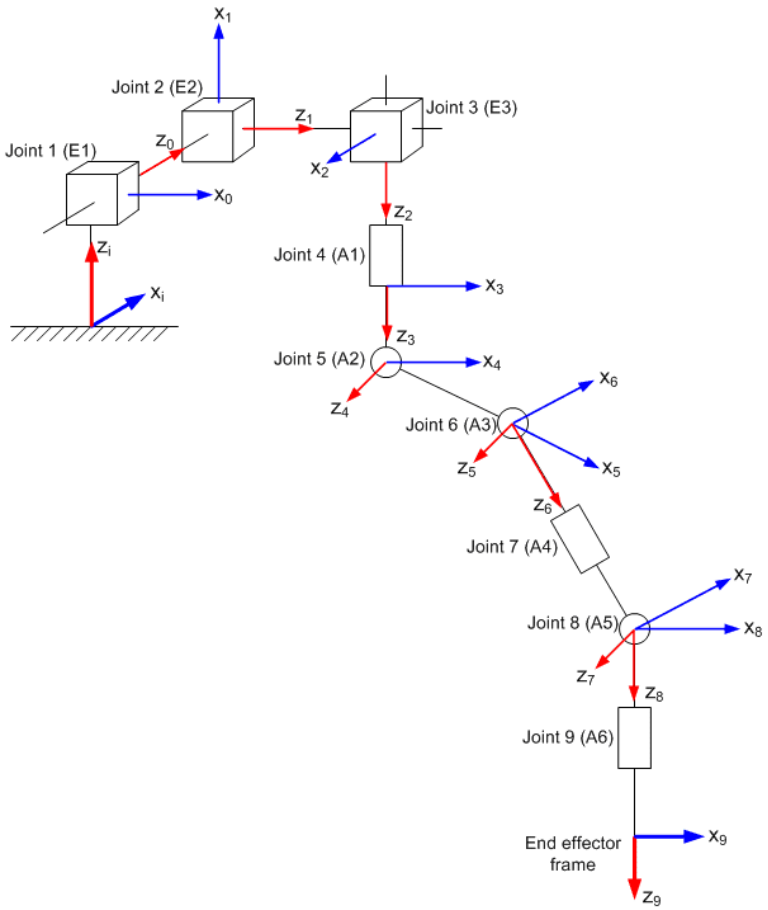

Figure 19: Forward kinematics of the Sintef robot

Solving these parts turned out to be cumbersome. Without going into details, getting message byte orders correct for sending and receiving took a lot time. Even so, it turned out that DH parameters given did not describe the forward kinematics correctly. To overcome this problem, A. Transet and T. Mugaas, who works at Sintef, found a way of sending end-effector coordinates directly, instead of joint angles. This is not normal procedure, but due to the situation at hand it was decided to do so anyway.

Before starting the experiments, there was one last practical problem to solve. We needed to identify the static transformation from the end effector frame to the camera frame, since this was not know in advance.

The idea was to place the goal object as accurately as possible in a known pose relative to the inertial frame. By orienting the object such that its axes are in line with the axes of the inertial frame, the transformation from inertial frame to object frame becomes a pure translation. If the effector is positioned

Table 6: DH parameters for the Sintef robot

Link	a_i	α_i	d_i	θ_i
1	-0.4	$-\pi/2$	q_1	$-\pi/2$
2	0	$-\pi/2$	q_2	$\pi/2$
3	0	0	q_3	$-\pi/2$
4	0.26	$-\pi/2$	0.675	q_4
5	0.68	0	0	q_5
6	0.035	$\pi/2$	0.07	q_6
7	0	$-\pi/2$	0.67	q_7
8	0	$\pi/2$	0	q_8
9	0	0	0.15	q_9

relative to the object, such that its coordinate axes are aligned with the axes of the inertial frame, they will also be aligned with the axes of the goal object. Hence, the transformation from the end-effector frame to the camera frame will also be a pure translation. Let T_o^i denote transformation from the inertial frame to the object frame and let T_e^i be the transformation from the inertial frame to the end-effector frame. Subsequently, positioning the end-effector such that the goal object is within the cameras field of view, we can find the homogeneous transformation from the end-effector frame to the camera frame by

$$\begin{aligned} T_o^i &= T_e^i T_o^e \\ &\Downarrow \\ T_o^e &= (T_e^i)^{-1} T_o^i \end{aligned} \tag{127}$$

Following this procedure, the identified transformation did not seem to make any sense. As it turned out, the coordinates that were supposed to describe the position and orientation of the end effector, did not really provide this information at all. Possibly, the coordinates that we received from the system, may be related to a joint further back in the kinematic chain.

Without knowing neither the configuration of the robot, the coordinates of the end effector nor how to command it to desired coordinates, we could not perform any experiments. It should be noted that the way that we were trying to control the robot, is not how the staff at Sintef normally do it. And by their own approaches things do work properly. It should also be mentioned that due to the nature of the situation, that is, the author requesting to use Sintefs lab at such a late stage, it is clear that author could not simply be granted to use this very expensive equipment without restrictions. Consequently, the author could only access fractions of the complete system during the implementation attempt. The camera that was used here, is the Fire Wire camera described in section 3.

8.5.1 Validation of the Pose Estimates

Since we did get not get a chance to conduct real experiments in a highly controlled environment, an alternative solution was needed to verify the pose estimates provided by the algorithm. With very little time left, it was not possible to obtain accurate data for comparison. By using millimetre paper and relying on the authors eye measurements, comparisons where made between "true" values for position and orientation, and the estimated ones. Table 7 and Table 8 show the comparisons for five different data sets. η_i refers to the "real" pose, while $\hat{\eta}_i$ refers to the estimates provided by the pose computation algorithm.

Table 7: Comparison between "true" and estimated values for the pose - 1

Data	η_1	$\hat{\eta}_1$	η_2	$\hat{\eta}_2$	η_3	$\hat{\eta}_3$
x	0.0	-0.44	0.0	0.50	-2	-2.2
y	-4.0	-3.96	-4.0	-4.2	-4	-4.1
z	39.0	39.1	43.0	42.9	40	38.9
ϕ	0.0	-4.6	0.0	-4.5	0.0	-4.0
θ	0.0	-1.6	0.0	4.5	0.0	1.0
ψ	0.0	0.5	0.0	0.1	0.0	0.4

Table 8: Comparison between "true" and estimated values for the pose - 2

Data	η_4	$\hat{\eta}_4$	η_5	$\hat{\eta}_5$
x	0.0	-0.5	0.0	-0.2
y	-4.0	-3.7	-4.0	-3.7
z	40.0	40.3	40.0	40.7
ϕ	0.0	-0.3	0.0	-0.5
θ	45.0	-45.0	45.8	-44.1
ψ	0.0	0.0	0.0	1.3

The x, y and z values are given in centimetres. The angles ϕ, θ and ψ denote the Euler angles, according to the zyx convention, and are given in degrees.

It clear that the obtained results do not qualify as scientifical data. The setup is very roughly organized. Hence, we cannot draw any absolute conclusions regarding the accuracy of the pose estimates. In the three first sets, it looks as if the estimated angles are a bit inaccurately estimated, showing errors of a few degrees, but in the next two sets the errors are considerably smaller. The most likely explanation for this is that author has been a little too inaccurate in the arrangement phase. As stated, the experiments are imprecise. However, based

on the comparisons, it is reasonable to assume that algorithm does provide good estimates.

9 Discussion

One of the hopes, prior to the execution of this thesis, was that at the end of the line we would have demonstrated a fully working pose-estimation and tracking system at the ABB test facilities in Oslo. However, anyone who has ever had to transfer modules developed on one system to a different setup, knows that this may be a lot easier said than done.What is supposedly just minor differences, can turn out to be major difficulties that need to be overcome. This is exactly what happened in the implementation stage at ABB. Even so, dealing with real hardware it seems inevitable to stay clear of all obstacles on the way. In light of what happened, it is clear that the undersigned should have taken further steps in the preparation phase, before going to Oslo. At the same, the workload on the author has continuously been large, and for that reason, it has not always been possible to do all the things that one had wished for.

Another practical problem arose from the fact that the initiator of this thesis - ABB Strategic R&D for Oil&Gas - is located in Oslo. They agreed to pay for the flights there and back, but could not compensate for expenses associated with accommodation. With this restriction, it is clear that it is only possible for a not-too-wealthy student to be staying somewhere for a fairly limited period of time. As a result, the undersigned made one trip to Oslo, that lasted for one week. The system did not become operational during this time.

As an alternative solution, Sintef - division for Applied Cybernetic - was kind enough to let us use their robotics test lab, which is quite similar to the ABB lab, at first glance. Sintef, in their experiments, are doing things by a completely different approach than what we would be attempting to do. Before starting, we were informed by dr. Axel Transeth, that it would surely require a great deal of effort to make the necessary adjustments to get an operational setup. This turned out to be true. Intensive work was put into the task of trying to solve all the details. However, just at the finishing line, all hope vanished when it turned out the we were unable to get the accurate, or even remotely close to accurate, data that we needed from the robot, in order to control it. Sadly, this meant that we would not get to see the system tested on a real robotic system. However, the separate parts themselves are functioning. The interface between the system developed by the author and the ABB robot system, should also be working, with little effort. This is due to the fact all the problems faced during the initial stay at ABB were solved while trying to make the system operational in the Sintef setup. Surely, some work will be needed at ABB, but at least for a relatively simple testing scenario, the problems faced should severe.

As a concluding remark: the author has spent a lot of time and effort working on this thesis, and hopefully, the results will come to use. Implementational

issues, both initially and at later stages, have required most of the author's time throughout. Although parts of this thesis were written in parallel with the practical parts, many things were put on hold, trying to achieve the desired results. In shortage of time, parts of the written thesis has had to suffer the consequences.

10 Suggestions for future work

With respect to the ABB setup, the undersigned believes that it should not require much work before initial vision-based experiments can be made. That is, if ABB choose to go further with what has been done in this thesis. The system should be capable of positioning the end-effector relative to a target object with reasonable accuracy, as long the necessary assumptions are made.

Furthermore, depending on the true nature of real target objects movement patterns, not to say the amplitude of such motion, the available resources for processing power, and the requirements of the system, further work can take on different directions.

Assuming that the computational power available is but unlimited, image processing and pose computation can potentially be done very fast, introducing only negliable time delays in the system control loop. If, at the same time, the robot response time is kept low enough, the system as a whole should be able to react to disturbances very fast, enabling accurate real-time tracking of a moving object. In this setting, there is no need to introduce any additional state model, or state prediction, as it will not improve the systems overall performance.

If, on the other hand the overall time delay in the system loop is not negliable, and if the target object exhibits smooth motion, the control system is likely to benefit greatly from introducing feedforward from an observer, such as a Kalman filter. The state space model describing the system, is perhaps the most essential part of the observer, and with an accurate[14] model highly precise predictions can be made.

A different aspect relates to the low level control of a robot. Certainly, industrial robots cannot afford to make mistakes, and it follows that the robot should behave in a predictable manner at all times. At the same time, in the context of vision-based control, it is not very common to control the robot strictly by giving it coordinates that it should go to. Controling a robot this way means that we need to have very precise model when it comes to camera parameters and the physical structure of the target object. The latter should not be too hard to figure out, but the former may not be as easy to identify with hundred percent certainty. Partly, for these reasons, it is common to use more robust control techniques, such as described in section 4.10.

[14] Certainly, if designed correctly, the parameters of a Kalman filter *will* approach the true system parameters as time passes. However, in order for this top happen, the structure of the model must be sufficiently describing for the true system.

Bibliography

[1] Camera calibration toolbox for matlab. Website, 2008. "$http$: $//www.vision.caltech.edu/bouguetj/calib\_doc/$".

[2] Nick Barnes and Zhi-Qiang Liu. *Knowledge-Based Vision-Guided Robots*. Physica-Verlag Heidelberg New York, 2002.

[3] Patrick Bouthemy. Maximum likelihood framework for determining moving edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(5):499–511, May 1989.

[4] Robert Grover Brown and Patrick Y. C. Hwang. *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*. Wiley, pub-WILEY:adr, third edition, 1997.

[5] A.I. Comport. *Robust real-time 3D tracking of rigid and articulated objects for augmented reality and robotics*. PhD thesis, Université de Rennes 1, Mention informatique, September 2005.

[6] A.I. Comport, E. Marchand, M. Pressigout, and F. Chaumette. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *IEEE Trans. on Visualization and Computer Graphics*, 12(4):615–628, July 2006.

[7] P.I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, March 1996.

[8] Daniel F. Dementhon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[9] Olav Egeland and Jan Tommy Gravdahl. *Modelling and Simulation for Automatic Control*. Marine Cybernetics AS, 2003.

[10] B. Espiau. Effect of camera calibration errors on visual sevoing in robotics. *3rd International Symposium on Experimental Robotics, Kyoto, Japan*, October 1993.

[11] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. In C. Laugier, editor, *Workshop on Geometric Reasoning for Perception and Action, LNCS 708*, pages 106–136, Grenoble, France, September 1991.

[12] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[13] T. I. Fossen. *Marine Control Systems*. Marine Cybernetics AS, 2002.

[14] G. D. Hager, W. C. Chang, and A. S. Morse. Robot feedback control based on stereo vision: Towards calibration-free hand-eye coordination. *IEEE Int. Conf. on Robotics and Automation*, 4:2850–2856, May 1994.

[15] J. Hill and W. T. Park. Real-time control of a robot with a mobile camera. *Proc. 9th ISIR, Washington, D.C.*, pages 233–246, March 1979.

[16] N. Hollinghurst and R. Cipolla. Uncalibrated stereo hand eye coordination. *Image and Vision Computing*, 12(3):187–192, 1994.

[17] R. Horaud, F. Dornaika, and B. Espiau. Visually guided object grasping. *IEEE Transactions on Robotics and Automation Magazine*, 14(4):187–192, 1999.

[18] Seth Hutchinson, Greg Hager, and Peter Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12:651–670, 1996.

[19] Bernd Jähne and Horst Haussecker. *Computer Vision and Applications*. Academic Press, 2000.

[20] E. Malis, F. Chaumette, and S. Boudet. 2-1/2-d visual servoing. *IEEE Robotics and Automation Magazine*, 15(2), April 1999.

[21] Ezio Malis. Survey of vision-based robot control. 2002. "$http : //www.societyofrobots.com/robottheory/Survey\_of\_vision - based\_robot\_control.pdf$".

[22] E. Marchand and F. Chaumette. Virtual visual servoing: A framework for real-time augmented reality. *Computer Graphics Forum*, 21(3):289–298, September 2002.

[23] E. Marchand, F. Spindler, and F. Chaumette. Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4):40–52, December 2005.

[24] Éric Marchand, Patrick Bouthemy, and François Chaumette. A 2D-3D model-based approach to real-time visual tracking. *Image Vision Comput*, 19(13):941–955, 2001.

[25] R. M. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[26] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modelling and Control*. John Wiley and Sons, Inc, 2006.

[27] Stefano Soatto Yi Ma, Jana Kosecka, and Shankar S. Sastry. *An Invitation to 3-D Vision*. Springer-Verlag New York, Inc, 2004.

Appendix A Source code

Both the source code written in C++, and the code written in Matlab can be found on the CD. For the source code to work, several external libraries are required. The most important one is the Visual Servoing Platform (ViSP) library. An explanation is given on the CD.