ISSN 0280–5316
ISRN LUTFD2/TFRT--5616--SE

A Robot Playing Scrabble Using Visual Feedback

Johan Bengtsson
Anders Ahlstrand

Department of Automatic Control
Lund Institute of Technology
April 1999

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
April 1999

Document Number
ISRN LUTFD2/TFRT--5616--SE

Author(s)
Johan Bengtsson and Anders Ahlstrand

Supervisors
Anders Robertsson, Bo Bernhardsson, Rolf Johansson

Sponsoring organisation

Title and subtitle
A Robot Playing Scrabble Using Visual Feedback. (En Alfapet-spelande robot).

Abstract

We have used a robot and a camera to design and implement a system which is able to play the known game Scrabble. To make this possible we have constructed a robot system and a vision system. The vision system task is to find the position of the cubes. The robot system uses these positions to generate the trajectories. The feedback from the camera is used to correct the trajectory. That the system is able to play Scrabble is not the interesting, the interesting is that the robot system uses the camera information in real-time to generate and correct the trajectories. Even if there has been done a lot of research to combine a robot system with the sensor information from a camera there still exist few commercial system on the market. In this thesis we will present two different solutions to the problem which correct and generate the trajectories.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
54

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

Preface

This master thesis project is a joint project between the Department of Automatic Control and the Department of Mathematics. The project is divided into two parts: one concerning the robot system and one concerning the image processing part. Therefore have we decided to write two reports. In this report we describe and evaluate the robot system, the vision system is only briefly described. A more detailed description and evaluation of the vision system is presented in the vision report, see [9]

I am very grateful to all the people that have helped me in the project. Klas Nilsson and Magnus Olsson has given me invaluable guidance during the project. The always enthusiastic Anders Robertsson has also supported me with his guidance and presence during the project. Finally, I want to thank my advisor Bo Bernhardsson, and Rolf Johansson for their comments on the report.

Contents

1 Introduction **5**
- 1.1 Earlier projects . . . 5
- 1.2 Existing system . . . 6
- 1.3 Problem formulation . . . 9
  - 1.3.1 Camera calibration . . . 10
  - 1.3.2 Finding three-dimensional coordinates . . . 10
  - 1.3.3 Tracking "points of interest" . . . 10
  - 1.3.4 Identifying object . . . 10
  - 1.3.5 Generation of trajectories . . . 10
  - 1.3.6 Correction of the trajectory . . . 11
  - 1.3.7 Grab the cube . . . 11
- 1.4 Organization of the report . . . 11

2 Methods **12**
- 2.1 Introduction . . . 12
- 2.2 The Robot systems . . . 12
  - 2.2.1 Matlab . . . 12
  - 2.2.2 Envision . . . 15
- 2.3 Vision . . . 17
  - 2.3.1 Camera calibration . . . 20
  - 2.3.2 Determining 3D-coordinates . . . 23
  - 2.3.3 Tracking points . . . 26
  - 2.3.4 Finding objects . . . 28
  - 2.3.5 Identifying objects . . . 31
  - 2.3.6 Algorithm of the test system . . . 31
  - 2.3.7 Algorithm of final system . . . 33
- 2.4 The final system . . . 35

3 Results **38**

4 Discussion **40**

5 Conclusion **41**

A Control of the robot speed using feedback from vision **42**

B Commands implementation **44**

C Envision problems **46**

D The signals in the final system **47**

E Side effects on the robot system **49**

F The Scrabble algorithm **50**

1 Introduction

When we approached the department and said we were interested in doing a master thesis on vision and robot control we were given the following challenge: Can you make a system with a robot and a camera that plays "Scrabble"? For those of you that do not know: Scrabble is a word game where every player is given a number of letters and the goal is to construct long words out of them. For instance the letters could be ABMOORRT which can give the word: " MOTOR", "ROBOT". The letters should be put on small cubic pieces of wood and the robot and camera should be able to first find out where the letters are located and then move the cubes into long words. To do this we need both a robot system and a vision system. The interesting part in the project is not the fact that the robot actual plays Scrabble. It is the fact that we use feed back from the camera to control the robot trajectory. Even the idea not is new, there exist few system that accomplish this fact good.

1.1 Earlier projects

There has been at least two previous studies in Lund with the goal of picking up objects by using a robot with a camera attached to the arm [1] [2]. These projects did not, however, reach the final goal even though big steps where taken in right the direction. The aim of this project is similar but will use another, less theoretical, approach.

A quite big part of Björn Johanssons report [1] was about how to be able to identify the position of certain points, such as corner points, in an image with very high, sub-pixel, precision. The focus of our work has rather been how to use these corner points to reconstruct the three-dimensional scene. This is actually quite close to the focus of the report of Peter Lindström [2]. However, there is a big difference between our solutions. He uses (thanks to sub pixel precision) a mathematical construction called "shape" to make a three-dimensional reconstruction. In our case, we do not use sub pixel resolution and due to that we can not use any shape criteria. Instead we have added a priori knowledge to our system. We know all objects we can encounter (just cubes) beforehand. That allows us to use many more images per time unit to make an average of the estimated three-dimensional positions.

One of the biggest differences in initial conditions between Peter Lindströms and Björn Johanssons project and our project is that we are using feedback not only from the camera, but also from the robot. The feedback from the robot contains the robot positions. That makes it possible for us to determine from where the different images are taken with a quite good accuracy. They had to do the reconstruction only from a set of images. That also gave them a scaling problem: you can not tell if an object is big or if you are close to it without using the relative movement between the images.

1.2 Existing system

We have used the RobotLab of Department Automatic Control Lund, in which two robot systems exist, one ABB Irb-6/2 and one ABB Irb-2000/3. The used control system can in large be described as three modules which are : IgripServer, Trajec and Regul. These modules did only work on the Irb-6 robot system, but effort had been made to get the modules work with the Irb-2000/3 robot system. We have finish these efforts. In other words we are using these modules on both robot systems. The robot systems uses two different coordinate systems, Cartesian and Joint. The Cartesian coordinate system expresses a world position in terms of x, y, and z values. The Joint coordinate system expresses a world position in a joint angle for each joint of the robot.

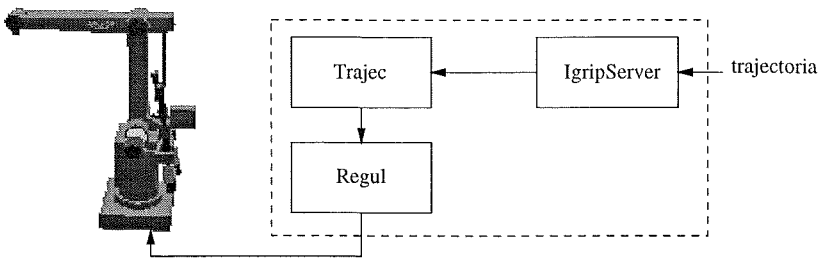

Figure 1: Existing system

IgripServer communicates by a socket to an application,e.g. Matlab, which generates the trajectory. Socket sets up a two ways network communication and usually does not use TCP. This makes it possible to communicate between computers, so the application only have to be run on a computer which is connected to the network. The IgripServer supports precalculated trajectories, which consists of a several via points, and it is prepared to support commands. Every via point include a so called time stamp, the position of every joint, and velocity references to every joint. The time stamp expresses the time the robot has to move to the via point. The IgripServer sends the precalculated trajectory to the Trajec module and which call-back routine Trajec is actual. The later, means which procedure in IgripServer Trajec will run when it call for it's call-back. Later, when Trajec runs the call-back routine in IgripServer, it is possible to perform changes in the via points. The call-back routine is performed for each via point in the trajectory. In the call-back routine IgripServer has access to the Trajec context. The existing context is given by the following table :

context	*description*
ActTime	The time since the motion started
RemainingTime	Remaining time of the motion
IsAtTarget	TRUE if the motion is completed
CoordMode	Active coordmode
CartDesiredOld	The Cartesian coordinates for the former via point
CartNom	The Cartesian coordinates for next via point
JointDesiredOld	The Joint coordinates for the former via point
JointNom	The Joint coordinates for next via point
TimeStep	The time for next via point
NormStep	Normalized value of TimeVia Point
CartStep	The length in Cartesian coordinates for next via point
JointStep	The length in Joint coordinates for next via point
Sensorbased	TRUE sensor is used

Trajec module is able to calculate a trajectory or use a precalculated trajectory. From the via points Trajec calculates velocity and acceleration references for every joints. These are used in the feed forward block of the controllers in Regul. Trajec sends the position, velocity, and acceleration of each joint to the Regul module, which will perform the motion.

Regul module controls the robot and uses cascaded PI controllers for each joint. The velocity and acceleration are feed forwarded.

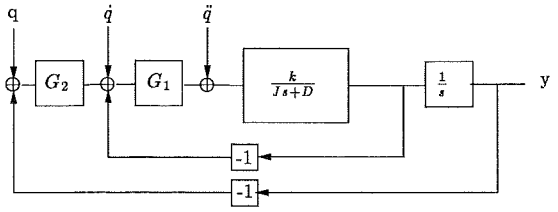

Figure 2: q is the position reference, $\dot{q}$ is the velocity reference, and $\ddot{q}$ is the acceleration reference.

The two robot systems are supported by computers operating in two software layers. On top is a SUN workstation on which the development and compilation takes place. Below is the target computer which is a VME system with M68040 as main processor. This real-time system runs the robot controllers, reads the sensors, etc. The existing control system is in greater part implemented using Modula-2 and a real-time kernel developed at the department. For more information about the existing system see [3] and [4].

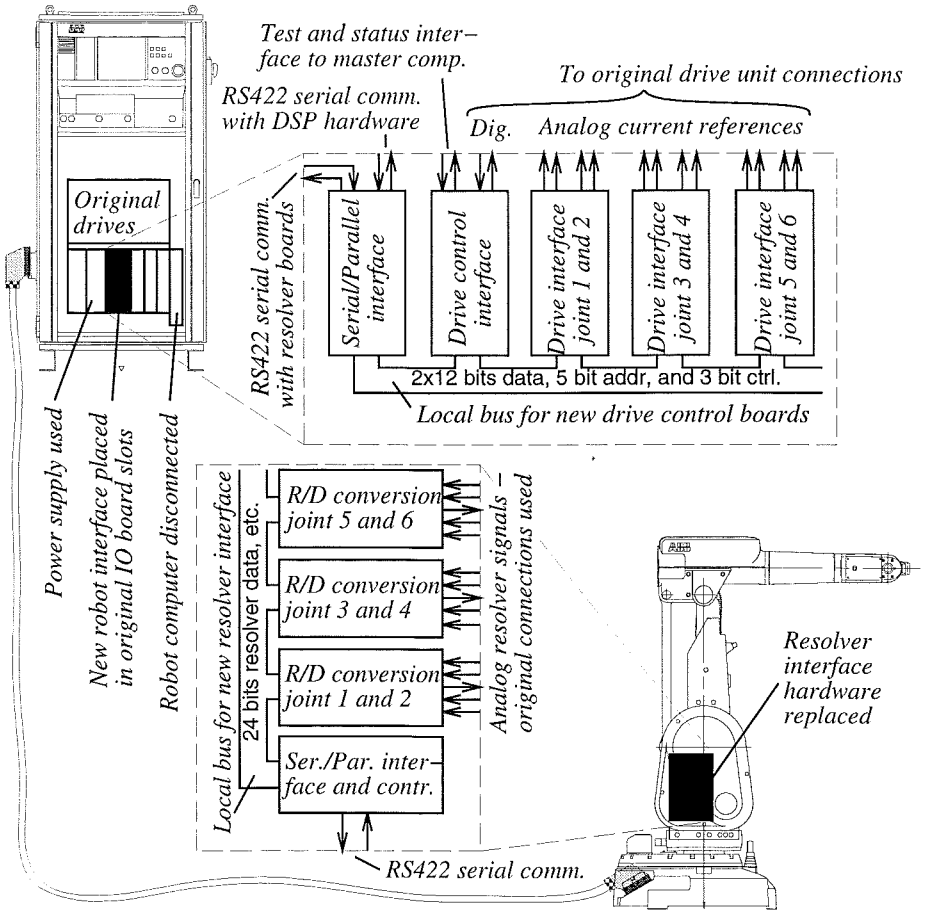

Figure 3: The hardware of the Irb-2000 system.

The camera we use is an analog SUN camera, and it uses composite video. This is the format a common video recorder uses. The camera is connected to a Silicon Graphics workstation, O2, on which the image processing is performed. We have also used the commercial program ***Envision***, which is a 3-dimensional CAD program with a graphical simulation environment. It contains complete robot models, which are able to simulate the robots. Envision uses a special programming language called Graphic Simulation Language, GSL. Every device in the work cell can be accessed and controlled from a GSL program. Envision also provides an opportunity to manipulate the devices in a work cell, e.g. change the devices position from outside Envision, by the Low Level Telerobotics Interface, LLTI. LLTI specifies an interface to the world outside Envision. This make it possible to e.g. get a device's position in a work cell. For more information on Envision see [5].

A work cell in Envision could look like :

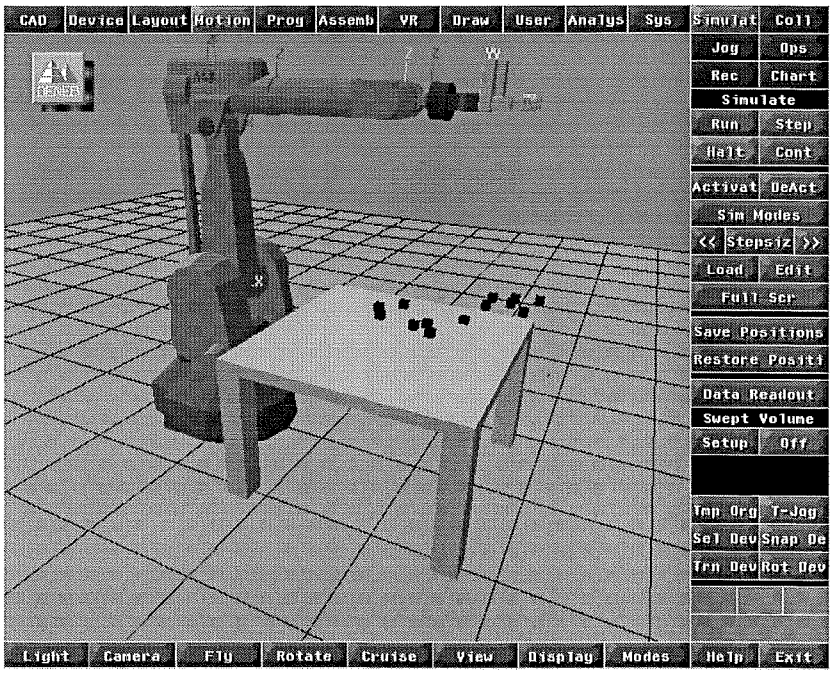

Figure 4: Envision

The cubes we use to play scrabble with look like :

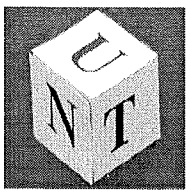

Figure 5: cube

1.3 Problem formulation

Our main goal is to construct a system that is able to get a robot using information from a camera, to play scrabble. This means that the robot should be able to move the cubes and construct long words. A camera is attached to the robot

gripper. In contrast to ordinary scrabble where you play with makers with one letter on it, we use cubes. Each cube has 6 letters. The a priori knowledge the system has is the number of cubes, that every object on the board is a cube and the height of the table on which the board is put. The scrabble algorithm uses a simple scoring system where each letter in a word will give one point and the lengths of the words are limited to 7 letters. Except for this modification all the common rules in scrabble is used. The main problem can be divided into the following subproblems.

1.3.1 Camera calibration

In the task of finding the objects the vision system accesses the data from the camera and the coordinates and orientation of the hand of the robot. However some parameters have to be estimated so that we can get an interpretation of the video data. To achieve that we have to calibrate the camera, i.e., determine its position and orientation in relation to the robot hand as well as its intrinsic parameters.

1.3.2 Finding three-dimensional coordinates

The vision system tries to make a three dimensional reconstruction using a series of images, projections, of the scene. Since the calculations are based on the movements of the projection of physical objects the vision system has to search for "points of interest" , physical points that can be easily distinguished in the different images. They can for example be corners of cubes. Those projected points are used together to find the original physical point.

1.3.3 Tracking "points of interest"

When using an image sequence with a multitude of "points of interest" we have to decide what projection that belongs to which physical point, i.e. the link "points of interest" in different images together. This can be done by using epipolar planes or reprojection of estimated physical points.

1.3.4 Identifying object

Even if features have been linked together into three-dimensional points and those points have been put together into objects, it is not always enough. Sometimes are we interested in telling two similar objects apart and we have to use other criteria as color or structure of the objects. In this case we must be able not only to find the cubes but also be able to read the letters.

1.3.5 Generation of trajectories

We want the robot to follow a particular trajectory, e.g. move to a cube. The position of the cube is given from the vision system. In order to make the robot perform a desired motion there will have to be some generation of the so

called "via points" in the trajectory. The via points can either be generated in real-time or be precalculated.

1.3.6 Correction of the trajectory

There are uncertainties in the positions from the vision system. The vision system gives a position of the cube and from it a tentative trajectory is generated. This needs to be corrected due to the feedback from the vision system to get a better accurate position of the cubes when we move the camera closer to the cube.

1.3.7 Grab the cube

In the existing robot system it is not possible to open or close the gripper. To be able to move the cubes and play scrabble the robot must be able to grab and drop the cubes.

If these subproblems are solved, we have the blocks needed to put together a system which is able to perform the different phases in the Scrabble. The Scrabble can be divided into a few phases, e.g. draw phase : The robot take cubes from the cube pool, put it on the hand and the most important construct word from the given letters.

1.4 Organization of the report

The structure of the report is following: In chapter 2 we presents our solution to the problems. In chapter 3 the result of our solutions is presented. In chapter 4 we discuss possible improvements of the system and the conclusions is presented in chapter 5.

2 Methods

2.1 Introduction

It is appropriate to divide the system into two parts, one controlling the robot, robot system and one handling the image processing, vision system. We have made two different implementations. We started with one using the Irb-6 robot system and a vision system implemented in Matlab, and later changed to one using the Irb-2000 robot system a vision system in C, implemented on the SGI platform. The reason why we change robot system is two : Firstly, the Irb-2000/3 robot has better precision, due to better mechanic and control system. This make it possible to use higher speed and still have good precision of the robot motions. Secondly, we want to have a robot with six degrees of freedom. Hence, six degrees of freedom is necessary to get the robot able to grab a cube in a general position. We have also implemented two different methods of generation and correction of the trajectory, one using Matlab and one using Envision. Envision is made for robots with six degrees of freedom. Hence, we generate the trajectories in Matlab when the Irb-6 robot system is used.

2.2 The Robot systems

2.2.1 Matlab

First we have to decide which type of trajectory generation we want to use, real-time generated trajectory or precalculated trajectories. Real-time generation of the trajectory means we do not know the end of the trajectory at the actual time, we know only a short distance ahead. We use the feedback from the camera to generate the trajectory and to correct the real-time generated trajectory. Using a precomputed trajectory means that we let the vision system calculate a position of the cube. Then we generate the complete trajectory, from start position to end position and only use the feedback from vision to correct the trajectory. A robot system has very high real-time requirements whereas Matlab is very slow and for that reason real-time generation of the trajectory is rejected. Due to this it is appropriate to chose a method that precalculates the trajectory. To solve the problem, new functions have to be added to the already existing robot-system and an interface to the vision-system has to be designed.

Trajectory generation The first subproblem to solve is the trajectory generation. To be able to generate a tentative trajectory, the Matlab program needs to know the start point and the an end point of the motion. Since this information comes from the vision system the Matlab program needs to communicate directly or indirectly with it. Moreover, if only a start point and end point is used, the path is not specified. Is the path important? Yes, the path and the information the vision receives is correlated. Since the system we desire should give us the best determination of the position of the cubes, the tentative trajectory should be generated from the image processing point of view to get good

precision in the cube positions. A path which moves straight to the end point will not provide the image processing much information, since the information change between the images is small. But we want the path to be relatively straight in order to get to the end position in a acceptable amount of time.

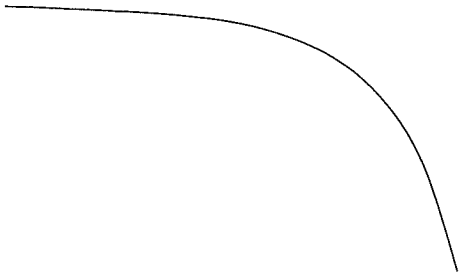

Figure 6: The path

The path we used is seen in figure 6 which is compromise between the needed information and the time to perform the trajectory. To generate a trajectory which looks like the one in Figure 6 we used the *spline* function in Matlab. This trajectory is expressed in Cartesian coordinates and the robot uses joint coordinates, the trajectory has to be converted to Joint coordinates before it is transmitted to Trajec. We also supply every via point in the trajectory with a time stamp.

Correction of the trajectories The correction of the trajectories could be done in the Matlab program or in IgripServer. It is not suitable to place it in Matlab, because of the high requirement of real-time performance. Hence, the correction must be done in IgripServer. By the feedback from vision, the IgripServer gets an updated position of the cube.

One solution is to use the differences $\triangle x$ between the cubes current position estimate and its former position estimate. Since it is possible to change the reference value for the next via step, $y(k)$, in the IgripServer call-back routine it is possible to add the difference $\triangle x$ to the reference values $y(k)$. If the difference is large and it is added without restriction you will get unwanted jumps of the robot. If the difference is very large, the robot will even stop. Due to this fact there has to be some limitation of the difference $\triangle x$ before adding it to the reference. The use of the limited difference $\triangle x_{limit}$ is following: In the IgripServer call-back routine we check if the trajectory need to be corrected, if so the limited difference $\triangle x_{limit}$ is added to $y(k)$ and to all following $y(k)$. The call-back routine runs for each $y(k)$ perform the correction until we have change the end position of the trajectory so it corresponds to the new cube position. The trajectory needs to be corrected if the position of the cube is changed or if the trajectory not have been completely corrected. What should the limit of the difference $\triangle x_{limit}$ be ? We want the correction to be fast but the robot should

not jump, therefore we have to compromise. The limit we chose expressed in Cartesian coordinates was 1 mm, so much do we change $y(k)$ in x- ,y- ,z-values in the call-back routine.

Grab the cube The IgripServer needs to support the commands Grasp, Drop if we should be able to get the robot to grab and pick the cubes . We will first describe the methods for the commands Grasp and Drop. The Irb-6 robot system already has the I/O-module Gripper which we use to control the gripper. The IgripServer is prepared to support commands, it sort out the commands from the trajectory. All the commands have a negative number instead of a time stamp which is used to sort out the commands. The number we add is:

command	*number*
Grasp	- 5
Drop	- 6

The Grasp and the Drop commands look like [command type, tool, force, grasptime, 0, 0, 0, 0, 0, 0, 0, 0, 0] for the Irb-2000/3. The Irb-6 robot has one joint fewer and the grasp command look like [command type, tool, force, grasptime, 0, 0, 0, 0, 0, 0, 0]. The tool value determines which tool should perform the command. The force value determines the force which is used to open or close the tool. The Irb-6 robot can not vary the force. The grasptime determines when the Grasp/Drop command will perform. We connect the Grasp/Drop command to a via point in the trajectory by using the grasptime. This makes it easy to know in the call-back procedure when to perform the Grasp/Drop. In order to know when the Grasp/Drop commands is to be performed in the call-back procedure the context of Trajec has to include the commands. For that reason we expand the context to include Client Context. The Client context is an address reference to the Grasp/Drop commands. The IgripServer expands to also contain a process which manages the performance of the opening or closing the tool. solution is almost correct, but will make the Grasp/Drop command perform one step earlier than wanted, due to the structure of the Trajec and IgripServer. This problem is solved by making the context also contain the time stamp of the following via point. This element is referred as GraspDelay.

The system Due to previous conclusions, Matlab needs to know the cube positions to generate the tentative trajectory. The existing interface between IgripServer and Matlab has been extended to also contain the cubes position. The data which is sent between the IgripServer and Vision is the following. Vision sends the position of the cubes to IgripServer and IgripServer sends back the robots position. Vision needs the robot position to be able to calculate the cube positions from the information the camera provides. Moreover, if vision needs long time to process an image, we want the robot to slow down. This problem is solved in appendix A.

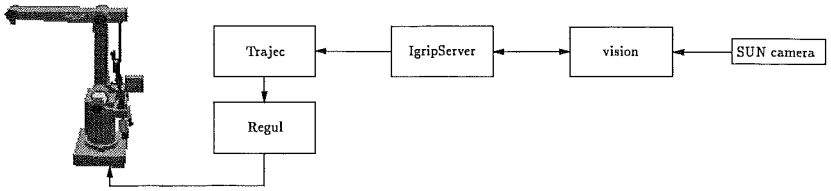

Figure 7: The Matlab system

This system is able to generate trajectories and correct them and also grab the cube.

2.2.2 Envision

Envision offers the opportunity to generate the trajectories in real-time. Moreover, the correction of the trajectories can also be done real-time in Envision.

Trajectory generation The problem to generate trajectories has already been solved in a joint project between the Departments Automatic Control and the Division of Production and Materials Engineering. A summary of the method in Envision is: By specifying "tag" points in the work cell, it is possible to control the movement of the robot. Tag points store position, orientation and configuration information. A tag point is represented as a coordinate system with the labels x, y, and z. It is possible to order a robot to move to a tag point, if reachable. This mean that the joint values of the selected robot device so the grippers center point coordinate system reaches the identical position and orientation as the tag point. Then by using LLTI, see [5], a C-program read the robots position and generate the via points in the trajectory, which is transmitted to IgripServer. By using tag points it is possible to decide how the path will look like. We place the tag points so the path look likes the one in Figure 6.

Correction of trajectories One advantage of using Envision is the easy way to generate the trajectory with the tag points. This is not the only quality the tag points have, every tag point is connected to a device. That means that if the device is translated the tag point is translated. This can be used to correct the trajectory. We only have to use the feedback information from the camera to translate the cube and the correction will be performed. There is one drawback with the tag point. When the simulated robot have started to move to the tag point. It will move to the position the tag point had when the motion started, it does not care if it has been translated during the move. Therefore it has to be close between the tag points. This is not any large problem, it is only time consuming to create many tag points. We use 16 tag point when picking a cube. To use this method we supply every cube device with own tag points. To

be able to use the suggested solution, we have to be able to change positions of different devices. This can be done by using the LLTI. We connect every cube to the LLTI. This result in that all the cubes have the same routines, interface and there is nothing that separates them from each other. Since we are going to use many cubes, every cube needs to have some identification, to make it unique. To every device in Envision is it possible to add kinematic. A robot have kinematic, but a table have no kinematic. In Envision the kinematic of a device can be changed and since the cubes do not have any kinematic, a kinematic parameter could be used as identification. One suggestion is to use the length value, which we use. Then it is possible to check a device's identification via LLTI, by check the device's kinematic. We are now able to move a specified cube to a new position, the position is received from the vision system.

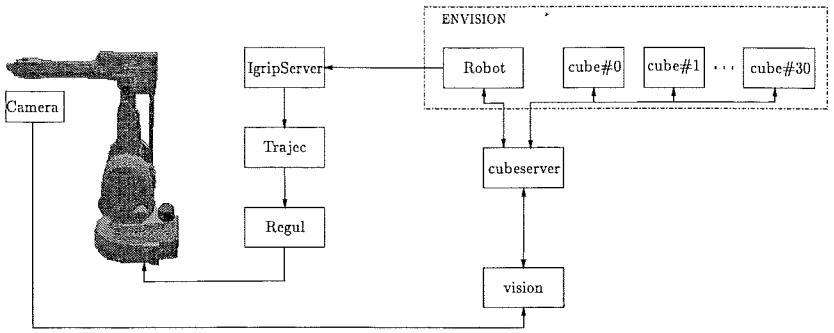

Figure 8: Envision system

The system The cubeserver is a C-program which uses the LLTI to manipulate the workcell, e.g. change the position of a cube. The cubeserver keeps track of the cubes and uses the sensor information from the vision system in order to update the cube positions in the workcell in Envision. The Irb-2000/3 robot does not as the Irb-6 robot has a "home" position, a position which the robot automatically moves when the robot is started. This is a problem, because the start position of the real robot and the robot in Envision must be the same. Trajec is capable to generate a trajectory from the actual position of the robot to a specified position which is perform during a specified amount of time. If we send a position and a time value t Trajec will generate a trajectory which move the robot to the position in the time t. Hence, we only need to make the IgripServer support commands that use this function in Trajec. We call the commands MoveL and MoveJ. As mentioned before, all the commands have a negative number instead of a time stamp and the number we used is:

command	*number*
MoveL	- 7
MoveJ	- 8

The MoveL and MoveJ look like [command type, joint1, joint2, joint3, joint4, joint5, joint6, motion time, 0, 0, 0, 0, 0] for the Irb-2000/3. The Irb-6 robot have one joint lesser and the grasp command look like [command type, joint1, joint2, joint3, joint4, joint5, motion time, 0, 0, 0, 0]. Joint1 to joint6 is the end position value of the joints. Motion time is the time during the robot have to perform the motion. We will not allow to mix the command MoveL and MoveJ with the trajectory, there can not be any MoveJ or MoveL between the via points in a trajectory. This is the only restriction in the use of the commands.

2.3 Vision

The ideal object to study when trying to make a three-dimensional reconstruction is covered by a mesh or some kind of numbered points. That makes it possible to see the objects movement relative the camera. Otherwise, if we for example move the camera radially around a sphere we will not notice any motion at all.

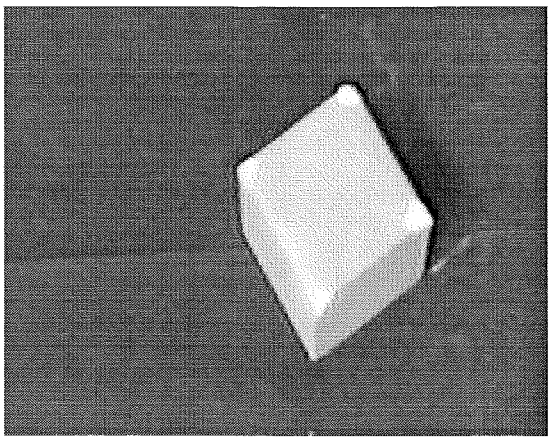

Figure 9: An image of a sample cube

As the reports of Björn [1] and Peter [2] among others have treated the problem of how to find features we will only show an example. Given a problem as this, where we are searching for cubes it might be a good idea choosing the corners as features. The image in Figure 9 is a magnified part of an original

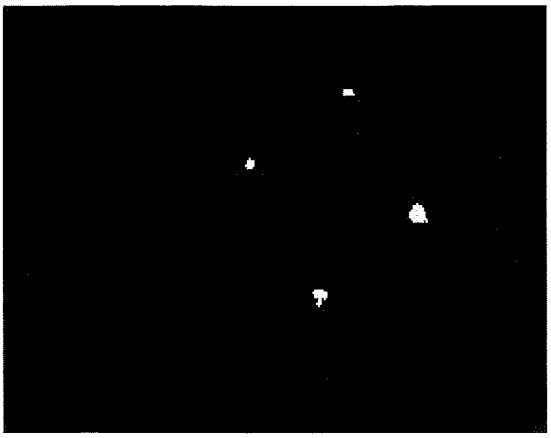

Figure 10: Example of points of interest if they are chosen as the corners of the cube

image of one of our cubes[1]. If we choose to use the corner points as features we get a new image as in Figure 10 after corner detection. To make the implementation easier for us, as this work already has been done,we have marked the corner with white dots so that they can be easily detected by thresholding the original image. What we actually are interested in is the coordinates of those points, so we scan through the image at this stage, discard the image and keep the coordinates. The theory described here is general, it does not matter how we find the points, any points will do as long as they are a fix three-dimensional points.

To be able to use images to reproduce a three-dimensional scene we must first investigate how an object is mapped onto the screen. We will here use a homogeneous (a base coordinate system) coordinate system *hom*, the coordinate system of the robot *rob*, the coordinate system of the camera *cam* and the coordinate system of the screen *scr*.

The transformation is divided into to steps: an euklidian transform and a projection. The affine transform corresponds to the motion of the camera in the homogeneous coordinate system and consists of a rotation R ($R \in M_{3x3}$) and a translation b.

$$\begin{bmatrix} x_{rob} \\ y_{rob} \\ z_{rob} \end{bmatrix} = R \left(\begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \right) \quad (1)$$

To write equation 1 more practically the use of homogeneous-vectors are

[1]It is also rotated 180 degrees, as all the other images from the camera as it is mounted upside down on the robot hand

introduced. The rotation and translation are put together as on single operation by adding an extra element, always set to one, at the end of the vectors.

$$\begin{bmatrix} x_{rob} \\ y_{rob} \\ z_{rob} \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \end{bmatrix} \tag{2}$$

where $t = R*b$. The whole transformation from the homogeneous coordinate-system to the camera coordinate-system can be written.

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} = \begin{bmatrix} R_{cam} & t_{cam} \end{bmatrix} \begin{bmatrix} R_{rob} & t_{rob} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \end{bmatrix} \tag{3}$$

where R_{cam} and t_{cam} corresponds to the rotation and translation in respect to the robot coordinate-system.

The second part of the transform is the projection from the coordinate-system

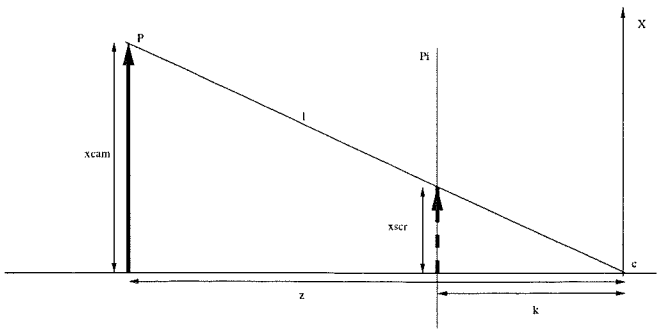

Figure 11: Projection of a point on the image plane

of the camera to that of the image. This can be made in several ways. We are here using a camera-model similar to the one found in [7].

Imagine having your eyes at the point **c**, the focal point, in Figure 11. You will then know that the point **P** is situated somewhere along the line **l**. In other words, a projection is not invertible. So if the point P is projected onto the plane $\mathbf{\Pi}$ you will not see any difference. The projection can be written (for one out of the two dimensions of the image plane), using congruent triangles.

$$\frac{x_{scr}}{f} = \frac{x_{cam}}{z} \tag{4}$$

As **f** is a camera constant the whole system can be written:

$$\begin{bmatrix} x_{scr} \\ y_{scr} \\ 1 \end{bmatrix} = \frac{1}{z_{cam}} \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} \tag{5}$$

To get rid of the fractions, z_{cam} is normally replaced by λ. The diagonal matrix **K** describes the intrinsic properties of the camera

$$\lambda \begin{bmatrix} x_{scr} \\ y_{scr} \\ 1 \end{bmatrix} = K \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} \tag{6}$$

This is true only for an ideal camera. For a normal, non-ideal camera another matrix **K** is used where four more parameters have been added.

$$K = \begin{bmatrix} f & sf & x_0 \\ 0 & \gamma f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

f works not surprisingly as a magnification factor. γ is the ratio between the size of the x and y pixels on the CCD. It is ideally 1. $(\mathbf{x_0}, \mathbf{y_0})$ is the principal point, that is the coordinate of the orthonormal projection of focal point onto the image plane. Finally, s is the skew, that models the angles in CCD-elements of the camera.

Combining equation (3) and (6) gives us the equation for the whole system.

$$\lambda \begin{bmatrix} x_{scr} \\ y_{scr} \\ 1 \end{bmatrix} = \underbrace{\underbrace{K \begin{bmatrix} R_{cam} & t_{cam} \end{bmatrix}}_{P_{cam}} \underbrace{\begin{bmatrix} R_{rob} & t_{rob} \\ 0 & 1 \end{bmatrix}}_{P_{rob}}}_{P} \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \end{bmatrix} \tag{8}$$

In equation (8) have we have here distinguished between the different contributions to **P** which will be determined in separate ways. The P_{rob} matrix can be extracted from the robot system whereas the P_{cam} has to be measured by hand or calculated out of images taken by the camera. The next section will describe the determination of P_{cam}.

2.3.1 Camera calibration

In camera calibration we distinguish between intrinsic and extrinsic calibration. Intrinsic calibration is to determine the parameters in the **K** matrix, i.e. the properties of the camera. Extrinsic calibration, or hand-eye calibration, is about finding position and orientation of the camera in relation to a known

point. These two problems can either be solved individually or simultaneously.

Normally when calibrating, the most straightforward method of solving the problem is to solve the linear system found in equation (8). Suppose that the point of origin, $\mathbf{P_{rob}}$, is known. The matrix $\mathbf{P} \in \mathbf{M_{3\times 4}}$ holds 12 unknown and an additional unknown is added for λ. As $\mathbf{P}$ depends only on $\mathbf{K}$, the position, and the orientation, this will remain the same for any number of added points. Six points would ideally be enough. Each point gives 3 equations. Using 6 points we will have 18 unknowns and 18 equations. The trick for solving this problem is rewriting equation (8) through

$$P \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \end{bmatrix} - \lambda \begin{bmatrix} x_{scr} \\ y_{scr} \\ 1 \end{bmatrix} = 0 \tag{9}$$

to the structure

$$\underbrace{\begin{bmatrix} x_{cam}(1) & y_{cam}(1) & z_{cam}(1) & 1 & 0 & \dots & x_{scr}(1) & 0 & \dots \\ 0 & 0 & 0 & 0 & x_{cam}(2) & \dots & y_{scr}(1) & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & x_{scr}(2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{bmatrix}}_{A} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ \vdots \\ p_{44} \\ \lambda_1 \\ \lambda_2 \\ \vdots \end{bmatrix} = 0 \tag{10}$$

and calculate the null space with respect to the elements in P [6]. Actually it is possible to do a parameterization of $\mathbf{P}$ using only 3(# unknown in a rotation matrix)+3(# unknown in the translation) +5(# unknown in $\mathbf{K}$)=11 unknown. However that does not change anything while solving the problem as a system of linear equations. All 12 unknown has to be found anyway as the decomposition is made after having determined $\mathbf{P}$.

The problem is that there are measurement errors. To minimize these we can add even more points. Now an other problem emerges: the fact that we have null space is because that the eigenvectors of $A \in M_{m\times n}$ does not extend $\mathbb{R}^n$. If we add more points that will do no harm if the new rows of A are linear dependent of the old. However, because of the measurement errors they will not. So, after adding the seventh point A will no longer have a null space.

Ideally, $\mathbf{A}$ has a null space. To find it we can make a singular value (SVD) decomposition of $\mathbf{A}$ into

$$A = U\Sigma V^H \tag{11}$$

Because of the trouble of measuring the tree-dimensional coordinates we used a fixed single point and instead moved the camera. That is possible because of

that the robot gives us the orientation and position of its coordinate system. Equation (8) can the be rewritten as:

$$\lambda \begin{bmatrix} x_{scr} \\ y_{scr} \\ 1 \end{bmatrix} = \underbrace{K \begin{bmatrix} R_{cam} & t_{cam} \end{bmatrix}}_{P_1} \begin{bmatrix} x_{rob} \\ y_{rob} \\ z_{rob} \\ 1 \end{bmatrix} \tag{12}$$

P from equation (8) will alternate when moving the robot whereas P_1 from equation (12) will remain constant for all points. After having found P_1 it may be factorized into $K\,[R\ t]$. This algorithm is made to minimize the mean square error of P but what happens to **K**, **R**, and **t**? We have as seen a problem with that our initial conditions are not met. Another approach to solve the problem is to do a parameterization in terms of the rotation angles α, β and γ and the positions, t_x, t_y and t_z, of the camera on the robot arm. Altogether we denote them Δx. We still do the calibration by using a known and fixed point.

If we try minimizing the error function

$$f = \sum_{i=1}^{N} \left\| \begin{bmatrix} \lambda_i x_{scr}(i) \\ \lambda_i y_{scr}(i) \\ \lambda_i \end{bmatrix} - K \begin{bmatrix} R_{cam}(\Delta x) & t_{cam}(\Delta x) \end{bmatrix} \begin{bmatrix} x_{rob}(i) \\ y_{rob}(i) \\ z_{rob}(i) \\ 1 \end{bmatrix} \right\|_2 \tag{13}$$

where N is the number of points used for the calibration. These are used to construct R_{cam} and t_{cam}. This is a problem that ought to be solvable with any standard-method, like Steepest-descent or Newton-Raphson. This works however only for variations in t, for variations in the R matrix the methods get unstable.

A method actually do work is found in [8], using the Gauss-Newton method. This method uses a vector Y that is the residual of the measured coordinates and coordinates calculated with Δx.

$$Y = \begin{bmatrix} \lambda_1 x_{scr}(1) \\ \lambda_1 y_{scr}(1) \\ \lambda_1 \\ \lambda_2 x_{scr}(2) \\ \vdots \end{bmatrix} - \begin{bmatrix} \begin{bmatrix} R_{cam}\delta x & t_{cam} \end{bmatrix} \begin{bmatrix} x_{rob}(1) \\ y_{rob}(1) \\ z_{rob}(1) \\ 1 \end{bmatrix} \\ \begin{bmatrix} R_{cam}\delta x & t_{cam} \end{bmatrix} \begin{bmatrix} x_{rob}(2) \\ y_{rob}(2) \\ z_{rob}(2) \\ 1 \end{bmatrix} \\ \vdots \end{bmatrix} \tag{14}$$

It minimizes the sum of the squared residual, $f = Y^T Y$ by doing a linearization of $Y(\Delta x)$

$$Y(\Delta x) = Y(0) + \frac{\delta Y}{\delta \Delta x} \Delta x \tag{15}$$

As we are searching for $Y(\Delta x) = 0$, equation 15 can be rewritten as
Här kan resten av metoden beskrivas

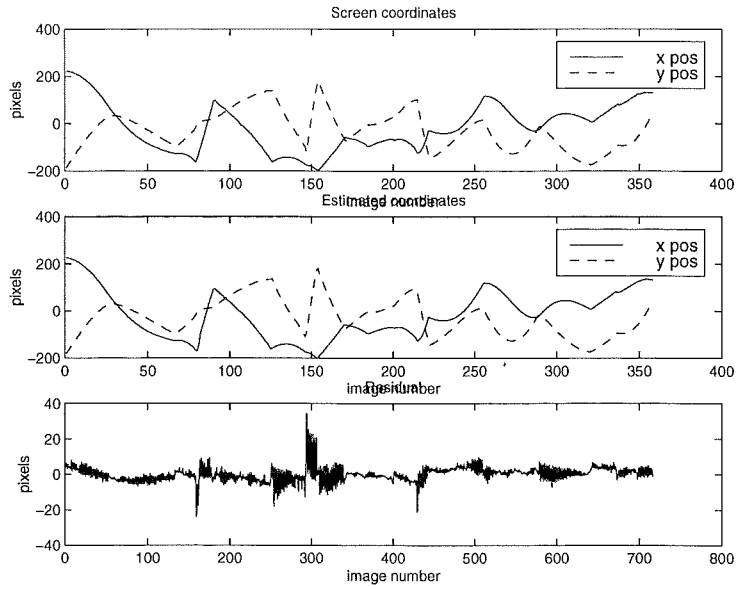

Figure 12: *Data from camera calibration off the Irb-2000. The first figure shows the the x, y, and z coordinates of an object registered with the camera. The z value is implicitly 1, (as seen in equation (8)).*

The second figure shows a superposition of the registered coordinates and the calculated coordinates calculated.

The third figure shows the difference between the calculated and measured x and y coordinates, that is why it here exists the double amount of points compared with the figures above.

2.3.2 Determining 3D-coordinates

A projection is, as we have mentioned before, a non invertible transform. By knowing the transformation,i.e. the intrinsic calibration of the camera, K, R_{camera}, t_{camera}, the extrinsic calibration R_{robot}, t_{robot} and the coordinates of a projected point it is possible to determinate everything but the distance to the it. To be able to find the three dimensional coordinates we can use triangulization (see Figure 13). That is, if we have to different images, taken from different positions, showing the same real point we can localize the three-dimensional coordinates by finding the interception point of the two lines.

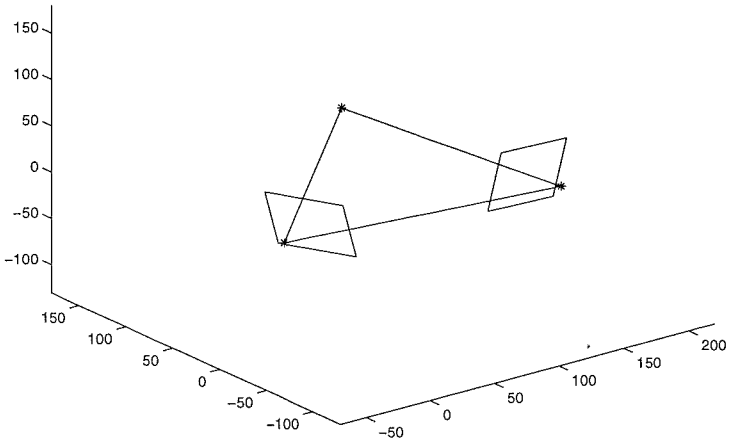

Figure 13: Two images of the same physical point taken from two different points

In practice this seldom occurs. Because of our limited camera resolution the lines will not coincide. However, that does not pose us a big problem, we simply have to settle with least-squares solution.
There are (at least)two different ways to find a solution to the problem stated above. Either through direct geometrical calculations that are quite messy because of all the coordinate systems involved. At last we end up with the two equations of the lines making it possible to determine λ_1 and λ_2. One of those values are then put back into the original equations to get the coordinates. The other solution is simply to rewritten the equation (8) through equation (16)

$$\begin{bmatrix} & -x_{scr} \\ P & -y_{scr} \\ & -1 \end{bmatrix} \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

The second solution is both easier to implement and more general than the first one. It can be generalized as equation 17 , which make it possible to use an arbitrary number of points, thus giving a higher resolution.

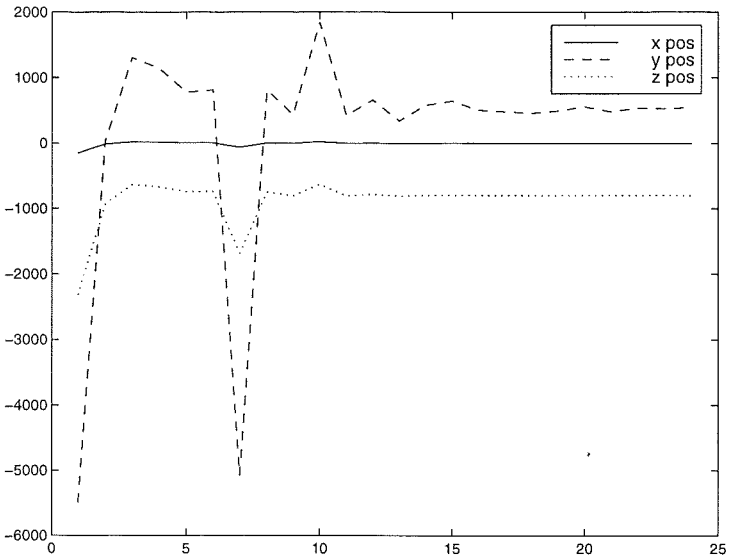

Figure 14: The estimated x, y , and z coordinates of a feature, note the correlation between them.

$$\begin{bmatrix} & -x_{scr}(1) & 0 & \dots \\ P_1P_2(1) & -y_{scr}(1) & 0 & \dots \\ & -1 & 0 & \dots \\ & 0 & -x_{scr}(2) & \dots \\ P_1P_2(2) & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} x_{hom} \\ y_{hom} \\ z_{hom} \\ 1 \\ \lambda_1 \\ \lambda_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{17}$$

The problem above does not always have a solution. If we are unlucky the motion of the camera has been the same as the direction to the point. That makes the two lines l_1 and l_2 identical and we will get a parametric solution to the problem. Even though that scenario is not that probable we get into other strongly related problems. If the difference between the lines are small and the resolution is low the distance from the camera to the intersection point is quite random. In practice it means that we loose primary resolution along z_{cam}. As the camera coordinate system is rotated in respect with the homogeneous this will be visible in all components in the homogeneous coordinate system as seen in Figure 14.

By using the second method and adding more points the big variance can be lowered and the penalty of unfortunate movements are more or less eliminated. If we have an over determined system of equations and add a new set of equations, no harm is done. In Figure 22 we have estimated the position of the same real point, using 4,8,12 and 16 points. As seen in the picture, the

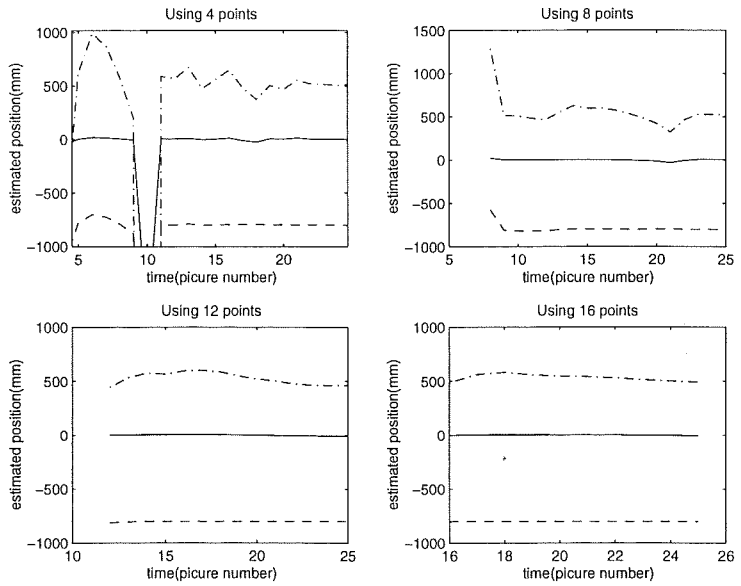

Figure 15: Estimated three-dimensional coordinates using a different number of simultaneous images

big peaks in the estimation of the position of the point occurs simultaneously for all of the components. The reason is that we look at the position in the homogeneous coordinate system when hardest part in determining the position is finding the distance. As x, y and z all are linear dependent of the distance they all change.

2.3.3 Tracking points

The problem of determining one or many three-dimensional points simultaneously is of course almost analog. In fact the only thing that makes it harder finding many points is that we can not easily tell which points in different images that corresponds to same physical point. To be able to do that there exists a multitude of various techniques. They are normally based on the principle of gathering information about the physical point and later on use it to make a guess of where the point ought to be. The calculated position is then compared with the points actually found and the error in the prediction is fed back into the prediction of the next point.

In the earlier projects, where they were only aware of the order of the images in the image stream they assumed that the images were taken in a reasonable high rate compared to the motion of the camera. So that the positions of a certain physical point that appears in multiple images would be correlated. By using

FIR-filtration they could use the old coordinates to predict the coordinates of the new point.

As mentioned before, we can not rely too heavily on our earliest prediction of a points position. However, as we know the position of the robot hand, and thanks to the calibration the position and properties of the camera we can make an artificial projection onto the screen our predicted point. In that task our former problem is helping us. By the same reasons as a small movement of the camera gave us a poor resolution of the distance to the point a big difference in distance will not ruin the projection of our estimation.

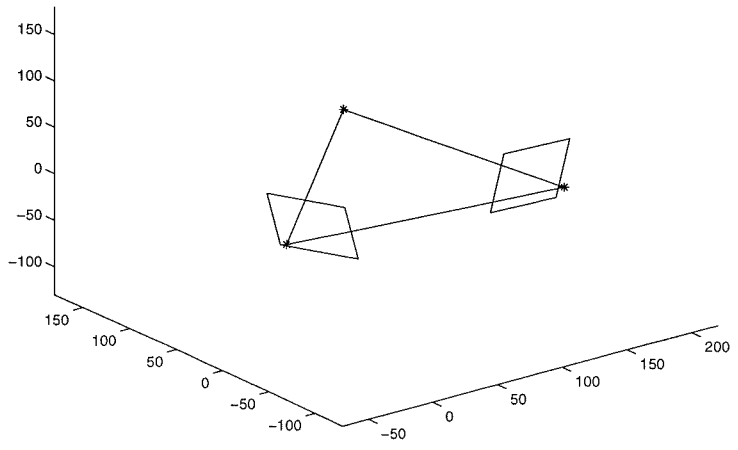

Figure 16: This figure shows the epipolar plane created between the object point and the two focal points

This approach is however only possible if we know already an approximate position of the point. To find the second point we use something called epipolar geometry. As seen earlier, for example in Figure 11 a point P projected onto $x_s cr$ is known to be situated somewhere along the line l. If two images are taken from different positions as in Figure 16 we will get a plane through the points P, $c1$ and $c2$ who's intersection with Π_2 is the same as the projection of l onto Π_2. I.e. we know that the point seen in the last image will be situated somewhere along the line. The exact position is depending on the distance to

the point.

2.3.4 Finding objects

When analyzing an image you have to be quite certain of what you are looking for and looking at. Unlike us humans the image processing have no a priori knowledge of the world. All it will know is what we tell it and because of that it will of course have trouble adapting to shifting conditions. Normally the conditions have to be tailored for the vision system and the program carefully trimmed to achieve it's goal.

Luckily much of the theory presented earlier is general. What we need to observe is just the coordinates of the projection of fix three dimensional points onto the screen. Those points can be found in any way as long as they satisfy this condition. A natural choice of features to use is the corner points of the cubes. However, a series of problems then arise. Due to lack of precision in the estimations of where to find a point there is a significant risk that an error will occur and the wrong point will be picked. That will make later predictions harder and of course makes the estimations even worse. Another problem is to get the information of where to find the cubes out of the information of where to find the corners. This rather trivial task can get bottle-neck in a time critical system. If we for example have 20 cubes they have 160 corners all together of which a maximum of 140 can be seen at one time. Matching them together in a most naive manner is an operation of $O(n!)$. This can of course be solved much more efficiently by using more complex data structures but it still gives a hint that it is a good idea to reduce the number of points being processed. We can instead look at the hole cube as a unit. The images taken (Figure 17) is simply thresholded at a suitable level, which gives an output as in Figure 18 or in Figure 19 depending on which features that is demanded to detect. This level has to be adjusted to the scene, lumination and objects.

A good property of the cube is that it is symmetrical in all three dimensions. That means that the point you get by making a projection of cube and calculating the center of gravity is almost the same as the direct projection of the center of gravity. The reason for that is a discrepancy is because of the distortion due to the perspective transform. By instead using those points we get a direct measurement of the position of the cubes.

However, this suffices only for retrieving three out of the cubes six degrees of freedom. To get the orientation we have to reanalyze the picture searching for the corners. As we already know that this can be done according to the results from previous projects [1] [2] by using corner detection kernels or the hough transform, we have not invested any time implementing it. Instead we have chosen to marking the corners in a bright color so that they can be easily found easily in the same way as we found the cubes by simply thresholding the

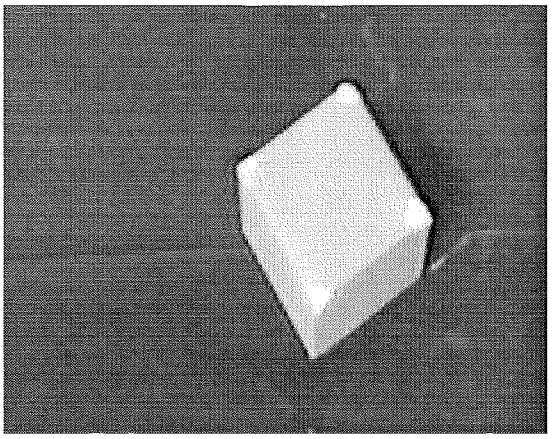

Figure 17: a sample image

image (picture 19).

Above, we talked about the problems we would get into by using the corner points finding the cubes. What makes it better using the thresholded image of the whole cube to find the cube and the thresholded image of the corners to find its rotation? The answer is simply that the orientation of the cubes is not as important. We can afford waiting to determine the rotation until we are close enough and interested of picking up a cube. I.e. we will not see too many corner points at the time.

What we are searching for now is simply the rotation of the cube round its three axes. If we presume that the cube is lying flat on table it will only be free to rotate around one of is axis. To find this angle as simply as possible we note that the camera will be always be positioned at a higher level than the cube. That means that the for upper corners will never be hidden neither by other cubes nor by it self. This could of course be solved anyway, but by using a somewhat more complicated method.

As we already know the position it is possible to give the cube an initial angle of rotation and make a projection of its corner points onto the screen. Those can be compared with the ones actually found in the image generating an error E. We will now try to minimize the error by rotating the cube. As E is always positive we need to use to different guesses of rotation of the cube to get aware of in which direction to rotate the cube.

All this is of course something that has to be tested out practically. As the methods are general it is possible to choose simply searching for the corners or

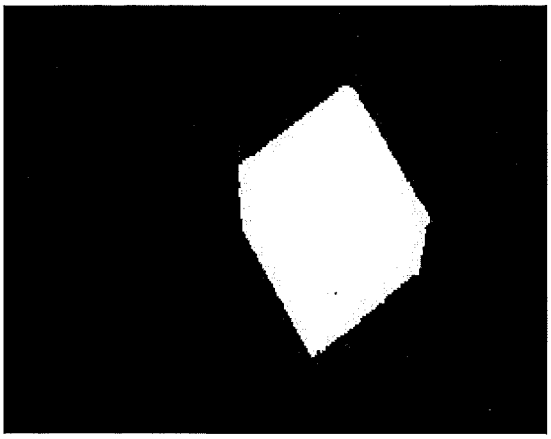

Figure 18: A thresholded version of figure 17 to find the cubes center

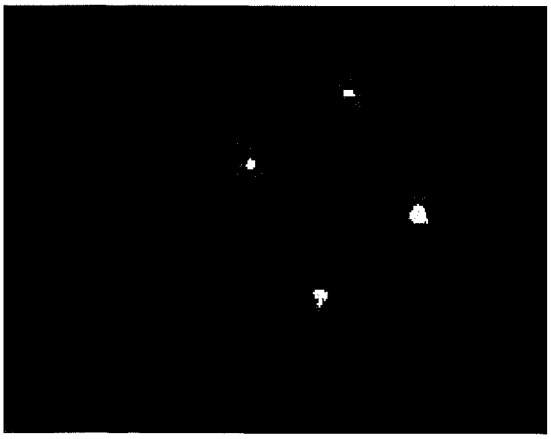

Figure 19: A thresholded version of figure 17 the corners of the cube

to make some kind of combination of the techniques. We could for example determine the three-dimensional positions for cubes and corners independently and later on try to merge this information together.

2.3.5 Identifying objects

As we mentioned before we have to help the vision system to interpret what it is seeing. We tailor the system to give it as much information that we can without loosing to much generality. Here we tell the system that all it can see is cubes. But the cubes are not identified as individuals, they are just cubes, all alike. For some purposes, like building towers out of the cubes that suffice but because that we will have the robot to play scramble it is not suffice.

There are some experiments done where vision systems have learned to distinguish between object itself. Objects were measured in some aspects, for example size, texture and color and the result was put into a database. It could afterwards decide itself how many different kinds of object there were, and how to find them. Here we know what is the difference between the cubes: the texture. All cubes are given six properties, namely the letters on the sides. The texture of the side of the cube can simply be compared to 29 prototypes, the Swedish alphabet including W.

As we know the already the position and the rotation of the cube is it possible to do an inverted projection, from the side of a cube to a plane, and compare it pixel wise with the prototypes. The prototype with the least "distance" to the reprojected image is used. To minimize errors due to lack of precision of three dimensional coordinates of the corner points of the cubes bold letters are used. If we have a slender **I** on the side of the cube but misjudges the corner position with a couple of millimeters the system will record that as a total failure and another letter will be chosen instead.

This an isolated problem, quite different from our main tasks, that we unfortunately have not had time to implement.

2.3.6 Algorithm of the test system

The main task of our project wasn't just to present the theory a system, but as well to implement it. The first part of the problem was to implement the functions discussed above. The second is to implement the "logistics", the part that sorts and moves data around the system.

Matlab was well suited to solve the first part of the problem because of two reasons: Thanks to the visualization possibilities in Matlab it is easy to verify that you have succeeded or at least to find algorithmic errors. Secondly, all functions useful to solve the problem, as matrix multiplication SVD-decomposition

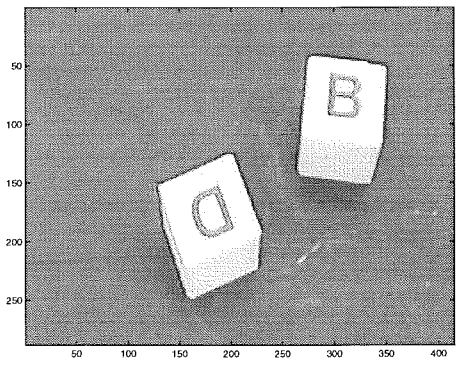

Figure 20: A sample image of two cubes

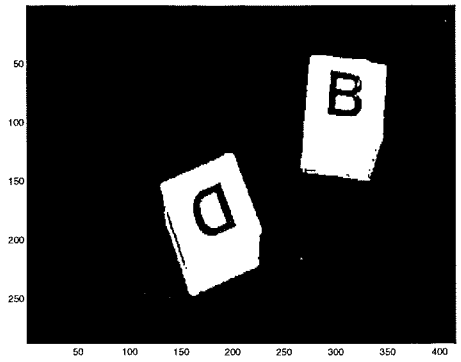

Figure 21: Threshholed image of figure 20 to recognizing letters

etc, are already implemented.

The drawbacks with using Matlab, which also was why we abandoned this concept later on, was the lack of speed and communication problems with surrounding C-programs. It was impossible to use the algorithms real-time. Instead we had to gather data while moving the robot and process it afterwards. This was also partly due to the problems to get the video image from the C-program that handled the camera to Matlab. As Matlab did not support pipes properly we had to pass all files as complete shared files.

Most of the problems discussed earlier in the report were solved in Matlab with the limitation that we could only find one three-dimensional point at a time. This was because of the problems using data-structures in Matlab. There exists some kind of data-structure support, but it is not possible to compile that

part of the code.

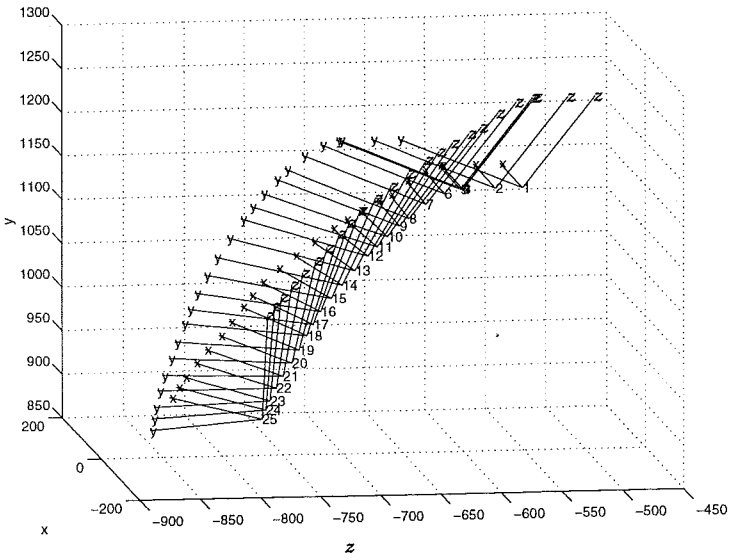

Figure 22: Trajectory used to calibrate camera

2.3.7 Algorithm of final system

The programs, although they work all together, can be divided into three different categories:

1. *image manipulating*

   searching the image for interesting points

2. *logistic* transports information round the system

3. *calculating* programs implementing the theory from the earlier chapters

Unlike the former projects were they where using Matlab this is supposed to be executing real time. After the first phase of the project when the main algorithms where tested using Matlab we they were rewritten into C++ for speed. C++ allows us as well to use data structures and objects in a totally different manner.

C++ classes of the program

This sections deals with the inner data structures of the c++-programs. If your not really interested of the details of the program you can browse throw

this section.
The whole program is based upon objects. Each transformation is for example one object, as is the points and the input image. This makes programming much easier as the points can calculate their own position and matrix multiplications can be performed without keeping in mind the dimensions of the matrixes. Here follows a presentation of the most important classes to make the flowchart *Figure* 2.3.7 easier to understand as well as the program code.

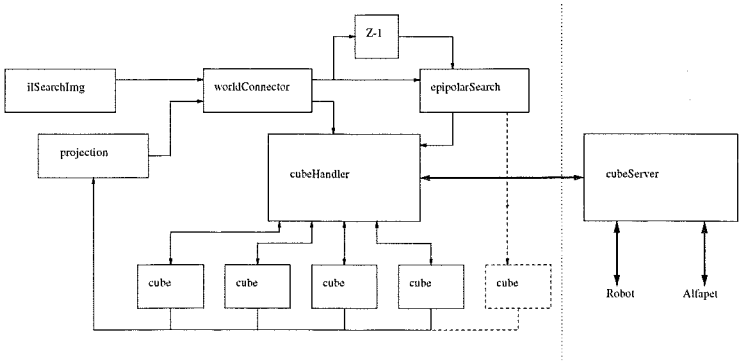

Figure 23: A data flow diagram between the main objects

The different objects

- *ilSearchImg* a subclass to ilDataImg found in the silicon graphics video image library
- *cubeHandler* a class that is administrates the cubes seen by the vision system.
- *epipolarSearch* a class to which coordinate, position and rotation data is sent. Out of that information it matches points from the different images together and creates an new point3D.
- *projection* is a class who performs a projection of a list of three-dimensional coordinates and the robot pose onto the screen.
- *point3D* is a class that represents a two dimensional point. The pose of the robot is entered together with a two-dimensional coordinate and point3d calculates it's position out of that data.
- *linkToEnvision* a class that handles the synchronization between the databases of Envision and the vision system.
- *mlMat* a wrapper class for handling matrixes by using the cblas library
- *point2Dsize* a subclass to mlMat allowing different indexes to be added to the different columns of the matrix. It is used to store the coordinate

information of a series points, either directly from the ccd-matrix of the camera or from a re-projection. The indexes that are used to indicate the size of the points found in the image and to which point in the model of the scene that the point belongs to.

of the algorithm

As mentioned earlier a lot of knowledge, and also limitations, are already built into the system. It can for example only see cubes. Anything else, with approximately the same size as the cubes will be mistaken for being cubes. This is simply due to the method of identifying all cubes as is center point.
The system is constantly searching to increase it's knowledge of the world. It consist of an inner loop using it's old knowledge, the cubes already found to analyze the new images, and an outer loop getting information from the camera used to find new cubes. The inner loop consist itself of an inner loop finding the rotation of the cube the system is about to pick up.

The inner loop The purpose of the inner loop is to reuses it's old information by reprojecting already known points onto the screen.
The cubeHandler who is administrating the cubes collects their coordinates and passes it over to projection. The projected coordinate as Here is the meeting point of the inner and outer loop.

The outer loop

The input picture is represented by ilSearchImg which changes every time a new image is taken. The points found in the image is then delivered for further processing as a point2Dsize object. It is sent to world connection that searches throw these points for matching points among the cubes already found. Here it could be useful also examining the size of the cube found and predicted.
Those points found that will not matched with any old ones are then passed to epipolarSearch

2.4 The final system

In order to get a system, which is able to play Scrabble, we need a main module which coordinates the designed robot system and vision system. This can be designed as a state machine. Hence, the game can easy be divided into different phases: move to cube, play word, etc. We chose to divide the state machine into the following phases: getNextCube, scan, checkCubeStatus, pickingCube, identifyAndAnalysTheCube.

The state **getNextCube** checks if there are any available cube, i.e. if the vision system found any cube in the pile of unused cubes. If so, one of the available cubes is chosen and the state changes to checkCubeStatus. If not the state changes to Scan. There is one exception: if all the cubes are used, the pile is empty, the new state is identifyAndAnalysTheCube.

The state **scan** requests the robot to perform a motion, in which the board is scanned. If a cube is found during the motion or if the motion is completed the state changes to getNextCube. Note that we assume that at least one cube is found if the motion has been completed.

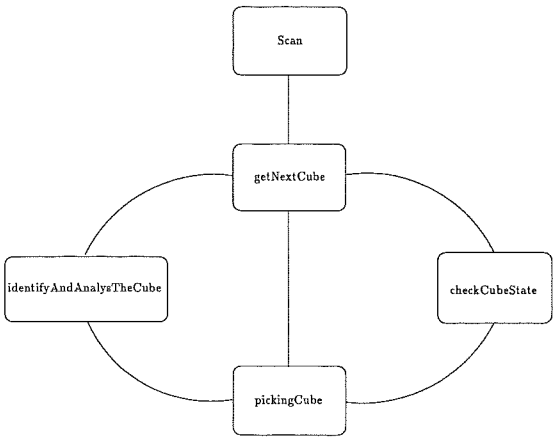

Figure 24: state machine

The state **checkCubeStatus** checks if it possible to grab a cube, e.g. checks the cube vicinity. If it is possible to grab the cube then the state changes to picking, if not the state changes to getNextCube.

The state **picking** requests the robot to move to the chosen cube and grab it. When the robot is finished the state is changed to identifyAndAnalysTheCube. If the cube during the motion becomes grabable or it is lost then the robot stops and the state changes to getNextCube.

The state **identifyAndAnalysTheCube** identifies the cubes letters and puts it on the "hand". The board has a defined area which is referred to as the "hand", this area contain the cubes which later is used to construct a word. If the hand becomes full, contains 7 cubes, the Scrabble algorithm is requested to make a word, using the cubes on the hand. Afterwards the robot is requested to put the word on the board and the state changes to getNextCube. If the hand does not becomes full, the state changes to getNextCube.

In order to get the robot to perform the different motions a GSL-program can be used. In the GSL-program all the different motion is specified by different tag points. We refer to the GSL-program as the RobotController. The RobotController communicates by socket with the state machine.

The cubeserver, which uses the LLTI to communicate with the cubes, is expanded. It needs to know: If the cube is grabable, which side is the upper. It also set the cubes state. The cube can be in two different states: grabbed or not. When grabbed, the vision is informed and the feedback from the camera does not effect the robot movements. It is the state machine which informs the cubserver of the cubes state. It also sets the current chosen cube in focus to facilitate the user which cube is chosen.

Now we only need one more module, the Scrabble module. This module can from given cubes construct a word which give the highest point possible, if it

is put on the board. There are many ways to do this. The easiest is to check the given cube letters against a database, which contains a list of words. We did this but this method is rather slow. Therefore we design another algorithm this algorithm is described in Appendix F. We chose to implement the state machine in Java and the cubeServer in C. C is suitable since the LLTI include C-routines. The reason why we chose to implement the state machine and the Scrabble algorithm in Java depend on the benefits the language provides.

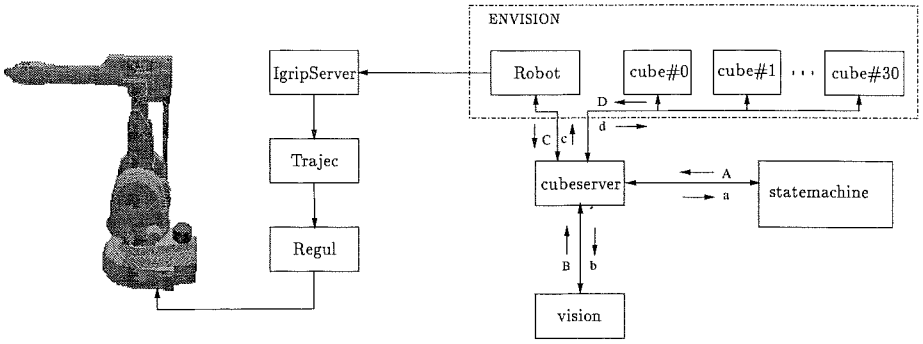

Figure 25: Final system

All the signal between the modules is described in appendix D.

3 Results

The greatest challenge in the project was to get a working robot system and vision system. We have designed and implemented the two different systems mentioned in Chapter 2, one with Matlab and one with Envision as the application to generate the trajectories. Both implemented robot systems succeed to generate a precalculated tentative trajectory and use sensor information from the camera as feedback to correct the trajectories. Unfortunately did we not get a vision system that works and all following results have been received by simulate a vision system.

The first system precalculate the trajectories and there is one advantage in precalculate the trajectories, you have completely control of the trajectory, its time stamps,via points and velocity references. Hence, we can guarantee that the trajectory is smooth and that it not includes any discontinuities. Also the speed of the robot is known. The correction often works well, but it has problem if it is necessary to correct the trajectory more than 70 cm. This is not normally a problem since the correction that is needed is far less than 70 cm. The first system will at best have a precision of 1 mm, since that is the resolution of the correction. That is acceptable in our application. If better resolution is needed, another method to correct the trajectory is needed.

The second system we implemented consists of a robot system with Envision and a vision system in C. The implemented robot system can both generate trajectories and correct them in real-time. The real-time correction in Envision by using tag point works well, if sufficient tag points are used. We used 16 tag points which is on the low side. But the drawback by using is that we can not make any correction between the tag points. This is not a large problem in our application, since our cubes do not move. Therefore we do not need any fast correction, but in a such case there have to be very close distance between the tag points. The real-time correction in Envision does not have the problem with precision and speed as the former system, since the tag points are attached to the cube. This guarantees that the correction is completely performed and the precision of the system is not limited by the correction. Generating the trajectories in real-time by using tag points is easy, but it has problems. There is a real-time problem when using a soft real-time application as Envision to real-time generate trajectories connected to a hard real-time system as our robot system. The real robot Irb-2000/3 shade the simulated robot in Envision and sometimes it happens that Envision do not fulfill the real-times demands. The simulated robot in Envision jumps which also makes the real robot jump. This behavior is not wanted, but the jumps are very small so it is acceptable in our application. In another application even small jumps are intolerable for example welding. We reduce the problem by only running the application Envision on the workstation and by lowering the graphic resolution in Envision. Then the jumps did occur less often. We have noticed that the time stamps generated in Envision are sometimes wrong, but we have not succeeded to find why this

happens, but it looks like the time stamps when this happens are all the same in between two tag points. This gives the robot system problems.

The final system which includes the scrabble controller works. But it is only use one letter on the cube, it is not capable to handle six letters on each cube. The implemented scrabble algorithm works well, the time to check one potential start letter is approximately 30 ms. The controller that manages the different phases in scrabble works, but due to the time limits of the project we do not have the time to implement the cubeserver and the vision system, so it support cubes with 6 letters. The system we have accomplished is only able to handle one letter on each cubes, but it is prepared for cubes with 6 letters. To get the system to work, the IgripServer and Trajec had been expanded to support commands and correction. The implementation works and the new commands is Grasp, Drop, MoveJoint, and MoveCart.

When we implemented the final system we noticed some faults in Envision which are presented in appendix C where also a way around the faults are presented. The project has generate several side effect on the robot system. The project have also generated several side effect is presented in appendix E

4 Discussion

In this chapter we discuss possible improvements of the system. All the improvements will be in the robot system, improvement in the vision system is discussed in the vision report [9].

We did not find the problem that generates the fault in the time stamps. This has to be solved if we should get a robust system. The method for management of 6 letters on each cube has to be implemented in the cubserver, if our goal should be fulfilled.

The system we have achieved contains no feedback from IgripServer to Envision and this is a problem. If we stop the real robot, we also want the simulated robot in envision to stop, this is to be able to continue the motion. In our system we must reboot the robot system if the robot is switched to standby when Envision is generating trajectories. This can be solved by IgripServer telling Envision it is switched to standby. Then Envision stops the generation of the trajectory. All this communication takes time during which via points have been sent to IgripServer and buffered in Trajec, and to be able to continue the motion they have to be buffered. This improvement will make the system more "user friendly". It is also needed to update Trajec with the position of the real robot when it switches from standby to run. We have used a priori knowledge about the table height. This is not necessary to know because we can get the table's height by using the robots force/torque sensors or use triangulation between images in the vision system. The robot position which the vision system has used is the simulated robot position and there is some delay between the two robots that lowers the vision system precision. We can improve the system if we instead use position from the real robot. If the system works well it is possible to get the two robots in the RobotLab play against each other, since they have common workspace.

5 Conclusion

The purpose of this project was to get a robot to play Scrabble with the help of a camera. This has not completely been achieved. We have a system that includes a robot system, a vision system, and a scrabble algorithm. The robot system and the scrabble algorithm works well, but the vision system does not work. Even if it not works we know that it is possible to get a robot to play scrabble. Even if the vision system not work we have seen by simulating the vision system that the generation and correction of the trajectory b using the feedback from the vision system works well.

We have used Envision in the final system and we found Envision is a good simulation environment. A lot of time has been saved by simulating in Envision. Except for the errors in Envision, see Appendix C, the only problem is that it can not meet the demands of a hard real-time system. Even if we have focused a lot to get the robot play scrabble the robot system and the vision system we have achieved can be used to a lot of other applications in the industry. A system that is able to use the information from a camera to change the trajectory of the robot is still not common on the market.

A Control of the robot speed using feedback from vision

If, for some reason, Vision needs long time to calculate the cube position, you do not want the robot to continue with unchanged speed. The performance of the motion will be better if it is a quite short distance between the images. It is possibly to observe the time distance between two images and to calculate the distance between two images. Preferably, the time difference is used because of the fewer calculations needed.

Before deciding how to control the speed, the IgripServer must be modified to be able to perform a change of the motion speed. There are two methods to slow down the speed of the motion. The first is to change the time stamps of the via points or extend the trajectory by adding new via points in the trajectory. The first method has a drawback: the Trajec does not support changes in the time stamps and IgripServer needs some structural modifications. The second method does not have this disadvantage. Even if the current Trajec and IgripServer not supports extending of the trajectory, the needed changes do not cause any structural problems. In fact by generating more via points we will smooth the motion. By using the relative speed to set the speed, there is no need to know the actual speed. To generate the new via points you could use interpolation. Every via point need a corresponding time stamp and velocity reference and the former via point time stamp and velocity reference could be used. The behavior of the robot speed must fulfill: The Robot must not decelerate if not a specified amount of time has passed, in other words if the time difference between two images is larger than the specified amount of time, the speed should be decreased. If instead the time difference between the image becomes small, the speed should be increased.

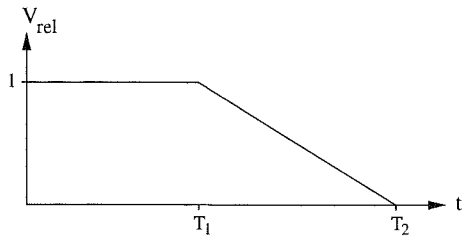

Figure 26: velocity

Figure 26 shows how the velocity depends on the time. t is the time since the former image. T_1 expresses the time when there is need to slow down the robot. If Vision needs more time than T_2 it should be stopped. We formulate

this as:

$$V_{ref}(t) = \begin{cases} 1, & t \leq T_1 \\ 1 - \frac{t-T_1}{T_2-T_1}, & T_1 < t < T_2 \\ 0, & t \geq T_2 \end{cases}$$

If V_{rel} is used to decide the speed of the Robots, it will result in a motion which very often changes the speed. To get rid of this the model extends to include the time between former images and the assumption: the current image needs the same amount of time as the former image.

$$V_{new}(t) = \begin{cases} V_{rel}(T_{Pold} - t), & t \leq T_1, T_P < T_1 \\ V_{rel}(t), & T_P \geq T_1, t \geq T_P \\ min(V_{rel}(T_{Pold} - t), V_{rel}(T_P)), & T_P \geq T_1, t < T_d \end{cases}$$

T_P is the time Vision needed to treat the former image and T_{Pold} is the treating time for the image before. This controller adapts the speed fast if it needs to be decreased, but the increase of the speed will always be one image delayed.

B Commands implementation

The IgripServer need to support the commands Grasp, Drop, MoveCart and Move Joint if we if we should be able to get the robot play Scrabble. We will first describe the methods for the commands Grasp and Drop. The Irb-6 robot system already have a I/O-module Gripper in the static part of the system, because of the open robot control system, see [3] we make the new module Equipment, this will be dynamic linked to the system.

The IgripServer is prepared to support commands, it sort out the commands from the trajectory. All the commands have a negative number instead of a time stamp and that make it easy to sort out the commands. The number that is used is:

command	*number*
Grasp	- 5
Drop	- 6
MoveL	- 7
MoveJ	- 8

The Grasp and the Drop command look like [command type, tool, force, grasp time, 0, 0, 0, 0, 0, 0, 0, 0, 0] for the Irb-2000/3. The Irb-6 robot have one joint lesser and the grasp command look like [command type, tool, force, grasp time, 0, 0, 0, 0, 0, 0, 0, 0, 0]. The tool value determine which tool should perform the command. The force value determine the force which is used to open or close the tool. The Irb-6 robot can not vary the force. The grasp time determinate when the Grasp/Drop command will perform. We connect the Grasp/Drop command to a via point in the trajectory by using the grasp time. This make it easy to know in the call-back procedure when to perform the Grasp/Drop. In order to know when when the Grasp/Drop command is to be performed in the call-back procedure the context of Trajec has to include the commands. For that reason we expand the context to include ClientContext. The Client context is a address reference to the Grasp/Drop commands. The IgripServer expand to also contain a process which management the performance of the opening or closing the tool. This solution is almost correct, but this will make the Grasp/Drop command perform one step earlier than wanted, due to the structure of the Trajec and IgripServer. This problem is solved by the context also contain the time when next step will perform. This element is referred as GraspDelay.

The MoveL and MoveJ look like [command type, joint1, joint2, joint3, joint4, joint5, joint6, motion time, 0, 0, 0, 0, 0] for the Irb-2000/3. The Irb-6 robot have one joint lesser and the grasp command look like [command type, joint1, joint2, joint3, joint4, joint5, motion time, 0, 0, 0, 0]. Joint1 to joint6 is the end position value of the joints. Motion time is the time during the robot should perform the motion. We will not allow to mix the command MoveL and MoveJ with the trajectory, i. e. there can not be any MoveJ or MoveL between

the via points in a trajectory. This is the only restriction in the use of the commands.

C Envision problems

Envision is not a new program, it have been on the market for several years. The version we used was released in December 1998. But even so it promise several function do not work. In our application we need to have communication between our state machine and the GSL-program. The ordinary socket communication in GSL is blockade, this result in that everything is stopped until someone sent a message to the GSL-program. This is not acceptable. In Envision there is a function called set async socket and it promise a socket communication that is not blockade. This function do not work. We must have communication both way between the state machine and the GSL-program and there is a way to solve this. The problem is when the GSL-program reading message, writing does not generate any blockade, so we have to use a other way when the GSL-program read. The LLTI involves routines to set variables in a GSL-program. If we let the state machine use the LLTI when writing to the GSL-program will it be possible to communicate both directions with the GSL-program.

The a other function that it promised is dout, this uses to pass signal between devices. We wanted to use this to control the gripper, but it does not work. This problem can be solved by using same method as before, using the LLTI. Instead of use this dout function a variable in the GSL-program can be used and by LLTI the value of the variable can be checked. The LLTI make it also possible to open or close the gripper.

By the LLTI Envision promises several function that could be used to change the robot speed and acceleration. No of these function does actual work. Hence, have we not been able to implement the speed controller described in Appendix A into the final system with Envision.

D The signals in the final system

The signals referred as **A** in figure 24

signal	*description*
J2CS_pick_block	order cubserver to pick the block
J2CS_stop_picking	inform cubserver the block is released
J2CS_find_letter	order cubserver to find letter
J2CS_block_grabbed	inform cubserver the block is grabbed
GSL_scan	order cubserver to set the scan variable in the gsl program
GSL_find_letter	order cubserver to set the findletter variable in the gsl program
GSL_put_letter	order cubserver to set the putletter variable in the gsl program
GSL_grab_block	order cubserver to set the grabblock variable in the gsl program
GSL_set_vel	order cubserver to set the vel variable in the gsl program

The signals referred as **a** in figure 24

signal	*description*
CS2J_block_found	inform the statemachine a block is found
CS2J_block_lost	inform the statemachine a block is lost
CS2J_block_in_critical_area	inform the statemachine if a block is in the critical area
CS2J_found_letter	inform the statemachine a letter is found

The signals referred as **B** in figure 24

signal	*description*
V2CS_object_moved	order cubserver to move the object
V2CS_put_letter	order cubserver which letter it is on the block
V2CS_create_block	order cubserver to create a new block
V2CS_delete_block	inform cubserver a block is deleted
V2CS_get_robot_pose	order cubserver to get the robot position

The signals referred as **b** in figure 24

signal	*description*
CS2V_pick_block	infrom vision the block is picked
CS2V_get_letter	order vision to identify letter
CS2V_block_taken	inform vision the block is grabbed
CS2V_put_block	inform vision the block is released
CS2V_robot_pose	inform vision the robot position

The signals referred as **C** in figure 24

signal	*description*
CS2R_set_gsl_block_in_critical_area	order robot to set the gsl variabel blockincriticalarea
CS2R_set_gsl_lost_block	order robot to set the gsl variabel lostblock
CS2R_set_gsl_identified_block	order robot to set the gsl variabel identifiedblock
CS2R_set_gsl_robot_pose	order robot to set the gsl variabel robotpose
CS2R_set_gsl_scan	order robot to set the gsl variabel scan
CS2R_set_gsl_grab_block	order robot to set the gsl variabel grabblock
CS2R_set_gsl_find_letter	order robot to set the gsl variabel findletter
CS2R_set_gsl_put_letter	order robot to set the gsl variabel putletter
CS2R_set_gsl_found_block	order robot to set the gsl variabel foundblock
CS2R_set_gsl_vel	order robot to set the gsl variabel vel

The signals referred as **c** in figure 24

signal	*description*
R2CS_robot_pose	inform cubeserver the robot position

The signals referred as **D** in figure 24

signal	*description*
C2CS_cube_pos	inform cubserv the cube positions

The signals referred as **D** in figure 24

signal	*description*
CS2C_set_in_focus	order cube to set the cube in focus
CS2C_set_out_focus	order cube to set the cube out of focus
CS2C_hide_cube	order cube to make the cube invisible
CS2C_move_cube	order cube to move the cube
CS2C_get_cube_pos	order cube to get the cube positions
CS2C_show_cube	order cube to make the cube visible

E Side effects on the robot system

During the projects there has been several improvements of the robot system. The major improvements is listed below

- The time which a trajectory use to performed the motion nowadays correspond to the ordered time.
- The structure with the IgripServer, Trajec modules was only used to control the Irb-6 robot nowadays also work robust on the Irb-2000/3 robot.
- The Irb-2000/3 analog resolver signals is nowadays filtered and the bus cable is put outside the robot. This has reduced the noise to a tolerant level. Before was the noise level very high and made the robot have a very shaky behavior.
- The positive direction of the joints on the Irb-2000/3 has changed to agree to the robot model in Envision.
- The kinematic for joint 6 on the Irb-2000/3 is nowadays properly implemented.
- The I-part in the regulator on joint 1,4,5,6 is nowadays turn on. Before did the high nose level make it necessary to have the I-part turned off.

F The Scrabble algorithm

There are two ways of making a computer build words out of random letters. Either by using statistics saying "if b is followed by an a it is often a good idea putting a d behind" or by using a word list. We chose the word list which greatest reward is that it constructs only existing words.

Our scramble algorithm use the common rules of Scrabble. We have to use an already existing letter on the board, so the first step is to choose a letter on the board. Using this letter, words can be constructed in two different directions, from left to right or from top to bottom. To find the optimal word we can not assume the first letter to have any particular place in the word to be played. arbritary.

Our algorithm do find the best word to play. The reason is that it test all possibilities. What gives us the time is simply by excluding all test that are unnecessary. A pseudo code version of our algorithm can be seen in Figure 27. Notice that this is not the real code, it just illustrates the idea of the algorithm and a lot of the special cases are not treated here.

We enter the algorithm at `searchBackwards` and simply check if there are any words starting with the letter we have already put on the board so far. If so we fixate the start of the word trying to build a word as long as possible down streams by calling `searchForward` which test all possibilities. `searchForward` will try to play its different remaining letters by checking with the dictionary if the word written is a part of an existing word. If so it calls itself recursively, each time with `nbr_letters_left_on_hand` decreased with one. After all possibilities are tested down streams we try building one step up streams, using the same strategy as down streams. Every time we succeed in going one step up streams we will check `start_of_word` to see if it may be useful calling `searchBackwards`. The result is gathered during the building of the word always saving the best result.

This is sequence is repeated horizontal and vertical and for every letter already on the board

```
searchBackwards()
{
   for i=1 to nbr_letters_left_on_hand do
   {
      if(start_of_word)
         searchForward;
      searchBackwards;
   }
}
searchForwards()
{
   for i=1 to nbr_letters_left_on_hand do
   {
      if possible_to_play(letter_i)
      {
         play(letter_i);
         searchForwards;
      }
      else
         back;
   }
}
```

Figure 27: The main idea of our Scrabble algorithm as pseudo code

The secret of the algorithm is to easily be able to tell whatever a letter combination is a part of real world or not. This is done be restructuring the words from the lexicon into two different lists (Note the lexicon is Swedish): a "backwards list" and a "forward list". The backwards list is constructed as following:

At start the backwards list and the forward list is lists of nodes containing the alphabet letters a-ö.

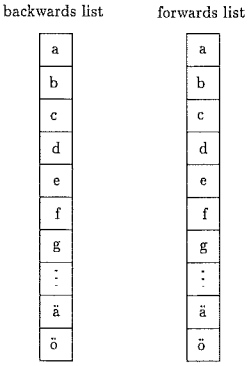

Figure 28: Start nodes

We will illustrate how the backwards list is built in a example.

Example: First a word from the word list is read in our example *bada*. The first letter is read, *b*. Hence, the first letter is a start letter of a word, *b* in the forward list gives the quality of be a start letter. Then the second letter is read, we now have *ba*. To the backwards list is we add a new node.

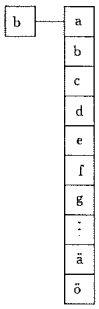

Figure 29: ba

The new node which contain *b* have also the quality of be a start letter of a word. The procedure repeats until the entire word bada is read. The backwards

list will then look like.

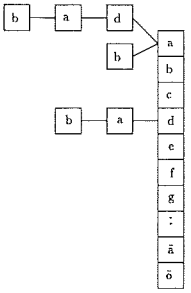

Figure 30: bada

All nodes which contain *b* get the quality of be a start letter. Every word in the word list is treated in the same way and the backwards list grows.

The building of the forward list is different to the backward list. We use a example to illustrate the building of the forward list.
Example: The word *bada* is read from the word list. First is letter *b* read, we note that the following node should be added after *b* in the forward list. Then *a* is read, we add a node after the *b* node after that *d* is read and added after the before made a node. Finally *a* is read, the *a* is added after the *d* node and the *a* is the final letter of the word. Hence, the *a* node gives the quality of be a final node. Every word in the word list is treated in the same way and the backwards list grows.

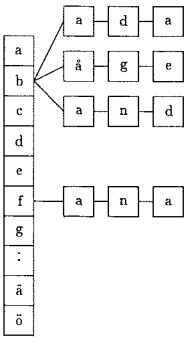

Figure 31: forwards list

A example of how the forwards list could look like when the word *bada*, *båge*, *band*, and *fana* is read from the word list is given in Figure 31.

We now illustrate the algorithm in an example. Assume that the only is one word on the board, ROBOT, and on we have the letters A, D, E, S, U, T, R on the hand. The first step is to choose a letter on the board. We choose R.

When we have chose a letter the algorithm starts. We use the backwards list and notice that R is a possible start letter. Hence, we shift to the forwards list. We take the first letter, A, in our hand and check if the forwards list contains R-A. R-A exist in the forwards list, one possible continuation is D-I-O. Therefor we take the second letter in our hand, D, and check if the forwards list contains R-A-D. The forwards list contains R-A-D and we take next letter in our hand, E. We check if the forwards list contains R-A-D-E. The forwards list do not contain R-A-D-E, since the lexicon we have use to build the forwards list do not contain any word that start with R-A-D-E. We put the E back to the hand and take the S and check if the forwards list contains R-A-D-S. Again we fail, we continue to try the remaining letters on the hand, but we fail to complete a word. Then we go one step up and put the D back to the hand and replace it with E. We check if the forwards list contains R-A-E. The forwards list do not contain R-A-E. By using this manner we will finally have construct ever possibly word that start with R and only contain the letters we have on our hand. Then we a finish with the forwards list shift to use the backwards list. We take one letter from the hand, A, and check if the backwards list contain A-R. The backwards list contain A-R, one possible word is arm. We shift list again to the forwards list and use the same method as before to construct every possible word that start with A-R and only contain the letters we have left on our hand. Then we again shift to the backwards list and take one letter on from our hand, D, and check if the backwards list contain D-A-R. We shift list to the forwards list and use same method as before to construct every possible word that start with D-A-R and only contain the letters we have left on our hand, E, S, U, T, R. We stop the example here. If have completed the example we have found that the word STUDERA is one of the possible word which contain 7 letters. We chose to put this word on the board, since this is the first 7 letter word that we found.

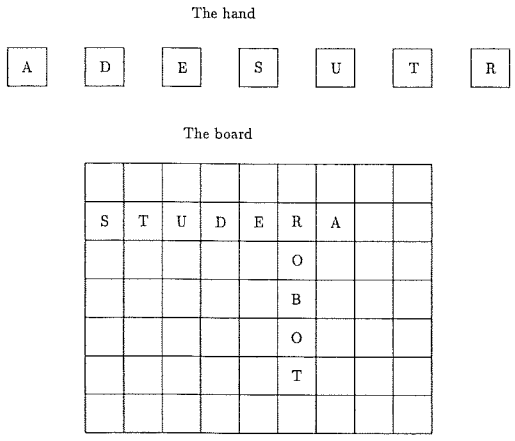

Figure 32: The board

References

[1] Björn Johansson *Studies on Computer Vision Applied to Robotics.* Mastere thesis, Department of Mathematics,Lund, Sweden, February 1997.

[2] Peter Lindström *Computation of geometric structure from an uncalibrated camera on a robot arm.* Mastere thesis, Department of Mathematics,Lund, Sweden, April 1998.

[3] Klas Nilsson. *Industrial Robot Programming.* Phd thesis, ISRN LUTFD2/TFRT–1046–SE, Department of Automatic Control, Lund, Sweden, May 1996.

[4] Jan Andersson. *Ett öppet system för programmering och styrning av robotar.* Mastere thesis, ISRN LUTFD2/TRFT–5557–SE, Department of Automatic Control, Lund, Sweden, May 1996.

[5] *Envision Online Documentation.*
file:/usr/deneb/vmap/docs/envision_HOME/HOMEPAGE.html.

[6] Sven Spanne *Föreläsningar i Matristeori.* Lund Institute of Technology, Department of Mathematics, 1995.

[7] Anders Heyden *Notes from the course Image Analysis.*
www.maths.lth/mathslth/bildanalys/bildanalys.html.

[8] Anders Heyden, Kalle Åström *Euclidian Reconstruction from Image Sequences with Varying and Unknown Focal Length and Principal Point.* Lund Institute of Technology, Department of Mathematics, 1995.

[9] Anders Ahlstrand, Johan Bengtsson *Robot playing Scrabble using visual feedback.* Mastere thesis, ISRN LUTFD2/TFMA–99/3003–SE, Department of Mathematics, Lund, Sweden, 1999.