INTERNSHIP REPORT

INTEGRATION OF A VISION SYSTEM WITH A MOTOMAN HP3 ROBOT SYSTEM AT THE UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

STUDENT: ERALD SCHIPPER (S0140872)

EMAIL: E.SCHIPPER@STUDENT.UTWENTE.NL

MENTOR AT UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ: DR. IR. D.F. DE LANGE

EMAIL: DIRK.DELANGE@UASLP.MX

MENTOR AT UNIVERSITY OF TWENTE: DR.IR. R.G.K.M. AARTS

EMAIL: R.G.K.M.AARTS@UTWENTE.NL

UNIVERSITEIT TWENTE.

DATE: 04-03-2014

MANAGEMENT SUMMARY

This is a summary of the internship of Erald Schipper performed at the Universidad Autónoma de San Luis Potosí (UASLP). The main goal of the internship is to apply a developed vision algorithm in a working application using a MOTOMAN HP3 six degree of freedom robot. A vision algorithm has been developed in former research which determines the location of a point or multiple points on a taken snapshot. The internship has been divided in three parts. In the first part the robot is tested for its capabilities. Secondly, the way of connecting two different systems on different computers is examined. The first is controlling the camera and the vision algorithm in Matlab and acts as the slave system. The second system and computer controls the robot using a library in C++ and this system acts as the master, which controls the slave. Thirdly, the found connection and robot capabilities are combined in a test to run the robot from point to point using an iterative learning process. This way the points on the paper can be found with high accuracy and are used as waypoints for the robot. Also a calibration method for the system has been developed.

The first parts results in the following conclusions. The robot can be moved from point to point in three different control groups. A control group is a group of axis that can be controlled. The control group defines the location of the Tool Control Point (TCP) and the rotation of the used coordinate system. The first one is “base”, meaning the movements are done with respect to a fixed base in the space. The second one is “robot”, meaning the origin of the system moves with the robot. This is relevant in case the robot base is mounted on a moving platform. In this case the robot is fixed to the world, which means that robot and base have the same fixed point in space. The third one is “station” but this one is not used in the system used by the UASLP In this case the origin of the system lies on a specified work station. The control group used is the one called “base”. The robot can be moved using four coordinate systems. The first is joint coordinates meaning that the joints of the robot can be moved separately. The second is cartesian coordinates meaning that the end effector moves in X, Y and Z direction and can be rotated around the TCP. Al systems but the joint system use cartesian coordinate systems but are otherwise defined. Therefore this system is specifically called cartesian coordinates. The third is user coordinates which puts the point (0,0,0) at a defined point in space and rotates the axis of the around some user defined angle around the TCP. Tool coordinates is the last system and this system moves the robot with respect to the TCP at the tip of the robot. The rotations are again rotations of the axis of the coordinate system around the TCP. The TCP can be altered using the software called High Speed Job Exchanger. The exact use of the different control groups in combination with the coordinate systems has not become clear because only one control group was tested with the cartesian en joint coordinate systems. There are three ways of moving the robot: by increment in mm or rotation of the axis, with specified pulses for every joint, or by specifying the coordinates of the final position in millimetres and rotation of the axis in degrees. Joint movement moves the robot in a kinematical favourable way for the robot and linear movement moves the robot with constant speed in a linear manner to the next point. The robot can be controlled using the Motocom32 library. Its functions can be found in the file MOTOCOM32_US.pdf [1].

The connection of the slave and the master is done by making use of sockets. The slave waits until the master opens a socket. There are several functions developed in the master program. It can open and close a connection which has to be established for every separate operation. The master program can send a trigger to the slave to do something, can receive a trigger to do

something and the programs can send and receive data. The slave can open a connection using the TCP/IP command in Matlab. This connection is used to send and receive the data and triggers.

The obtained knowledge is tested by finding several drawn points on a piece of paper. The used method is that of iterative learning. The robot runs an initial path with the TCP within a range of 8 mm of the drawn spot. At every point a snapshot is taken and the slave calculates the deviation in millimetres from the spot to the TCP. This correction is then being sent to the master which constructs a new path for the next run. This procedure can be repeated to iterate to an improved path. At every run the error ε decreases. No further decrease of ε has been seen after 3 iterations. The ε found is not larger than 16μm.

Before the test can be run, the system is calibrated. This calibration is done by moving the robot to a start position in which a single point shows in the camera image. Next, the robot is moved to a square matrix of positions around that point. The square matrix is 20x20 mm^2 and consists of 9 positions. That way the conversion between pixels and millimetres is done. The robot moves a specified amount of times to the point, each time from another direction in a circle around the point. This way the influence of robot kinematics is brought to a minimum and more statistical values are available. Also the influence of distortion due to the lens can be seen in the results. Deviation from the optical centre with 10mm results a maximum of 0.2mm. Because of the iteration steps the found points move to the optical centre resulting in minimum errors due to the distortion.

A manual for the software has been written and can be found in the full report. Also guidelines about how to cope with errors and bugs can be found. In all circumstances should the user be careful on how to use the software as sudden moves can damage the camera or the system.

Table of contents

1 INTRODUCTION

Robots become more and more common in all kinds of industries. One of the processes in which robots are involved is laser welding. Because laser welding is a very precise work where the beam should not deviate from the seam more than 0.1mm because of the laser spot radius just being 0.15mm, there is a need for very precise control. Most commercially available robots are delivered with a controller attached which cannot be altered. This controller directly controls the motors by means of encoders on the motor axes. Unfortunately this system can only be given a path or waypoints and not be controlled any other way. This causes the need to alter the path via some other signal. This other signal is often given by means of vision. By methods of triangulation, the seam can be monitored. This knowledge can be used to alter the path. There are two main ways of implementing an algorithm to solve this problem:

- Teach and play. This method teaches the robot a path. The robot runs the path and the camera takes snapshots every now and then. From the snapshots, an error between preplanned path and an actual desired path like a welding seam is calculated. After this, the preplanned path is corrected with the new information and then executed by the robot. This is useful for coping with clamping and construction tolerances that arise due to earlier productions stages. The main algorithm is written as:
  1. Extract seam coordinates from image
  2. Compare coordinates of extraction with pre-planned path
  3. Correct new path with found error

- Real-time correction during the run. This method gives feedback from the snapshots during the run of the robot and calculates new waypoints of the path during the run. For the sake of time and little present knowledge on the robot and image capturing system, this will not be part of the internship.

The internship is roughly divided in three parts. In the first part, as much knowledge and information about the robot is gathered, including running some tests on velocity and response on C++ code. The information obtained and test results are given in chapter 2. In the second part of the internship a coupling is designed to communicate information between a Matlab and a C++ code running on two different computers. This is described in chapter 3. In the third part of the internship the work is focused on an application in which the information from the vision system is used to control or correct the robot movements, as described in chapter 4. Manuals of the developed software can be found in chapter 5 and conclusions and recommendations can be found in chapter 6.

2 ROBOT KNOWLEDGE

2.1 USED EQUIPMENT AND CONNECTION TOPOLOGY

The used camera is the CMOS camera of Pixelink. The lens connected to the camera is the ZOOM 7000 NAVITAR TV ZOOM. Both parts are shown in Figure 1. The camera is used to capture an image which is processed with a newly developed methodology at the Universidad Autónoma de San Luis Potosí (UASLP).

Figure 1 ZOOM 7000 NAVITAR TV ZOOM LENS and PIXELINK CMOS CAMERA

This method minimizes the error between the image and a proposed image that is generated. The error is minimized by maximizing the in-product of the normalized image and a proposed image. The proposed image is an image which is created as "expected image". In this case the spot is represented as a Gaussian profile. The minimization is done in an iterative optimization process. At every iteration step a new proposed image is generated. The shape and radius of the Gaussian profile can be adjusted by the user. Several shapes can be searched by making use of multiple Gaussian profiles resulting in one or multiple lines. In former research at the UASLP the method is used to find a triangle on an inclined surface. For the application used in this internship a simple application is used in which a single spot is to be found by the vision algorithm. The same method is used to find multiple spots for the calibration. More on the calibration and practical use of the vision algorithm can be found in chapter 4.

The robot used at the USALP is the MOTOMAN HP3 from YAKASAWA. The robot has six degrees of freedom at the end effector. Its maximal payload is 3kg, and it has a repeatability of ±0.03mm. The robot has a teach panel, in which the robot can be given a job or can be controlled manually. The equipment is shown in Figure 2. The used controller is the NXC100 Controller.

Figure 2 MOTOMAN HP3, teach panel and NXC100 controller

The connection topology of the system is shown in Figure 3.

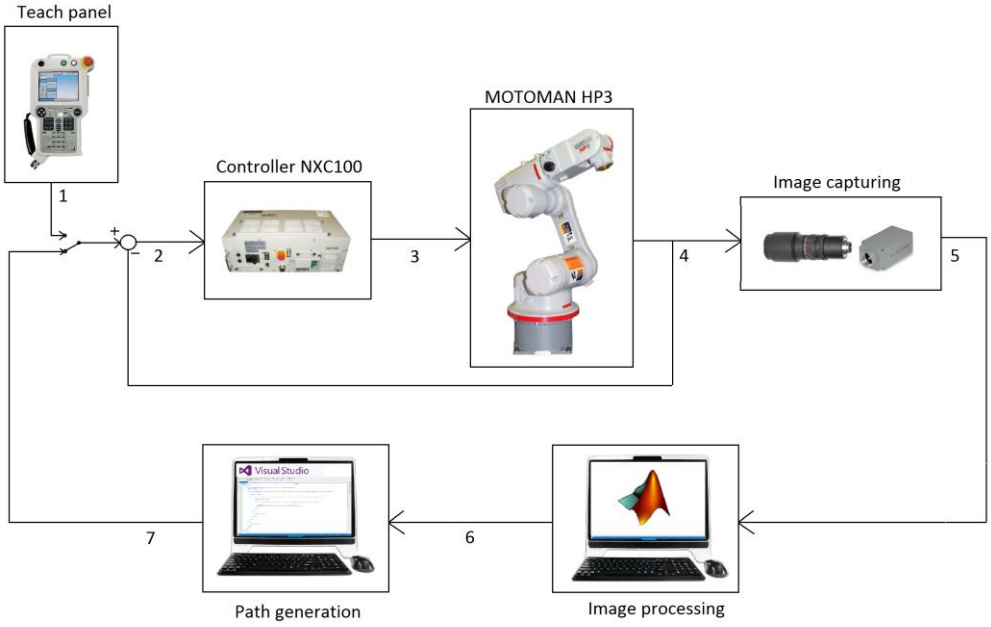

Figure 3 Total system schematics and signal flow chart

The following flows of information are determined:

- -1- The teach panel has 3 settings. The first is *teach,* where a path can be determined by just moving the robot and capturing points in space to which the robot should go at a specified velocity. Secondly, a setting *play* is available in which the robot repeats the path determined during *teach*. The third possibility is *remote* where the teach panel is switched off and an external signal is used, see point 7.
- -2, 3 and 4- At this points the controller calculates the difference between reference and position and controls the robot toward the just positions as specified in the path at point 1 or 7. At point 4 a position is available in different coordinate systems and encoder pulses.
- -5- In the research done so far at the UASLP the camera captures an image and sends the information towards a computer via FireWire.
- -6- The image processing is done in Matlab. The m.files should extract relevant position or shape information from each image. This information is then send to another computer and used for path generation via a C++ routine.
- -7- A program in C++ made in Microsoft Visual Studio receives the coordinates corrections and creates a new path for the robot. The new path is send via Ethernet to the controller. It then can be used as a next learning path for increasing accuracy.

2.2 CONTROL GROUPS

There are three different control groups in which the robot can operate. From now on the coordinates will be given as (X, Y, Z, Rx, Ry, Rz) with X, Y and Z in mm and Rx, Ry and Rz are in degrees. The rotations are rotations of the coordinate system.

- Base. The base control group puts (0,0,0) at some point on the base where the robot is connected to the static world. At 0 pulses for every joint the robot is in the so called

work home position. The end effector is positioned in the point (945.0, 0.0, 445.0, 180.0, -90.0, 0.0). When having more than 1 robot, this coordinate system remains fixed.

- Robot. The robot control group puts (0,0,0) at some point on the robot. In this case the robot is not moving and no other robots are in one system. Therefore this system is the same as the base coordinate system.
- Station. This puts the (0,0,0) at a moving workstation. This is not used is the system at hand.

Only the base control group is used in the project.

2.3 MEANS OF CONTROL

There are two ways of controlling the robot. The first is by teaching a path, saving the path in a job and run the path afterwards. This can be done by the teach panel or by means of programming in C++. With the teach panel the robot can be moved by specifying the motion in the following ways:

- Joint coordinates: The joints can be moved separately.
- Cartesian coordinates: The end effector moves in X,Y and Z direction and can be rotated around the TPC.
- User coordinates: Putting (0,0,0) at a defined point in space while rotating the axis around some user defined angle.
- Tool coordinates: putting (0,0,0) at the tool tip. This point is also called Tool Control Point (TCP). Also the axis can be rotated over an defined angle. The tool coordinate can be altered and the controller provides room for 24 different tools. The original axis of the TPC are as shown in Figure 4. When altering the TCP, fist the rotation around the axis are done, followed by translation in x, y and z direction. How to alter the TCP by the teach panel is described in chapter 8.3 of the NX100 instructions [2]. A second way is to download the tool file called TOOL.CND with the software called High Speed JobExchanger. The TCP of 24 tools can be altered in the file using a notepad. This file can be uploaded again using the High Speed JobExchanger.

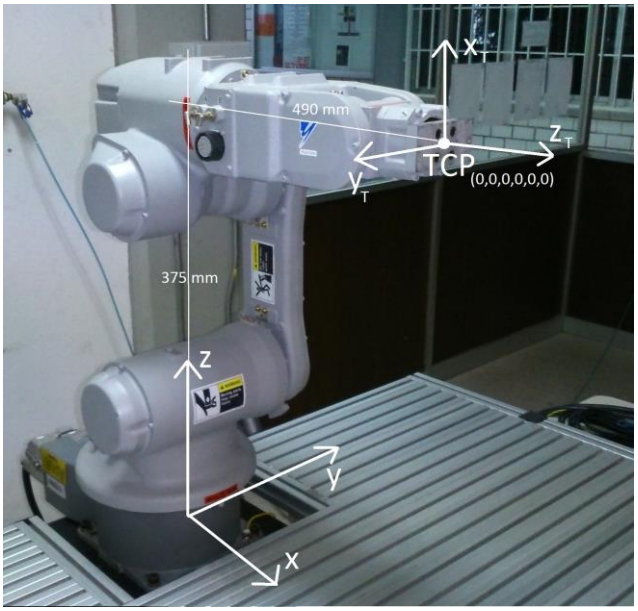

Figure 4 TCP (0,0,0,0,0,0) with respect to the base coordinate system

The manual states the presence of another coordinate movement, namely cylindrical, but for this robot it is not present. The exact use of the different control groups in combination with the coordinate systems has not become clear because only the control group base was tested with the cartesian en joint coordinate systems.

The C++ method requires the use of the MOTOCOM32.dll library. In addition, an example code[1] is available from earlier work done with the robot. In the library there are two main ways for moving the robot. One is by specifying cartesian coordinates with respect to its control group. The robot is capable of going to an x, y and z coordinate with respect to the control group with the end effector and do a rotation of the axis around the Tool Control Point (TCP). It can do this with linear interpolation using the function BscMovl and joint interpolation using BscMovj from the library. The linear interpolation is used when a specific path has to be traced in a straight line between the points while the join interpolation makes a curve that depends on the kinematics of the robot. Another way is to use BscImov which moves the robot by a specified increment in x, y, z direction and rotations around the TCP using linear interpolation. This movement can be done with respect to base and robot control group and the TCP. The other main way of moving the robot is by specifying the amount of pulses for every joint. The pulses can be converted to degrees by $\frac{Pulses}{Degrees} = \alpha_i$ for the corresponding axis as shown in Figure 5. The values are:

$$\alpha_S = 0.000732420823155$$

$$\alpha_L = 0.000549315890441$$

$$\alpha_U = 0.000732422557497$$

$$\alpha_U = 0.001098634224191$$

$$\alpha_R = 0.001098630666737$$

$$\alpha_B = 0.001757812500000$$

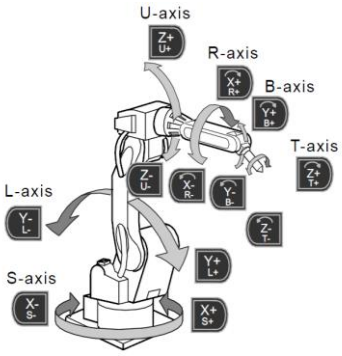

Figure 5 Axis nomenclature

Again the robot can be moved by BscPmovl and BscPmovj as with the method using Cartesian coordinates.

2.4 TESTING THE ROBOT

2.4.1 ACCELERATION AND DECELERATION

There is limited information about the reference signal of the robot. In the job commands, the acceleration can be set, with respect to some maximal acceleration. Using the MOTOCOM32 library, it is not possible to specify the acceleration. Therefore, some tests are performed to test the default reference signal. The first test is the time it takes to cover a distance. The time necessary for acceleration and deceleration is then calculated as

[1] Written by ING. GABRIEL FERNANDO GARCIA CEDILLO

$$T_{acc,c} = T_{dec,c} = \frac{1}{2}\left(T(x)_c - \frac{x_c}{v_c}\right) \quad with\ c = x, y, z$$

Where $T(x)$ is the mean of the captured time of 25 tests, x_c is the set distance and v_c is the set velocity of 20mm/s. The values for x_C are 1, 5, 10, 20, 30, 40 , 50, 75 and 100mm. The results are shown in Figure 6. The results of the found $T_{acc,c}$ and the standard deviation σ^2over the 9 tests are shown in Table 1.

Table 1 Results of acceleration tests

C	$T_{acc,c}$ $[s]$	σ^2 $[s]$
X	0.0983	0.0131
Y	0.0988	0.0129
Z	0.0994	0.0153

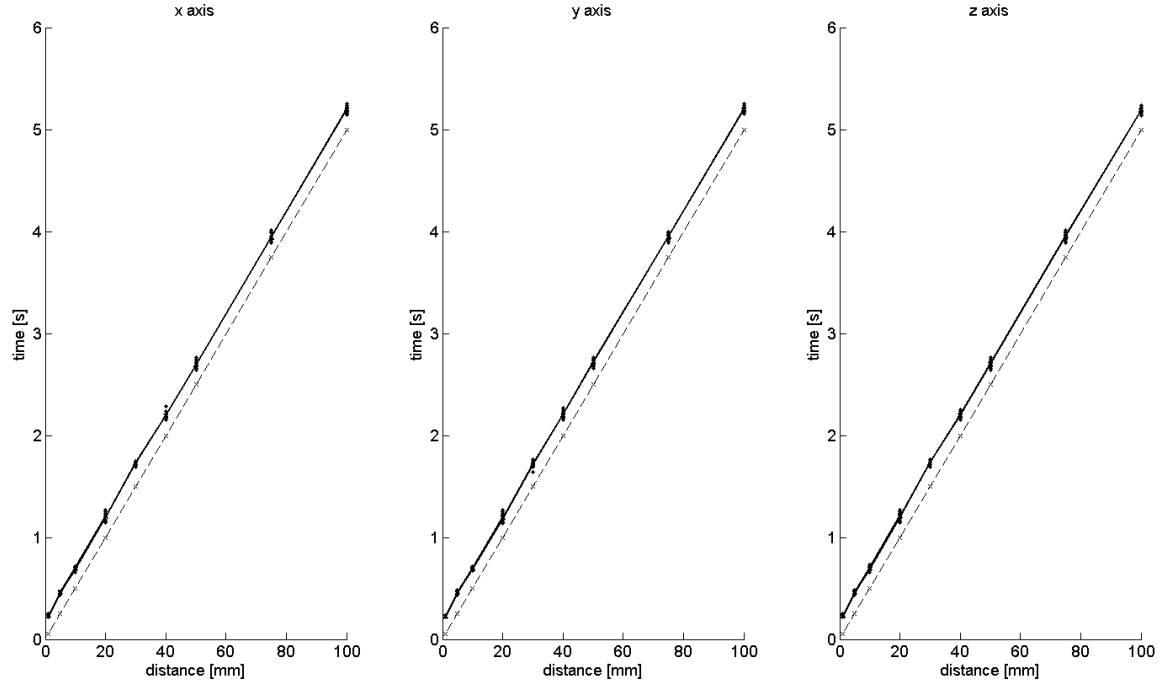

Figure 6 Test result (solid line) and $\frac{x_c}{v}$ (dashed line) with v_c= 20mm/s on different distances

2.4.2 *VELOCITY AND SETTLING TIME*

The second test is run on different velocities over a distance of x_c = 20mm. This test is in done in order to check if acceleration and decelerations changes with velocity. The expectation of the time is given by:

$$T(v)_c = T_{acc,c} + T_{dec,c} + \frac{x_c}{v_c} \quad with\ c = x, y, z$$

The results are shown in Figure 7. Because the errors are small, they are shown in Figure 8.

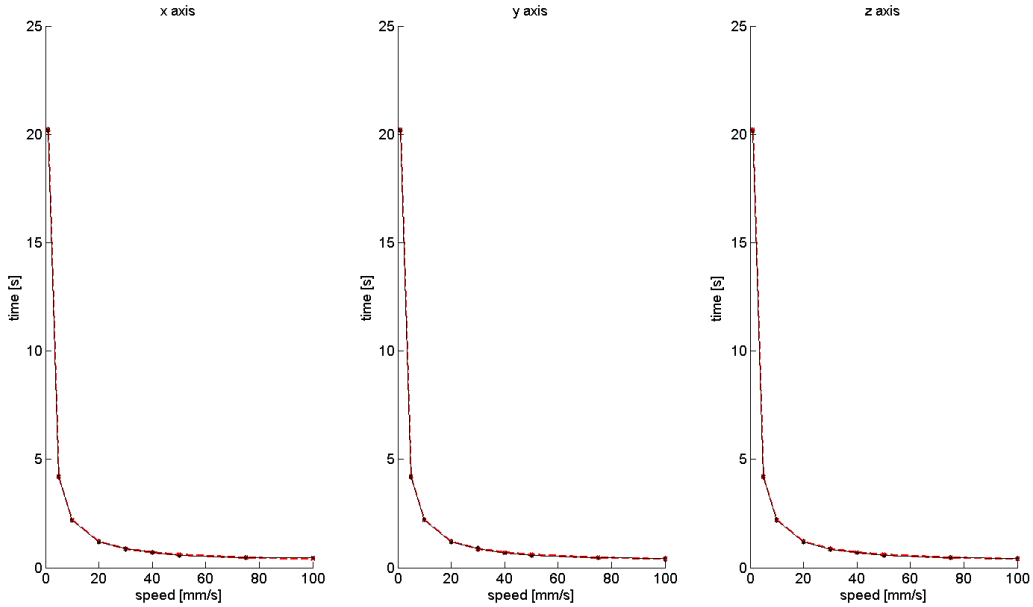

Figure 7 Test result (solid line) and $T(v)_c$ (red dashed line) with x_c = 20mm on different velocities

The errors show minimal deviations but do increase for v=100mm/s. A recommendation is to further investigate the damping of the motion on higher velocities, as it seems that settling time increases as can be seen in Figure 8.

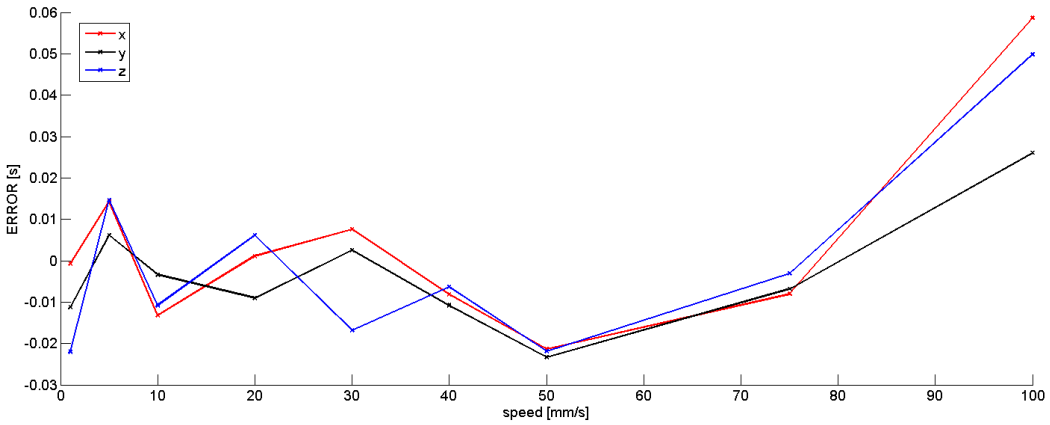

Figure 8 Error between test and theory at x_c= 20mm for different velocities.

2.4.3 CONTINUOUS COMMANDS AND STEPS

While programing in C++ the program reads line by line. This way, the robot can move from point to point. But the code cannot be synchronized with the robot as is possible with a JOB. Therefore programming the robot as:

```
BscMovl(nCid, "V", VELOCITY, "BASE", CONFIGURATION, TOOL, Position 1);

BscMovl(nCid, "V", VELOCITY, "BASE", CONFIGURATION, TOOL, Position 2);
```

will result in the robot performing the commands step by step. This results in starting up and slowing down at each step. The result is that the more steps are taken, not only it takes longer to reach the end position, also the set speed is not accomplished. This was tested over a

distance of 100mm with several steps and the results are shown in Figure 9. The tests are repeated 16 times resulting in a standard deviation of around 0.5% of the time results.

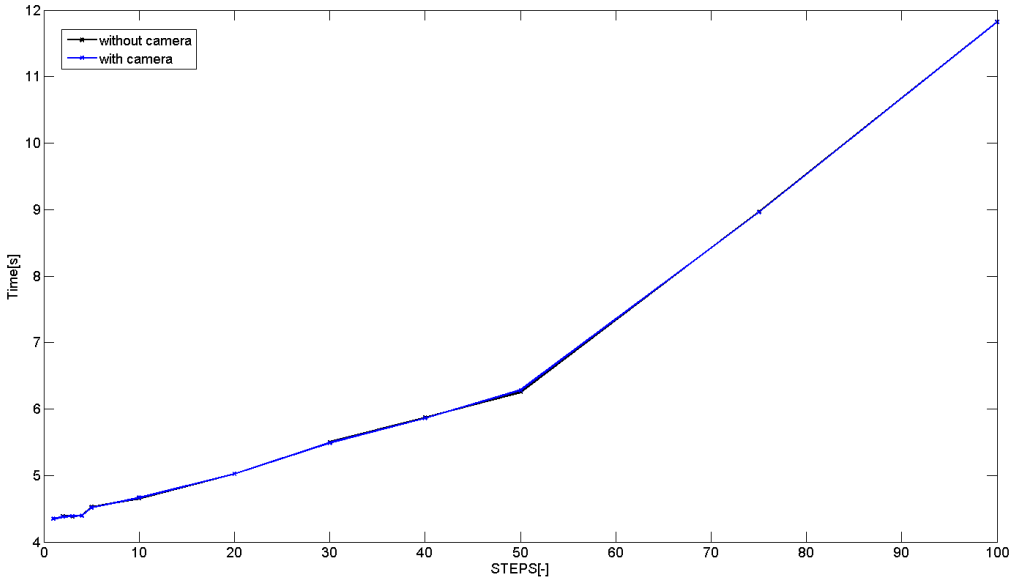

Figure 9 Test over 100mm with multiple steps (1,2,3,4,5,10,20,30,40,50,75 and 100).

Another way is to let the robot wait until it reaches the desired position using the command:

```
EsperaRobot(nCid, Position 1);
```

This function reads the robot position and keeps the code in a while loop until the actual position is in reach of the desired position within a specified interval. The used default value is 0.01mm or 1pulse. A disadvantage of this method is that the robot will accelerate and decelerate for every position, resulting in an even more shocking motion. This function can be used for instance when other commands need to be executed on the waypoints.

3 MATLAB/C++ COUPLING

As shown in Figure 3, the image processing in matlab has to be coupled to the robot control in C++. From now on the control of the robot is called the master and the image gathering and processing is called the slave. There are four methods to accomplish the coupling:

A. Rewrite the image processing in C++ and couple master and slave via C++ routine.
B. Do the image processing in matlab, couple matlab to a C++ communication routine in the slave computer and then send the information via the C++ routine.
C. Do the image processing in matlab and then send the difference of coordinates in mm to the master computer in matlab and transfer the information in the master computer from matlab to C++. This way a coupling of matlab and C++ has to be designed at the master computer.
D. Do the image processing in matlab and send the difference of coordinates in mm directly to C++ on the master computer.

As the image processing was already written in matlab, method A is not the most favourable, although it is a way of communication with the least translation between different software. In former research a coupling has been made between matlab on both slave and master computer. Therefore the first approach will be method C or D and not B. Because C is less direct, option D is the used method. This method makes use of the so called TCP/IP connection.

In the matlab code on the slave there are two methods. Firstly, the coupling can be made by making use of so called MEX function. The MEX functions can be written in C++ code and used in matlab. This way the connection is made via C++ and used directly in matlab as a function. A more thorough description of the MEX functions can be found on the website of Matlab [3]. Because the MEX function cannot return a variable TCP/IP, a new connection has to be made for every communication between master and slave. A second disadvantage is the dependency of the version of matlab and compiler for the MEX function to work properly. These commands open a connection to the master. It is most important to first open the master connection, else no connection can be made. The code used to open the connection by the slave in Matlab is given by:

```
t = tcpip('192.168.0.15',27015,'NetworkRole','Client');
set(t,'InputBufferSize',8);
set(t,'Timeout',30);
fopen(t);
```

For writing and reading the functions `fread` and `fwrite` are used. The connection is closed and cleared using `fclose(t)` and `clear t`. For further help the Matlab help or the Matlab website can be used.

In the master program the coupling is done by making us of so-called winsockets. The winsockets can be seen as an actual socket were cables can be plugged in in order to transfer data or any other kind of flow. These sockets open a connection with the slave and can be used to send and receive data of bytes. The returning type is always char and must be transformed to the preferred data type. More details on the winsockets can be found on the website of MSDN Microsoft [4]. In the file RobotvE.cpp 4 functions are created for connecting the master with the slave.

The first is `OpenConnectionToSlave()`. This function returns an open socket wich can be used to send or receive messages or data.

The second is `CloseConnectionToSlave(SOCKET Slave)`. This function closes and clears the socket used.

The third is `SendData(SOCKET Slave, char *sendbuf)`. This function sends the char via the socket to the slave were it can be used for further processing.

The fourth is `ReceiveTrigger(SOCKET Slave)`. This function receives a char from the slave and is seen by the master as a trigger. The function used is revc and this function is blocking which means the code will not continue until the trigger is received. A double is returned being 1.

The last function is `ReceiveData(SOCKET Slave)`. This is practically the same function as the one for receiving a trigger with the difference that the buffer is longer to make sure all the data is collected. In the code the received double is rounded to 3 decimals. This was done because it seemed the robot could not process all doubles. As it turns out this was not the case, but the rounding was left in place due to time limitations.

The full code is found in the attached code files. The used IP addresses found in Table 2.

Table 2 used IP-adresses

System	IP4-adress
Master on DELL computer	192.168.0.15
Slave on SUN computer	192.168.0.2
Robot	192.168.0.1

4 SYSTEM CALIBRATION, TESTING AND EVALUATION

In order to put the gained knowledge of the system in practice and to show the usefulness of former research done by the UASLP, a test application has been designed combining the found information. The goal is to couple the image processing software to the robot in order to gain information about the total system, e.g. its repeatability, accuracy and speed of processing. The way to test the system is in this case to locate a few dots with the vision system, and let the TCP of the robot move to the right position afterwards. The TCP is represented by a crossing of three lasers as shown in Figure 10. This application can be used for e.g. drilling at the right location, or welding at spots and several other applications. This test can also be expanded for the use of for instance the determination of a seam for welding. The vision algorithm and calibration algorithm are developed by the UASLP. The calibration values are directly used in the vision algorithm and the output of the both is the error in mm in x and y between the TCP and the found point.

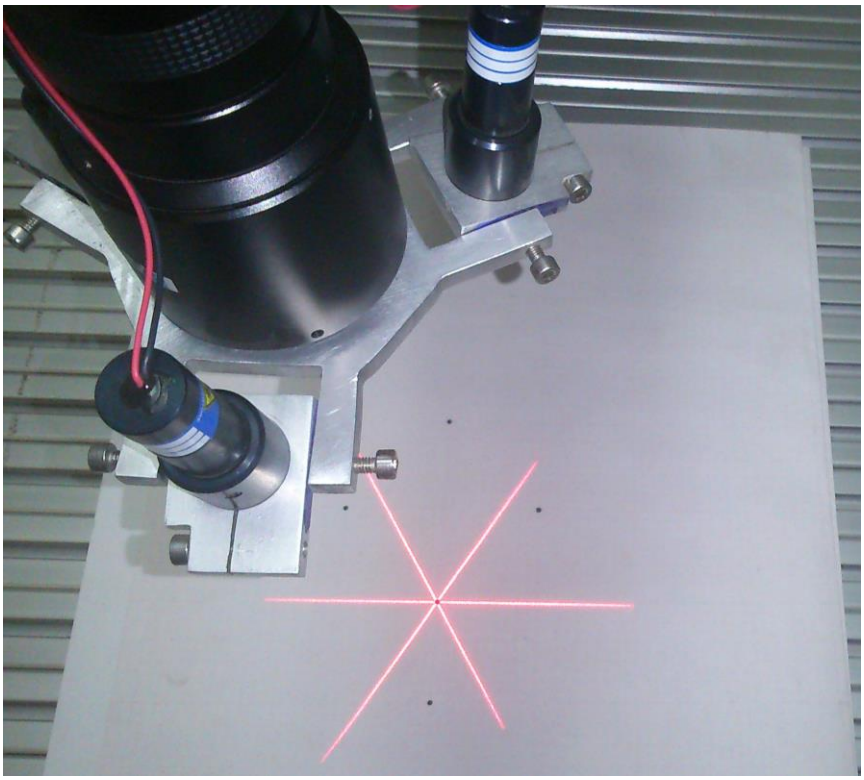

Figure 10 Camera and displayed laser cross

4.1 METHOD

The positions of the spot are determined using the algorithm if iterative learning. On a blank piece of paper, 5 points have been drawn, which have to be captured by the process. The master will instruct the robot and slave to do the following:

1. Move to position near a point
2. Take snapshot
3. Calculate coordinates of the point with respect to the centre of the laser cross
4. Send the found error to the master
5. Move to next point and start with 1, unless all 5 points are captured

After the capturing of the data, the slave sends the data to the master where it is used to construct a new path. This new path is then run again in order to decrease the error. By doing the iterative steps, the errors due to robot kinematics, calibration and lens distortions are taken out. The eventually found path can be run in a demo. The used configuration of dots and teaching path for the first iteration are shown in Figure 11. The program on both slave and master will be explained in the manual in chapter 5.

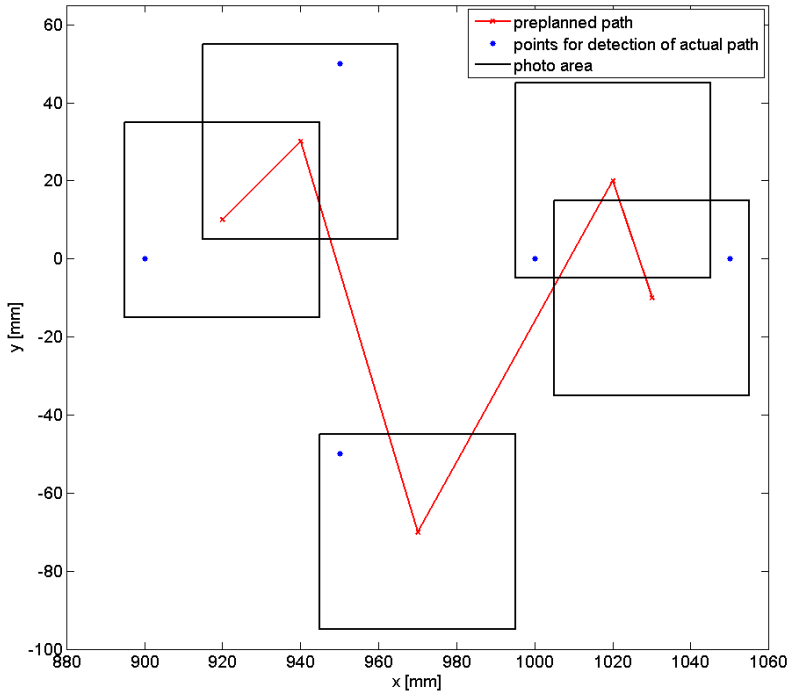

Figure 11 test path and point locations

4.2 CALIBRATION

The camera captures the image in pixels. In order to determine the deviation of the spot from the TCP, a calibration is needed. In this case the following values are found by the software:

1. k_u. This is the factor between pixels in x direction in millimeters of movement in x direction. Its units are pixel/mm.
2. k_v. This is the factor between pixels in y direction in millimetres of movement in y direction. Its units are pixel/mm.
3. α. This is the rotation of the camera coordinate system with respect to the TCP coordinate system.
4. $u0$. This is the location of the cross in pixels in x direction.
5. $v0$. This is the location of the cross in pixels in x direction.

First a picture is made from one of the points. Then the robot moves a fixed distance in x and y directions and takes an image from the two translated positions. The vision software calculates the positions of the points on the captured image and the differences result in the calibration values. Because of robot kinematics and robot repeatability and because more pictures give a statistically more accurate result, the system takes several images, each time arriving from different points around the set points. The algorithm of the calibration is as given in Figure 12. For the calibration at least one layer k is needed. The robot moves to one of the green points and takes a snapshot. Then, before taking the next image, the robot moves to a different location

on the circle, goes back to the same green point and takes the next snapshot. This way the data is more reliable in statistics and the calibration includes possible errors due to different kinematic positions of the robot. This process is repeated n times for every dot. The value of n can be specified by the user. An increment of the angle is made of $\beta = \frac{360}{n}$ until a full circle is reached. For further research on the camera e.g. distortion of the image further away from the centre, more layers k can be specified by the user. First results of calibration on the edges show the outer corner points to be pushed inwards due to lens distortions.

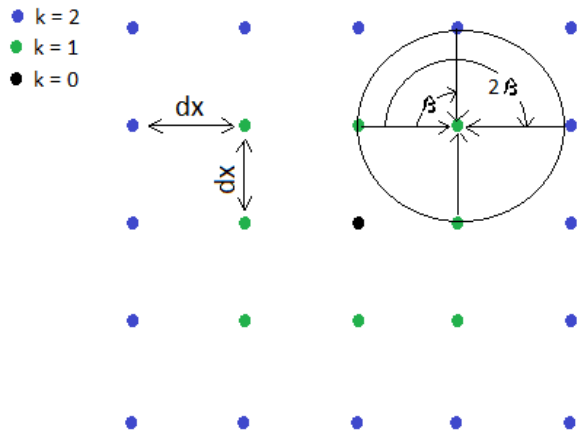

Figure 12 Calibration algorithm

For the laser, n snapshots are taken in the same position because in this case, only the centre point of the crossing has to be determined. The used calibration in the project is done with k=1 and n=16. The found results are shown in Figure 13 and zoomed in in Figure 14. As can be seen the points do not lie in the target position. In order to determine the target positions are subtracted from the found points in order to get a clear view of the magnitude and

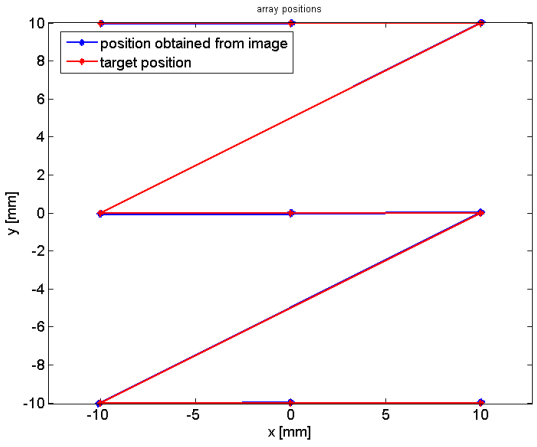

Figure 13 array of calibration positions total

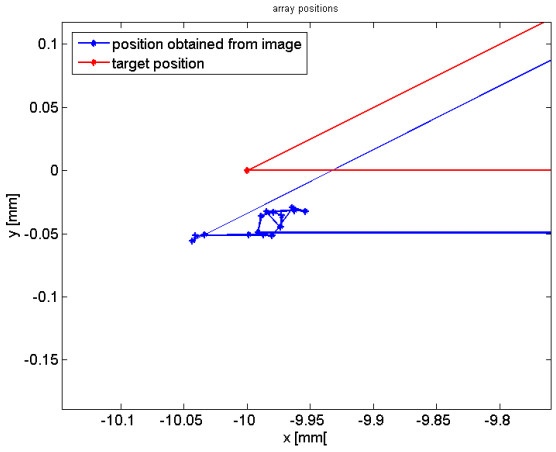

Figure 14 array of calibration positions zoomed in

displacement of the found calibration points. These results are shown in Figure 15. From the figure, two conclusions are drawn:

1. Every point has a variation in a certain direction due to the lens. The calibration is used for research and shows that if the point lies further away from the optical centre point,

the cloud of points have a clear direction. In an area of 20mm by 20mm the deviation lies within 0.2mm.

2. The picture is distorted. The distortion is small, but it is clearly seen in the figure. This can be due to the fact that the paper is not really horizontal with respect to the robot, the pixels in the camera are not perpendicular or the robot movements are not perpendicular.

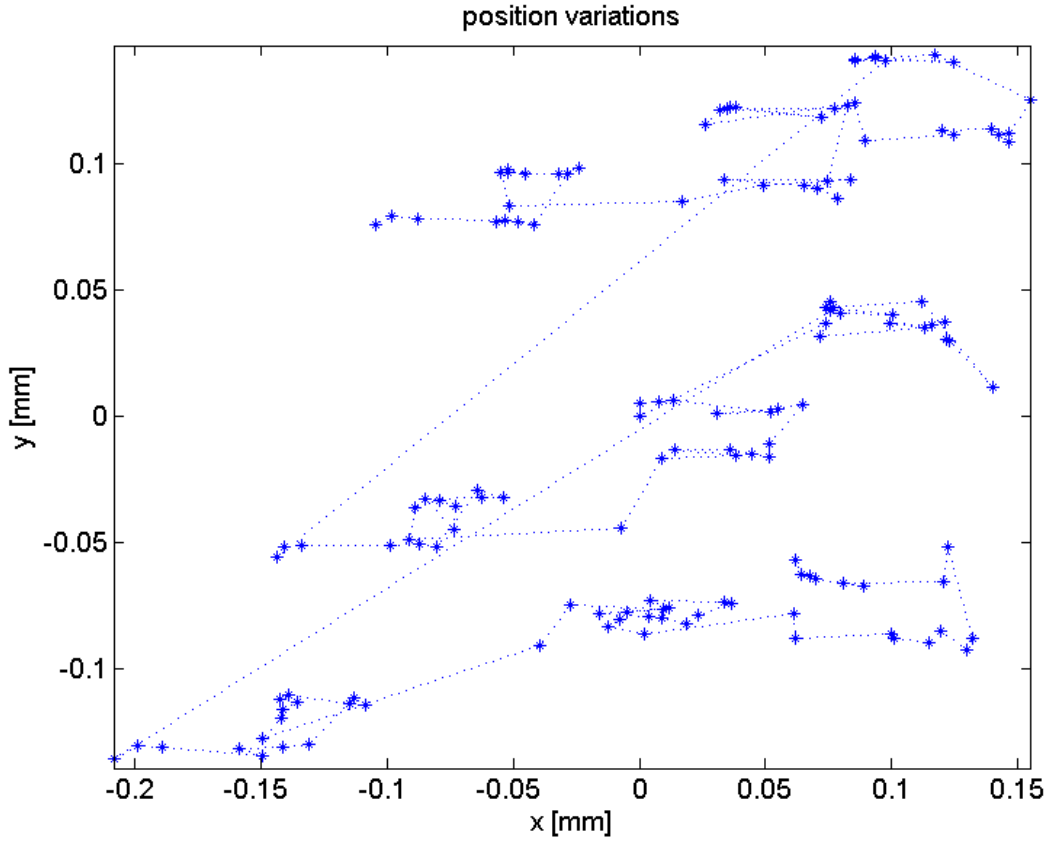

Figure 15 Found clouds of positions variations

The found calibration values are:

1. $k\_u = -11.5897\ pixel/mm.$
2. $k\_v = -11.5941\ pixel/mm.$
3. $\alpha = 1.5809\ rad$
4. $u0 = 468.7840\ pixel.$
5. $v0 = 422.6686\ pixel.$

4.3 RESULTS

The calibration values are used in the vision algorithm. As stated before the iterative learning process is used which return a new learning path. For the test 4 iterative steps are used where the fourth step does not give any more improvement. The four paths of the iterations are shown in Figure 16 to Figure 19.

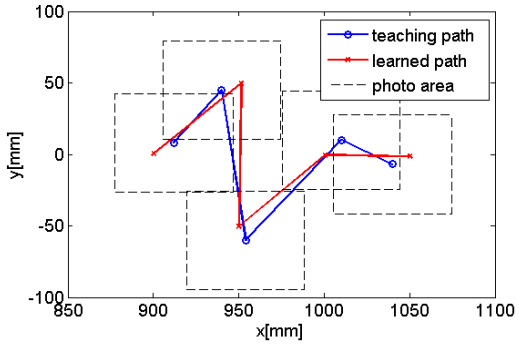

Figure 16 Iteration 1

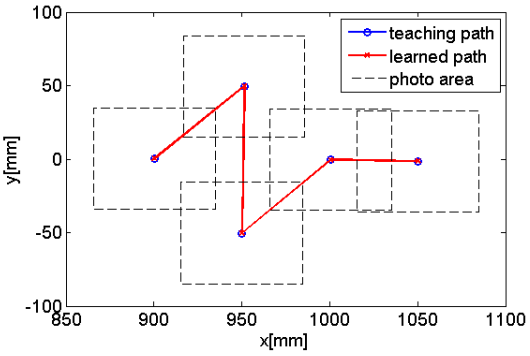

Figure 17 Iteration 2

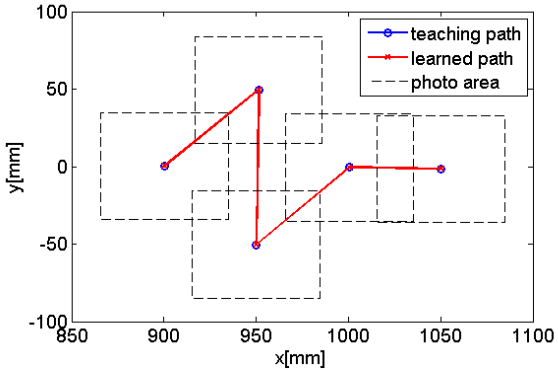

Figure 18 Iteration 3

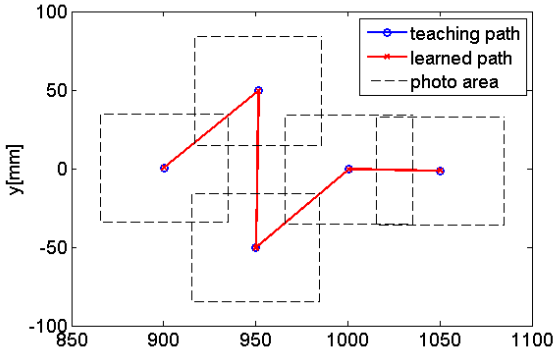

Figure 19 Iteration 4

As can be seen from the figures, almost no difference can be seen. The found error is calculated as$\epsilon = \sqrt{x^2 + y^2}$. The error for every point is shown in Figure 20. As can be seen, decreases the error to a maximum of 16 µm.

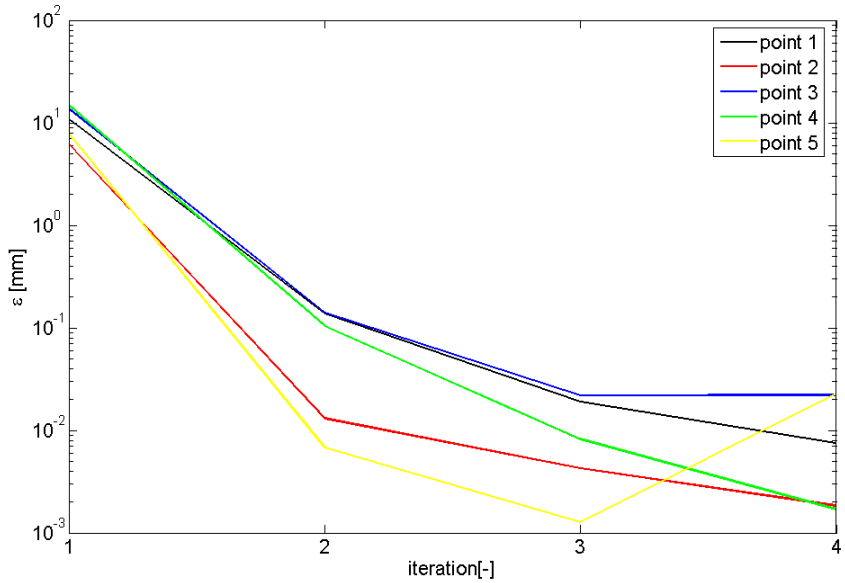

Figure 20 Decreasing error with more iterations steps

5 DEVELOPED SYSTEM

The goal of the software is to have a master which controls the slave and the robot. The slave is decided to be the computer controlling the camera via matlab. The master is the computer controlling the robot. The connection is made via 1 GB network via a TCP/IP connection. In this chapter the manuals of the software on both computers is presented.

5.1 SYSTEM START-UP

The following setup should be installed before using the system:

1. Mount the camera on the robot and connect the FireWire before starting up the connected computer. For using the laser the wires should be connected as shown in Figure 21.

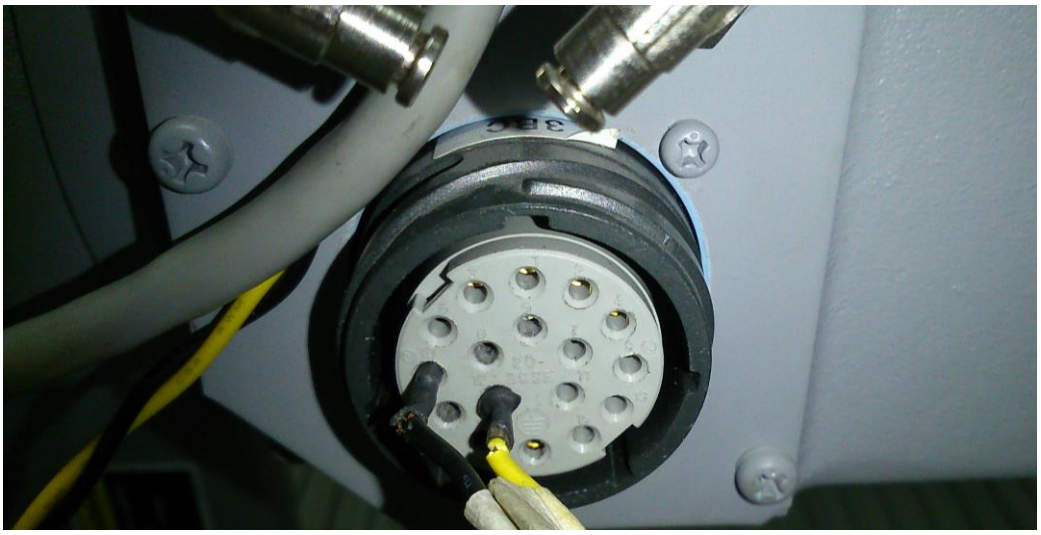

Figure 21 LASER wire connection

2. Make sure the hardware key on the USB is plugged in the Dell computer for control of the robot making use of the MOTOCOM32 library.
3. Start up the SUN computer connected to the camera and start up Matlab. The system is tested using Matlab 11 so it is recommended to use that version. Make sure the vision folder is the current directory in order to use its functions. The main file to be opened for the using the system is called VISION_MAIN_ERALD.m.
4. Start up the Dell computer and open Microsoft Visual Studio. The version used in the project in Microsoft Visual Studio 2008 so it is recommended to use that version. In the folder ERALD on the desktop, open the folder master and inside open the project ROBOT.sln.
5. For a clear and sharp image, the camera possibly needs to be adjusted manually. In order to get a preview the Matlab command *imaqtool* can be used.

5.2 MANUAL OF VISION SYSTEM AND MASTER CONNECTION ON SLAVE COMPUTER

The vision system has a main file VISION_MAIN_ERALD.m, which by running gives a choice to be operated manually or by the master via TCP/IP connection. On opening it gives a choice between 1: manual use and 2: use as slave by master via TCP/IP connection. All important variables are saved in the global structure config. The program has 5 options:

1) Calibration of camera

This option and function is not yet developed. It is put in for future applications where the camera calibration is done separate from the system calibration that is described in the next option. Therefore it can only be run in manual use calling the function CALIBRACION_CAMERA.m.

2) Calibration of system

This option makes use of synchronized communication between the two computers and can therefore only be run in use of slave by master. First the user specified amount of pictures are taken on the right positions by using the function CALIBRATION.m. This method is explained in chapter 4.2. After the procedure of taking pictures, the captured data is processed with the vision algorithm using the function CALIBRATION_SYSTEM.m. The calibration values as described in chapter 4.2 are saved in the global config.cam.k_u, config.cam.k_v, config.cam.alpha, config.cam.u0 and config.cam.v0. The structure is saved in a file “calibratie.mat” for future use. This is done so that when the calibration is done it does not have to be done again. This can only be used when the hardware is not adjusted in any way. In order to get a clear image for the calibration of the dots the following camera settings are used and set by the commands:

```
src.Brightness = 0;
src.Contrast = 22;
src.Exposure = -1;
```

For the localization of the laser cross, different settings are used. This is because the laser is light and the dots are dark spots. The used settings are given by the commands:

```
src.Brightness = -50;
src.Contrast = 50;
src.Exposure = -8;
```

3) Single image processing

This option can only be used in manual use. This option provides the means to take a snapshot and process the single image. The output is the position of the spot in pixels. The vector result gives first the vertical coordinate and secondly the horizontal coordinate.

4) Teach new path

This option makes use of synchronized communication between the two computers and can therefore only be run in use of slave by master. First it loads the config file containing the latest calibration values. The slave waits until a message is received from the master to have sent the robot to the right position. Then a command is given to give a snapshot. Then a command is send to the master that it can move the robot to the next position. After that the snapshot are processed using the PROCESS_IMAGE.m function. The result is directly in corrections to be made to the path. This process is repeated in a loop for every point. After the last point, the master computer sends a signal back confirming to be ready for receiving the correction and the corrections are send to the master computer. The whole process for all points can be repeated. In the test 4 iteration steps are done.

5) Stop

This option can be used by using slave mode and manual mode. The code is stopped. When using the slave mode, make sure to press enter after pressing the stop button in the master.

The full code with comments can be found in the files VISION_MAIN_ERALD.m CAPTURE_IMAGEN.m, PROCESS_IMAGE.m and TEACH.m. The files for the vision algorithm are developed by UASLP and can be found in folders Optim and LaserProy.

5.3 MANUAL OF ROBOT AND SLAVE CONTROL ON MASTER COMPUTER

The Motoman manuals suggest programming the robot using the MOTOCOM32.dll library in Microsoft Visual Studio 2008. Other options are not examined as the Motoman manual only gave an explanation of the implementation in this software environment. The main developed program has 7 header files. The most important is the MotoCom.h containing all possible functions incorporated by the NXC100 controller. The robotvE.h is a further developed version of the existing robotv5.h[2]. It has been extended with the network functions as explained in chapter 3. The rest is automatically created by the Microsoft Visual Studio software. The actual functions are found in robotvE.cpp. Every function has code to explain its functionality. There are a total of 4 source files with the main code of the different buttons in ROBOTDlg.cpp. The other 2 are automatically created again. The visual components are found in the resource files and the main menu is shown in Figure 22.

Figure 22 Main menu

1) MAIN FUNCTIONS

There are a total of 6 main functions. The HOME button moves the robot to the position (945.0, 0.0, 445.0, 180.0, -90.0, 0.0). The HOME3 button moves the robot to the position (1000.0, 0.0, 300.0, 180.0, -90.0, 0.0). This is for the system with a mounted camera a favourable position for

[2] Written by ING. GABRIEL FERNANDO GARCIA CEDILLO

the robot to move freely with the camera and laser pointing in vertical direction. The choice for using the definition of the TCP in the HOME position seems odd, but altering the position of the TCP resulted in more arithmetic errors of the robot. More on this is found in chapter 2.3. The third and fourth button are CONNECT and DISCONNECT which respectively connect the robot and turn on the servo motors or disconnect the robot and the servo motors. Then there are the buttons LASER ON and LASER OFF for turning on and off the laser cross. Make sure to first connect the robot.

2) MANUAL PLAY

The manual play is designed to move the robot without any due to the desired position. This can be done with MOVE TO POSITION. This option moves the robot to the given position with given speed. If speed is left to be less than 20mm/s, it is set to 20mm/s by default. The robot moves with linear movement to the position. For angular movement, take care because the speed is not defined for this. Also take care because Rx, Ry and Rz are rotations of the coordinate system and not of the robot. Large deviations from the current position will probably cause high speed movements which can damage the camera.

The robot can also be moved by using MOVE BY INCREMENT. This option moves the robot or coordinate system with the desired speed and increment.

The last option is to move the robot by using MOVE PULSES. In this option the position in degrees for every joint can be specified. The program will calculate the degree to pulses and move the joint to that specified pulse. For the last two options a minimal speed is set to 10mm/s and 5deg/s respectively.

CAUTION: Make sure to understand the robot and its reaction before putting in the numbers. And begin with low speed and small distances.

3) MAIN PROGRAM

The MAIN PROGRAM has 3 functions. For every option a connection is made with the slave sending the choice of the option to the slave. In the slave the choice is made automatically. First a choice has to be made in the master, then in the slave the user has to press a key. This is because the TCP/IP connection has to be opened in the master first. The first option is the calibration. The amount of layers k, directions n and used distance dx can be specified by the user and will be sent to the slave. Then the master sends the robot to a position, sends a signal to the slave to take a snapshot and waits for the signal from the slave that the picture is taken and the robot can move to the next location. This is repeated for every k and n and the robot is send to HOME3.

The second option is TEACH AND PLAY. The amount of used iterations is specified in the program code and is set to 4. The robot is send to the position by the master, and then a signal is send to the slave to take and process an image. When the snapshot is taken, the master sends the robot to the next position. When the slave is ready for the next step, a signal is send to the master to send the robot to the next point. This is repeated for all points. At the end the master sends the robot with the laser on to every point in order to show the found path. After the demo, the robot is send to HOME3.

The last option is STOP SLAVE. In this option the slave will be send a signal to stop the program. In the slave a key has to be pressed to actually finish the running code.

5.4 ERRORS AND BUGS

Sometimes the codes in C++ and/or Matlab are stuck in a loop due to several reasons as:

1) An arithmetic error in the robot control. This error is cause by sending wrong coordinates to the robot. This bug is taken out, but for manual use of the robot this can still be an issue.
2) The connection of the slave is first opened. Therefore there is no connection possible and between the systems and both programs keep on running.

To restart the system first the servos are shutdown with the emergency button on the teach panel as shown in Figure 23.

Figure 23 emergency button on teach panel

Then the teach panel is set to teach instead of remote by the switch on the teach panel as shown in Figure 24.

Figure 24 Mode switch on teach panel'

In the slave the Matlab session is shutdown using Ctrl-C. In the C++ session on the master the session is shut down using the stop button in the menu of Microsoft Visual Studio. In order to shut down the remote connection the menu shown in Figure 25 has to be shut down as well. The figure shows the status monitor with and without any communications. For safety the program is shutdown in errors.

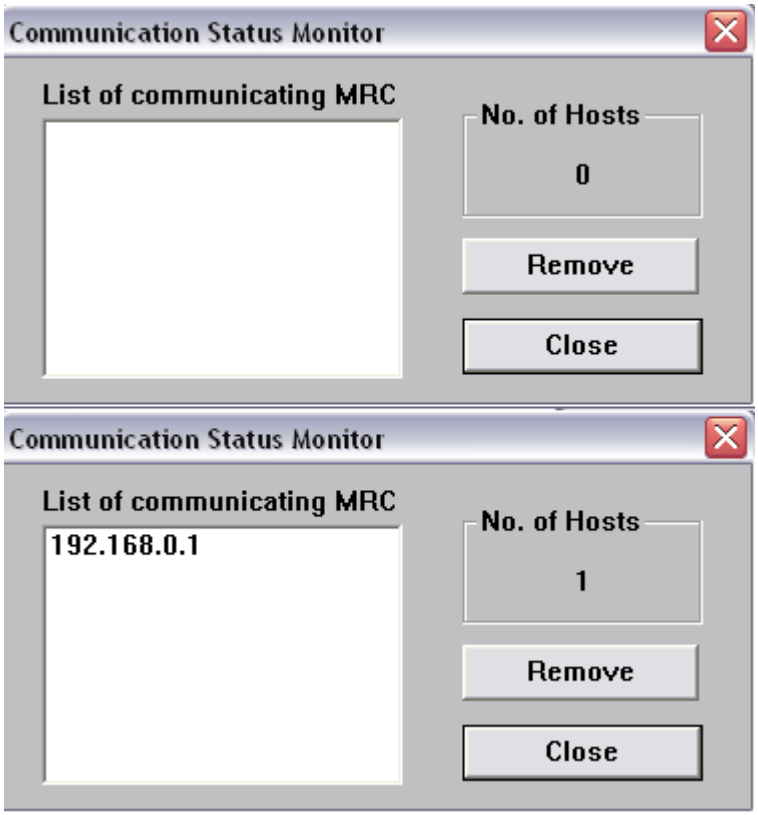

Figure 25 No communication with robot (up) and with communication with robot (down)

There is a last bug in the program on running the calibration menu. At finishing the following error is given as shown in Figure 26. There are no consequences to this error and the results appear to be in order but it is recommended to find the source of the error and solve it if the program is further used in research.

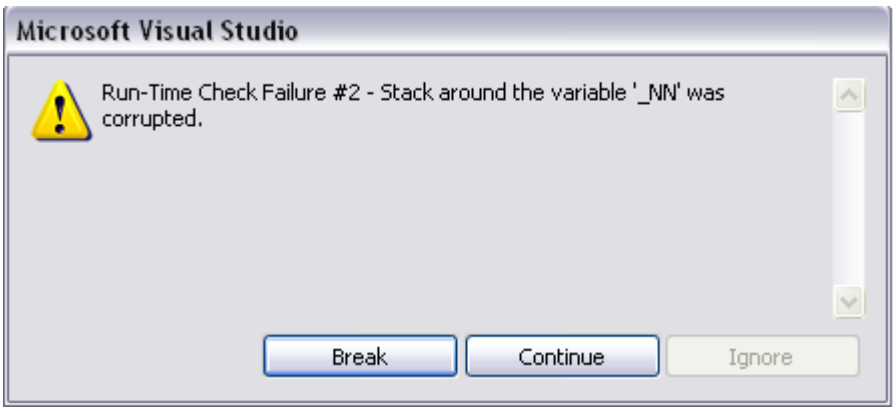

Figure 26 Error at finishing calibration method

6 CONCLUSION AND RECOMMENDATIONS

In the examination of the robot the following conclusions are drawn:

- The robot can be moved with linear movement and joint movement using MOVJ and MOVL.
- The robot has different control groups, BASE, ROBOT and USER and TCP and the movements can be defined from these coordinate systems.
- The robot can be moved in coordinates or in pulses using MOV or PMOV. The coordinates are in mm and are the place of the TCP. The rotations are the rotations of the axis in order of respectively x, y and z.
- While moving, the end coordinate cannot be altered. The command line is first executed and only after completion the robot can be send to a next position. This behaviour makes continuous processing for real-time corrections hard to realize. An implementation can be done for low speeds with the implementation of multiple threats in the main program.
- The given rotation movements are rotations of the coordinate systems defined in the TCP. The TCP coordinates can be read by the robot in the file TOOL.CND with the program called High Speed JobExchanger designed by MOTOMAN YAKASAWA.
- More on control of the robot can be found in the manuals of the robot and the used library as given in the used literature list in chapter 7 [2] [5] [1].

In the combination of the robot and the vision system the following conclusions are drawn:

- The vision system is used in an iterative learning process to send the TCP of the robot from point to point with an error ε of 16μm. This error is defined as the deviation of the found point from the learning path.
- The calibration is used for research and shows that if the point lies further away from the optical centre point, the cloud of points have a clear direction. In an area of 20mm by 20mm the deviation lies within 0.2mm. Because of the iteration steps the optical centre and the TCP lie close enough causing to be of no significant although its magnitude could be examined. This distortion can be taken out in the vision software in further research.
- The program designed for the teaching algorithm can be used for further research and can be used as a basis to develop further applications in practical use of the developed vision algorithm by the UASLP.
- The vision code and the master code used the x and y coordinates exactly the other way around. Also the y coordinate has to be extracted instead of added. Care should be taken by the user of the code about this.

Always should the user take care of how to use the software. Sending the robot manually could result in fast movement and destruction of the attached camera or nearby hardware. Always make sure that surrounding people are not within reach of the robot.

7 USED LITERATURE

[1] Y. E. CORPORATIONS, “MOTOCOM32 OPERATION MANUAL,” Japan, 2008 08-03.

[2] Y. E. CORPORATION, "NX100 INSTRUCTIONS," Japan, 2007.

[3] M. A. t. p. o. e. a. science, MathWorks, [Online]. Available: http://www.mathworks.nl/products/matlab/. [Accessed 14 Februari 2014].

[4] Microsoft, “Microsoft Developer Network,” Microsoft , [Online]. Available: http://msdn.microsoft.com/en-US/#fbid=XxAPw0zP4TS. [Accessed 14 Februari 2014].

[5] Y. E. CORPORATION, “NX100 OPERATOR’S MANUAL,” Japan, 2008 03-10.