Overhead Localization of Mobile Robots

Mikael Borg

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--5914--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2013

Abstract

The objective of this thesis has been to develop a system that detects a mobile robot with tracking and controlling its position with the help of one or more cameras. This has been done by implementing different algorithms that are able to detects markers. By using several cameras one can track over a bigger area and estimate the markers position with better precision by using triangulation where the marker has been detected by at least two cameras. A tracking marker is mounted on a robot which enables the camera to detect the robot and a reference marker is placed on the ground in order for the mobile robot to move towards the reference marker. When the tracking and reference marker has been detected the direct commands are then sent to the mobile robot for controlling its position. Topics that are of interest in this report are image analysis in general, image analysis algorithms, camera calibration, different libraries for implementation, Lego mindstorms NXT, sockets TCP/UDP, bluetooth.

Acknowledgements

This thesis has been done at the Department of Automatic Control, LTH, Lund University. I would like to thank my supervisor Vladimeros Vladimerou, with his patience and guidance and my examiner Anders Robertson who gave me this opportunity to work with this wonderful thesis. I would also like to thank my wife and family and especially my father who inspired me all my life and taught me to never give up.

Contents

1. Introduction

This chapter describes the problem to be solved and analyzes its feasibility.

1.1 Background

Computer Vision is an area that constantly gains more ground on the market and its variety of applications is immense. With the development of Computer Vision the last two decades a new and cheap sensor has been introduced, namely the ccd-camera. Tailor-made software can be made to fit various environments, and the speed of the camera as a sensor lies within its algorithms. In this thesis several solutions will be presented on how tracking system can be designed.

1.2 Problem Formulation

The main objective of this thesis was, with the help of one or several cameras, to track a robot in real-time. To detect the robot, a recognition or identification algorithm had to be constructed.

2. Expermintal Setup

This section describes the hardware and tools used during the thesis.

2.1 Setup

By using multiple cameras one can cover a broader area of tracking and achieve a more robust tracking system, since a camera is limited by its physical parameters such as focal length and resolution. The cameras were mounted according to Figures 2.1 and 2.2.

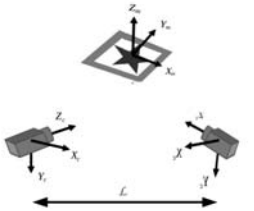

Figure 2.1 Camera setup for detecting the marker

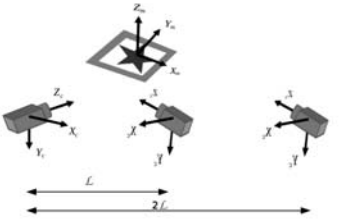

Figure 2.2 Camera setup for detecting the marker

2.2 Hardware

Cameras

The following cameras where used, Logitech Quickcam pro 9000 [21] and Logitech C250 [22] with a usb-connection, and a marlin camera by Allied vision technologies [23] with a fire-wire connection.

Figure 2.3 Quickcam pro 900 (*left*) [21], C250 (*center*) [22], marlin (*right*) [23]

During the early stages of this thesis the marlin camera was used with the high level API libdc1394 [4] which was later on discarded due to integration problems with OpenCV and ARToolKit. The Quickcam pro 9000 can manage up to 1600x1200 pixels and C250 up to 800x600 pixels. The Quickcam pro was considered inferior compared to C250 since it has an embedded auto focus function which made it very hard to calibrate and didn't give satisfactory results.

Lego mindstorms NXT

For the tracking of a robot a lego mindstorms NXT[9] was used which is a programmable robotic kit. The interface to the NXT is either usb or bluetooth and it can be programmed with several types of languages such as for example Not exactly C, LeJOS NXJ [10] , Matlab. For this thesis the NXT was flashed with LeJOS NXJ and its firmware. Omni wheels where attached with RJ12 connection cables to the nxt brick.

Figure 2.4 (*left*) NXT brick CPU [9], (*right*) Omni wheels mounted on the NXT [20]

2.3 Tools

libdc1394

Libdc1394 is a library that provides the user with communications for fire wire cameras. Libdc-1394 [4] has a high level programming interface that adapts to the 1394-based Digital Camera Specification.

OpenCV

OpenCV [6] is a computer vision designed library which mainly focus on real-time processing. It has many embedded functions which makes it easy for the user to write his/her own computer visions programs.

ARToolKit

ARToolKit [5] (*where AR stands for Augmented Reality*) is a computer vision library tracking library that allows the user to create augmented reality applications. The augmented reality is then drawn by OpenGL [11] that overlays the virtual image on the real world image.

3. Methodology

In this section the different methods that were used through out the project will be presented.

3.1 Libdc1394 with OpenCV

OpenCV provides a lot of useful functions but it lacks the high level programming interface for 1394-cameras, and therefore a merge between libdc1394 and OpenCV was necessary. OpenCV has its own frame capturing functions for 1394-cameras but it lacks any control at all over the camera such as buffers and camera parameters. The programs were constructed in such a way that libdc1394 handled all the communication with the camera and OpenCV did the drawing of the GUI and all the calculations. From this platform different algorithms were tested to detect a marker.

3.2 Camera Model

The pinhole camera model [2, *Ch.*2] was used during this thesis, which describes the mathematical relationship of a given coordinate $\mathbf{R^3}$ in space and its projection onto the image plane $\mathbf{R^2}$ of an ideal pinhole camera. The pinhole camera is within itself a simple camera without a lens. The incoming arrays of light pass through a single point and are then projected as an inverted image on the image plane. The model neglects geometric and illumination distortions and blurring of unfocused objects caused by the lens. Due to the fact that the model neglects many of the the physical disturbances one can only approximate it to the first order when mapping the coordinates from $\mathbf{R^3}$ in space to $\mathbf{R^2}$ into the image plane. Meaning that in order to get a good approximation one is very dependent of the quality of the hardware, i.e., the camera. It will be shown it the thesis that the results is dependent of the quality of the camera and the illumination of the testing environment

3.3 Image Analysis

Edge- and Corner-detection are tools that are widely used within image analysis. They both attempt to capture significant properties of objects in the image. These properties that are captured include different discontinuities in the photo metrical, geometrical and physical characteristic of the object. The information that is captured will give rise to variations in the gray scaled image; discontinuities such as step edges and local extrema (line edges), and 2D features from where edges meet (junctions).

The purpose of edge- and corner-detection is to identify these variations. Almost all of the detection algorithms are based on some kind of convolution with a digital image. For simplicity, the digital image is converted to a mono8 image with a value range of 0-255. To make the algorithms more robust, image derivatives are computed. However the derivatives are sensitive to various sources such as noise, e.g., electronics, discretization/quantification effects. To solve this, several of the algorithms use a pre-filtering with smoothing.

Edge-Detection

Edge-detection can be grouped into two categories: Search-based and zero-crossing based. Search-based methods detect edges by first computing the first-order derivative i.e., the magnitude of the gradient and then locate the local directional maxima of the gradients magnitude. The zero-crossing based methods search for the zero crossings in a second-order derivative expression which is computed from the image. The most commonly scheme for a simple edge detection can be divided into three steps:

- **Differentiation** evaluates the derivatives.
- **Smoothing** reduces the noise.
- **Labeling** localizes the edges and increases the SNR-ratio of the edge by surpassing false edges.

Two edge detection algorithms where tested during this thesis

The Sobel Operator uses a discrete differentiation operator which computes an approximation of the gradient in the intensity function of the image. In each pixel of the image the result of the Sobel operator is attained, giving the possible largest direction in the variation of the gray-scaled image and the rate of change in that direction. The result is achieved by convolving the image with a small 3x3 kernel which makes the Sobel operator the fastest edge detection there is. If the image was to be represented by matrix $\mathbf{A}$ and horizontal and vertical derivative approximations be $\mathbf{G}_x$ respectively $\mathbf{G}_y$. Then the computation are as follows [1]:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A \qquad G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{3.1}$$

Where * denotes two dimensional convolution. In each point of the image, the resulting gradient approximation is given by taking the magnitude G of (G_x,G_y)

$$G = \sqrt{(G_x^2 + G_y^2)} \tag{3.2}$$

Using this information one is able to calculate the gradient direction

$$\phi = arctan(\frac{G_y}{G_x}) \tag{3.3}$$

Canny Edge Detection is one of the most used edge detection due to its adaptability to various environments. The variable parameters makes the user very flexible allowing them to choose the environment conditions. It is a robust algorithm which has a low rate of detecting false corners. This edge detection works in several stages [17].

- **Gaussian Filtering** is applied to the gray scaled image to reduce the noise. The raw image is convolved by a Gaussian filter. The effect of the convolution will slightly blur the image. A typical filter that is used is a 5x5 kernel with $\sigma = 1.4$:

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A$$

- **Differentiation** of the gradient in the image. By taking the directional derivatives with the help of differentiation operators as in [17]. The result is given by Equations 3.2 and 3.3.
- **Non-maximum Suppression** estimates the image gradient. This procedure is used to trace along the edge in the edge direction and it suppresses any pixel value that is not considered to be an edge.
- **Hysteresis** with thresholding is applied to determine the strength of an edge. By having two thresholds high and low; one eliminates the possible corruption of noise in the edges.

Corner Detection

Corner detection can be derived from edge detection, and it has the same structure in many ways but the algorithms are more complex and the computational time is longer. A corner can be defined as an interest point, that is where two dominant edges meet in a local neighborhood of the point. In general most corner detection methods detect interest points rather than corners. An interest point is a point in an image which has a defined position and can be robustly detected. Hence an interest point can also be a isolated point of local intensity maximum or minimum, line endings, or even a point of a curve where the curvature is locally maximum/minimum.

Harris Corner Detection is based on the Movarec's corner detection. Harris corner detector [18] uses a local auto-correlation function of a signal; where the auto-correlation function measures the local changes

of the signal with the help of shifted patches. Given a shift $(\Delta x, \Delta y)$ and a point (x, y), the auto-correlation function can be defined as:

$$c(x, y) = \sum_W [I(x_i, y_i) - I(x_i + \Delta x, y_i + \Delta y)]^2 \tag{3.4}$$

where $I(\cdot, \cdot)$ denotes the image function and (x_i, y_i) are the points in the window W centered at (x, y). By approximating the shifted image with Taylor expansion of the first order:

$$I(x_i + \Delta x, y_i + \Delta y)] \approx I(x_i, y_i) + [I_x(x_i, y_i) I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.5}$$

where $I_x(\cdot, \cdot)$ and $I_y(\cdot, \cdot)$ denote the partial derivatives. Substituting Equation 3.4 into 3.5 yields,

$$= \sum_W ([I(x_i, y_i) - I(x_i, y_i) + [I_x(x_i, y_i) I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}) \tag{3.6}$$

$$= \sum_W (-[I_x(x_i, y_i) I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}) \tag{3.7}$$

$$= \sum_W ([I_x(x_i, y_i) I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}) \tag{3.8}$$

$$= [\, \Delta x \, \Delta y \,] \begin{bmatrix} \sum_W (I_x(x_{i,y\,i}))^2 & \sum_W I_x(x_{i,y\,i}) I_y(x_{i,y\,i}) \\ \sum_W I_x(x_{i,y\,i}) I_y(x_{i,y\,i}) & \sum_W (I_y(x_{i,y\,i}))^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.9}$$

$$= [\, \Delta x \, \Delta y \,] \, C(x, y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.10}$$

where $C(x, y)$ represents the intensity structure of the local neighborhood. Let λ_1, λ_2 be the eigenvalues of the matrix $C(x, y)$.

- If both λ_1, λ_2 are small, then the windowed image has an approximately constant intensity
- If one eigenvalue is large and the other one small, then auto-correlation only shifts in one direction thus giving little change in $C(x, y)$ but significant change in the orthogonal direction; this indicates an edge.
- If both eigenvalues are large, then shifts in any directions will result in significant increases; this indicates a corner.

Morphological corner detection performs a dilation of an image with a given structuring element followed by an erosion followed by another structuring element[16]. The idea is to make dilation and erosion complementary in terms of the type of corners they affect. This can be realized by choosing a cross as the first structuring element and a lozeng[1] as the second, where the mentioned structuring elements can have the appearance of

$$+ = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \diamond = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.11}$$

Let $\mathbf{I}$ represent the image and $\mathbf{c}_+$ and $\mathbf{c}_\times$ are the complementary operators. The combination of the two operators will make the corner detection almost rotationally invariant. By combining the two operators one gets the following operator [16] :

$$\mathbf{c}_{+,\times}\,\mathbf{I} = |\mathbf{I}^c_{+,\diamond} - \mathbf{I}^c_{\times,\square}\,| \tag{3.12}$$

The complementary operators are convolved over the pixels in the image.

3.4 ARToolkit

ARToolkit [5] allows the user to draw virtual imagery on a live video of the real world, by sending frames to the application; frames that have been captured by the camera to the AR application. The application then searches each frame after any square shape as shown in Figure 3.1. If a square is found, the application then calculates the position of the camera relative to the marker. The computer graphics model can then be drawn once the position is known. The drawn model is then placed on top of the real world stream, which makes the drawn model appear to be stuck on the marker.

Figure 3.1 The patterns [5] that was used for tracking with ARToolkit.

[1] a rhombus or diamond shape

ARToolkits Vision Algorithm

For the detection of the marker ARToolKit converts the frame into a binary image which is based on a threshold value. The image is then searched for square regions. ARToolKit then finds all the squares in the binary images. For each square it finds ARToolkit matches it with a pre-attained pattern template. If the square matches the pattern template; a known pattern has been found. It then calculates the position of the real video camera relative to the marker with the help of the marker's orientation [5]. The detection procedure for ARToolkit is shown in Figure 3.2-3.7.

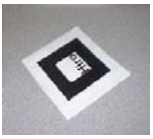

Figure 3.2 Original image [5]

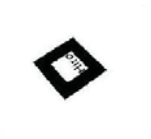

Figure 3.3 Thresholded image [5]

Figure 3.4 Connected components [5]

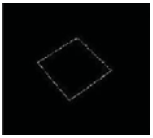

Figure 3.5 Contours [5]

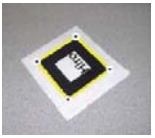

Figure 3.6 Extracted edges and corners [5]

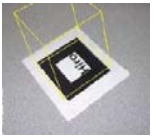

Figure 3.7 square fitted [5] according to Fig. 3.6

A more detailed description is shown in Figure 3.8 below.

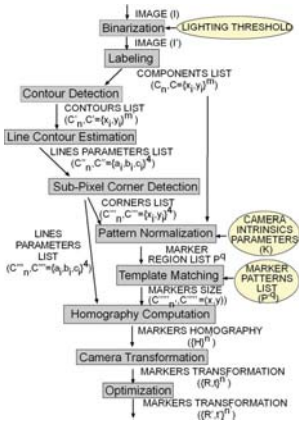

Figure 3.8 Flow chart over the detection algorithm [5]

ARToolkit's Coordinate system

The marker has its own coordinate system with respect to the cameras coordinates, which is used in order to get the orientation of the projection onto the image plane. ARToolkit computes the rotation matrix $R_{3\times3}$ and the translation matrix $T_{3\times1}$ as seen in Equation 3.13.

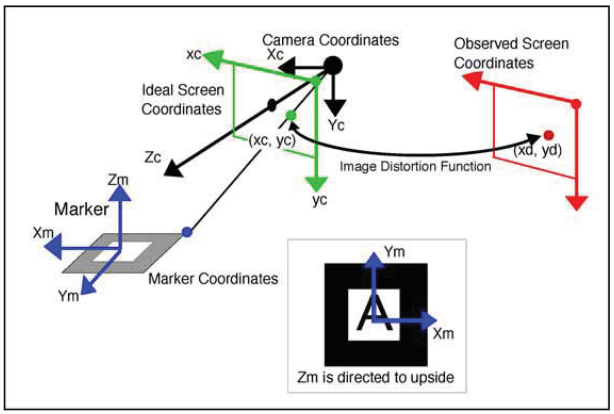

Figure 3.9 ARToolkit coordinate system [5]

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{m11} & r_{m12} & r_{m13} & t_{mx} \\ r_{m21} & r_{m22} & r_{m23} & t_{my} \\ r_{m31} & r_{m32} & r_{m33} & t_{mz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3\times3} & T_{3\times1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{3.13}$$

Where
$[r_{m11} \quad r_{m12} \quad r_{m13}]$ coordinate of X_c relatively to the marker axes
$[r_{m21} \quad r_{m22} \quad r_{m23}]$ coordinate of Y_c relatively to the marker axes
$[r_{m31} \quad r_{m32} \quad r_{m33}]$ coordinate of Z_c relatively to the marker axes
$[t_{mx} \quad t_{my} \quad t_{mz}]$ camera center position relatively to marker axes

Glut and OpenGL

OpenGL is used for the rendering and drawing within ARToolkit, where Glut is used as an API for implementation. When a pattern has successfully been detected ARToolkit calls functions from Glut to render the captured frame.

Figure 3.10 Glut rendering a cube after detected a marker

3.5 Calibration

Camera calibration is needed to link the known coordinates of a set of 3-D points and their projections, and for solving the camera parameters also known as the intrinsic and extrinsic parameters. In order to access the coordinates of a given 3-D point, camera calibration methods use several images of a calibration pattern.

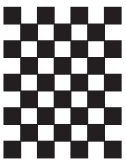

Figure 3.11 Typical calibration pattern

The intrinsic parameters are needed to represent a point in space which is measured in a coordinate system fixed to the camera. The parameters can be defined as a set of parameters that are needed to describe the optical, geometrical and digital characteristic of the viewing camera. The parameters can be specified respectively:

- the perspective projection that is given by the focal length f.
- the transformation between the camera frame coordinates and pixel coordinates, which is described by the center in pixel coordinates (σ_x, σ_y) and the effective pixel size in horizontal and vertical direction (s_x, s_y).
- the geometric distortion which is given by the radial distortion coefficient k_1.

The extrinsic parameters specify the transformation between the camera and the world reference frame as shown in Figure 3.12. The transformation is calculated with the translation vector $\mathbf{T}$ and the rotation matrix $\mathbf{R}$, which are described by the following:

- a 3-D translation vector $\mathbf{T}$, that describes the relative position of the origins of the two reference frames
- a 3×3 rotation matrix $\mathbf{R}$, that brings the corresponding axes of the two frames on each other.

The relationship between a point P in the world and camera frame, P_w and P_c respectively, are

$$\mathbf{P}_c = R(\mathbf{P}_w - \mathbf{T}) \tag{3.14}$$

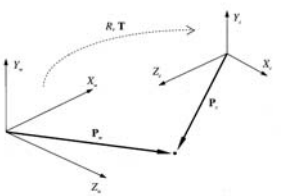

Figure 3.12 The relationship between camera and world coordinate.

with

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.15}$$

3.6 Sockets

Since this thesis required more than one camera for detecting a marker, there was a need for a server to process the data. Sockets [8] where used for endpoint network communication using UDP transmission. First

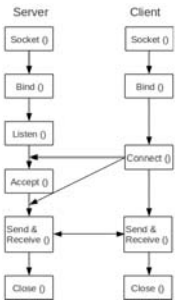

Figure 3.13 Server and client flowchart

the *socket()* API is used for creating a socket with a descriptor and an endpoint communication. Then when an application has its socket descriptor it can be used to bind a unique name to the socket. The server then binds a name to be accessible through the network. The *listen()* API initiates a scan for other client's connection requests. A client then casts a *connect()* command to establish a connection with the server. Then the server uses *accept()* to accept the client's request. Once these procedures are completed the *send()* and *receive()* can be used to exchange data. The close() API is called when one wants to end the transmission. See Fig 3.13 for a description of the connection procedure.

3.7 Bluetooth communication protocol NXT

The communication with the NXT brick was done by using its bluetooth interface [9] and sending direct commands to it.

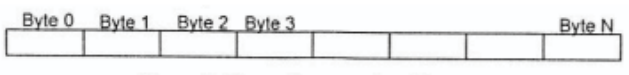

Figure 3.14 General protocol architecture [9]

The direct commands are implemented by using the assigned addresses as shown below.

0x00: Direct command telegram, response required.

0x01: System command telegram, response required.

0x02: Reply telegram.

0x80: Direct command telegram, no response.

0x81: System command telegram, no response.

The bluetooth packages looks at follow[9]:

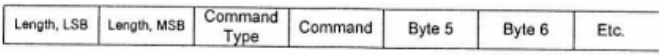

Figure 3.15 Structure of the bluetooth package [9]

where a direct command **SETOUTPUTSTATE**[9] was sent to the NXT:

Byte 0: 0x00 or 0x80

Byte 1: 0x04

Byte 2: Output port (Range: 0-2; 0xFF is special value meaning 'all' for simple control purpose)

Byte 3: Power set point (Range:-100-100)

Byte 4: Mode byte(Bit-field)

Byte 5: Regulation mode (UBYTE:enumerated)

Byte 6: Turn Ratio (SBYTE:-100-100)

Byte 7: RunState (UBYTE:enumerated)

Byte 8-12: TachoLimit (ULONG: 0:run for ever)

4. Results

Libdc1394 and OpenCV

Initially it was my intention to merge libdc1394 with OpenCV in order to achieve a satisfactory tracking program with a fire-wire camera. The API that followed with Libdc1394 was not for someone who hadn't any prior knowledge within the field. I managed to implement a lot of the algorithms that is described in Chapter 3, with the OpenCV API for calculus. My results were varying and demanded a lot of CPU for the real time calculations. The choice of marker was a big topic; how to design a marker that is easily detected and not too complex for the algorithm? I chose a simple black and white checkers pattern for its simplicity and its contrasts which optimizes the detection and minimize the illumination disturbances.

Out of all the edge- and corner detection algorithm that are presented in this thesis the sobel operator and canny edge detector were proven to be the most robust detection methods. The computation time for sobel operator and canny edge detection was to slow to be considered as a real time system. After spending a lot of time and effort with the fire-wire solution I decided to test a USB camera and use OpenCV for capturing frames and to skip libdc1394, since it was only compatible with fire-wire. My main focus was to improve the detection algorithm and its performance by changing the processed frame to a mono8 resolution by 0-256 compared to a rgb which has $0\text{-}256^3$, gave noticeable enhancement in the performance and the suppression of disturbances. By making some minor changes with the *gaussian filter* when using the canny edge detector the performance improved a lot.

ARToolKit and NXT

After a lot of time spent trying to construct my own computer vision algorithm I discovered ARToolkit, that had built-in detection functions, a lot of them which I already had tried in some way. Most importantly the algorithms were not that CPU demanding as mine were. The simplicity lies within its detection method, a mono8 frame is captured then thresholded and binarized and the corners and edges are easily detected after this procedure. It makes it very robust and performance efficient if you neglect the disturbances. My new goal became to merge ARToolkit with OpenCV using multiple cameras. The procedure that followed was to initially mount two cameras to detect one marker according to Figure 2.1, where one camera would be used as reference, meaning if the marker went out of sight for the reference camera it would hopefully be detected by the second camera. The coordinates of the marker in the second camera were to be translated into the reference camera coordinates. In order for this setup to work I had to calibrate each camera individually to acquire the intrinsic parameters. After that procedure

I had to calibrate them for stereopsis usage. This was done by using *Matlab Camera Calibration Toolbox* [12] where the frames from each camera were captured by a simple OpenCV script.

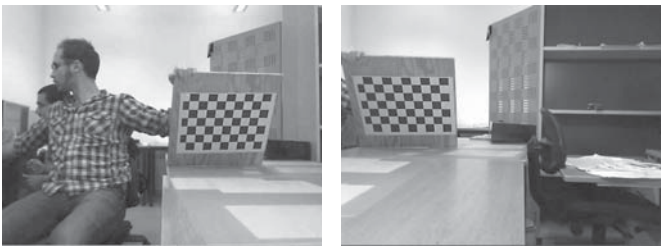

Figure 4.1 Calibration frames taken by each camera

Once the sufficient number of frames had been gathered they are then uploaded into the *Matlab Camera Calibration Toolbox*. The next procedure is to decide the orientation. This was done manually by marking the corners in the checkers pattern in the exact order for each frame, which is illustrated in Figure 4.2.

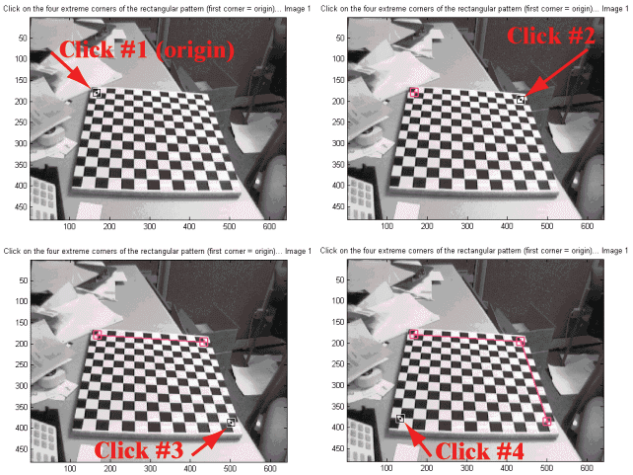

Figure 4.2 Marking the orientation with the toolbox [12]

The toolbox then calculates the intrinsic parameters such as *focal length*, *principal point*, *skew*, *pixel error*. A m-file containing the calibration result is generated by the toolbox.

```
%-- Focal length:
fc = [ 673.332617656223306 ; 675.469422332000249 ];
%-- Principal point:
cc = [ 352.675399053343085 ; 223.126470067957825 ];
%-- Skew coefficient:
alpha_c = 0.000000000000000;
```

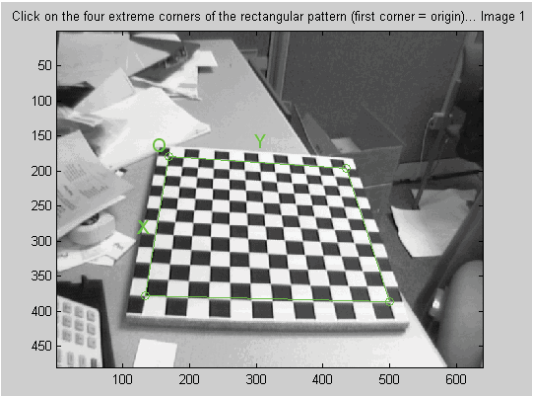

Figure 4.3 Processed frame after the orientation has been decided by Matlab Calibration Toolbox [12]

```
%-- Distortion coefficients:
kc = [ 0.181190418032493 ; -0.453522384855907 ; -0.000121406579967 ; -0.002780710618230 ; 0.000000000000000 ];
%-- Focal length uncertainty:
fc_error = [ 12.345386682495805 ; 11.805187092744728 ];
%-- Principal point uncertainty:
cc_error = [ 11.591420612578597 ; 7.841410893477586 ];
%-- Skew coefficient uncertainty:
alpha_c_error = 0.000000000000000;
%-- Distortion coefficients uncertainty:
kc_error = [ 0.078385534079341 ; 0.250001823025222 ; 0.004312853578146 ; 0.010190477067105 ; 0.000000000000000 ];
```

Since my goal was to have three cameras running simultaneously I focused on to achieve the same result as I would have got with two cameras, meaning I would just add the third camera to the system once I had two cameras working. After the calibration of the intrinsic parameters for each camera a calibration with stereopsis was done in order to get the rotation matrix and the translation matrix between the two cameras. This can be seen in Figure 4.5.

Figure 4.4 The effect of a poorly calibrated camera that can be seen on the cube that OpenGL draws; the lines are not perpendicular to each other

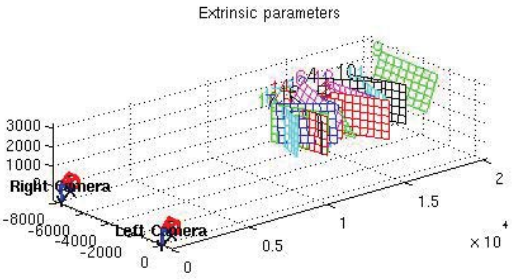

Figure 4.5 Stereo calibration where captured frames can be viewed

With the help of the *Essential matrix*, $E_{4\times4}$ (which contains the rotation matrix and translation vector) I could translate the marker's coordinates to a world reference camera coordinate system.

Once the calibration was done with the two cameras my goal was to construct a server that could manage the data from each camera. It was a time consuming process since I had no knowledge of how to implement a server using a socket solution. I tried both TCP e.g. *blocking* and UDP e.g. *nonblocking* sockets and my experience was that it required a lot more programming witch TCP due to the synchronization that is needed between the packages. Since I had a sampling of about 26 frames per second for each camera and a stream that was constant, there was a reason to not use TCP, meaning if a packet was lost during the transmission from client to server it would not have any big impact on the system.

By integrating the computation on-line within the server program I could calculate the translated coordinates to the reference camera on-line. The results didn't coincide to what I had anticipated. All the numeric values were completely off chart; the rotation matrix is quite complex within it self and for the untrained eye even harder. Even the most important parameters t_x, t_y and t_z were not even close. With the systems behavior it was a major setback since in my naivety I thought it would work without any problems. Without spending too much time on this problem I kept looking for new solutions.

By reading the article Activity Replay System of Life Review Therapy

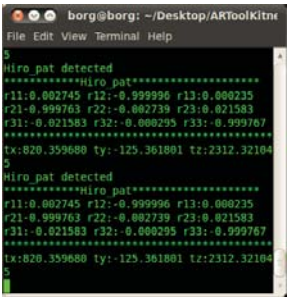

Figure 4.6 The clients terminal window with the rotation and translation printed

Using Mixed Reality Technology [14] and combining that knowledge with High precision camera calibration in vision measurement combining [15] took my thesis into a new path, automating the calibration for the essential matrix. I started off by making it possible for the user to parse commands to the server application; one of them calibration mode. The server application bulks out data to an m-file containing the essential matrix for each camera with 500 data points. Since the server application is built on UDP sockets the packages will not be synchronized, that did not have any effect on the results since the detected marker is fixed when running this option. *Datapoints selected to be gathered between camera 1 and camera 0*

```
x70{500}=[ -422.783610 -22.376211 -11.153704];
Cam0{277}=[ -0.023132 0.988597 0.148799 197.615005 ;
     0.999716 0.023740 -0.002315 162.284912 ;
    -0.005821 0.148703 -0.988865 993.472900 ;
     0        0        0         1];
Cam1{1}=[ 0.039589 -0.998035 -0.048569 -466.650299 ;
 -0.996854 -0.042789 0.066722 105.619263 ;
 -0.068670 0.045775 -0.996589 1026.066650 ;
  0        0        0         1];
x{1}=[ 0.923378 -0.047199 -0.380980 669.424201 ;
       0.075776 0.995297 0.060353 -98.974785 ;
       0.376339 -0.084598 0.922611 -21.587476 ;
       0        0        0         1];
indexes{1}= [  97 1 ];
::
Cam0{860}=[ -0.025727 0.989239 0.144029 198.511185 ;
     0.999651 0.026324 -0.002240 163.019485 ;
    -0.006008 0.143921 -0.989571 998.185547 ;
     0        0        0         1];
Cam1{591}=[ -0.038071 0.998417 -0.041408 -467.600677 ;
     0.996589 0.040973 0.071633 105.850342 ;
     0.073217 -0.038540 -0.996571 1028.211304 ;
     0        0        0         1];
x{500}=[  0.982688 0.025210 -0.183544 844.070306 ;
 -0.011683 0.997159 0.074409 -24.501607 ;
  0.184898 -0.070977 0.980191 84.313360 ;
  0        0        0         1];
indexes{500}= [  860 591 ];
tempString='Camera1';
```

Datapoints selected to be gathered between camera 2 and camera 0

```
Cam0{97}=[ -0.995126 -0.002723 -0.098578 153.771286 ;
    0.010722 0.990706 -0.135595 2.234667 ;
    0.098031 -0.135991 -0.985848 818.526733 ;
    0        0        0         1];
Cam2{1}=[ -0.881172 0.021379 -0.472313 -152.305084 ;
   0.049347 0.997680 -0.046904 53.999798 ;
   0.470214 -0.064638 -0.880182 977.660645 ;
   0        0        0         1];
x{1}=[ 0.923378 -0.047199 -0.380980 669.424201 ;
       0.075776 0.995297 0.060353 -98.974785 ;
       0.376339 -0.084598 0.922611 -21.587476 ;
       0        0        0         1];
indexes{1}= [  97 1 ];
::
Cam0{742}=[ -0.994945 -0.004067 -0.100335 153.816528 ;
     0.009818 0.990452 -0.137508 2.212940 ;
     0.099936 -0.137798 -0.985406 818.891602 ;
     0        0        0         1];
Cam2{1105}=[ -0.881172 0.021379 -0.472313 -152.305084 ;
       0.049347 0.997680 -0.046904 53.999798 ;
      0.470214 -0.064638 -0.880182 977.660645 ;
      0        0        0         1];
x{500}=[ 0.924020 -0.048449 -0.379262 667.955100 ;
 0.077470 0.995088 0.061628 -99.973941 ;
 0.374413 -0.086327 0.923235 -22.032014 ;
 0        0        0         1];
indexes{500}= [  742 1105 ];
```

```
tempString='Camera2';
```

The Cam0 is the reference camera, $indexes(i)$ the cameras packages number when it has been successfully transmitted to the server and $x(i)$ the inversion of the essential matrix from the translating camera. The server application will save the m-file with filename of CalibCam0toCamX.m where the user parses which camera number to use. Cam0 will always serve as the reference camera. The data points were then easily imported into matlab where off-line calculus where made in order to get the rotation matrix and translation matrix between the two cameras. This was done with the help of Eq. (20) in [15]. For more details on how this was done I refer to the matlab script in *Appendix E*. The script outputs a .txt file that can be imported into the server application. The calibration mode that implemented in my server does not require any manual action as in the toolbox provided by Matlab, the procedure is automated where the user places a marker in camera view and the data points are gathered. Once these functions worked properly I had managed to overcome a major milestone. Next to follow was the evaluation of the system. By having the cameras at fixed positions and then detecting two markers also at a fixed distance.

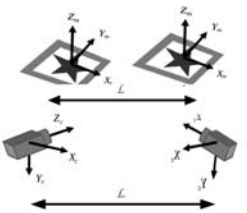

Figure 4.7 Performance evaluation setup

By varying the distance between the markers and cameras and the distance between cameras I gathered the corresponding data points.

There were 500 data points gathered at each distance with an incrementing step of 10 cm. Notice that I choose to just focus on the translation vector since it is a lot more transparent than the rotation matrix.

```
x70{1}=[ -421.338812 -22.255979 -11.513136];
⋮
x70{500}=[ -422.783610 -22.376211 -11.153704];
```

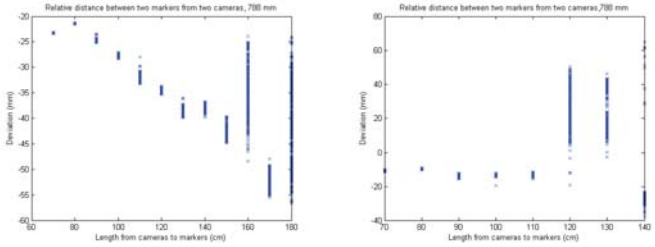

Figure 4.8 Distance between the markers 510 mm, small marker (*left*), Distance between the markers 437 mm, big marker(*right*)

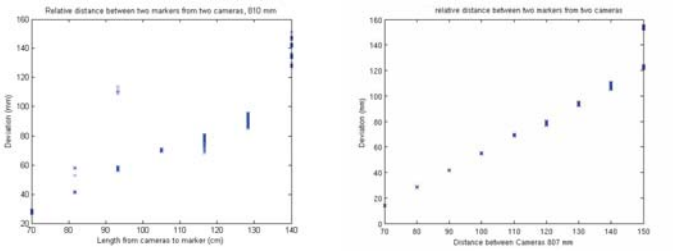

Figure 4.9 Distance between the markers 510 mm, big marker (*left*), Distance between the markers 510 mm, big marker (*right*)

The difference between the left and right plot in Figures 4.9 and 4.10 is that in the center one I have used a threshold filter with the intention to improve the detection performance and hence the accuracy.

By making some minor changes I could easily add the third camera to the system. The protecting cover on the C250 web cams were dismounted for placing the ccd camera chip on a rail with a distance of 1400 mm between each camera. Just as seen in Figure 2.2 the rail which contains the cameras, was mounted in the ceiling for the tracking to be overhead. The marker were then tracked in a plane perpendicular to the cameras. This new setup meant I had to calibrate the cameras again, following the same procedure as mentioned earlier.

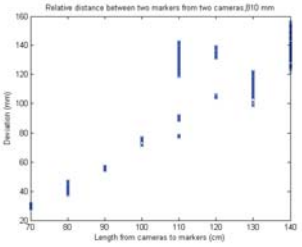

Figure 4.10 Distance between the markers 450 mm, small marker

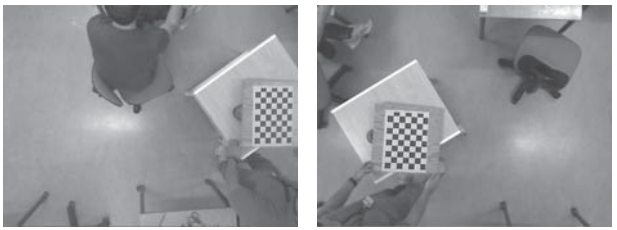

Figure 4.11 (*left*) Camera 1, (*right*) Camera 0 (the reference camera)

The system with three cameras behaved as I expected with the same performance as shown above. By integrating the Lego NXT mobile robot to the system I could track it in real-time and send direct commands to it. The NXT mobile robot is controlled by using visual servoing. From the server direct commands were sent to a tracking application (that is stationary on the computer) that communicated with bluetooth to the NXT. A tracking marker was placed on the NXT and a reference marker for each camera was placed on the floor. The purpose was that the NXT was supposed to move towards the reference marker for each camera. If the reader is curious about the end result see reference [24].

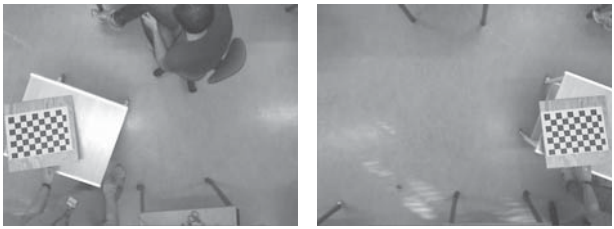

Figure 4.12 (*left*) Camera 0 (the reference camera), (*right*) Camera 2

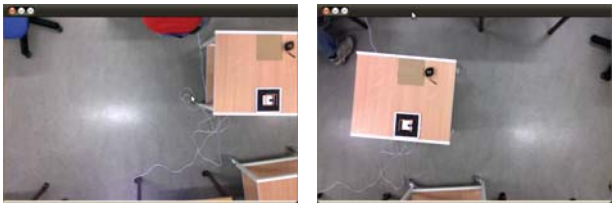

Figure 4.13 (*left*) Camera 1 , (*right*) Camera 0, the reference camera, tracking the marker

Figure 4.14 (*left*) Camera 0 showing the mobile NXT-robot moving towards the reference marker, (*right*) Camera 1 showing the NXT-robot moving towards the reference marker

5. Discussion and Conclusion

It is plausible that the same result can be achieved with just using Libdc1394 and OpenCV, but since my prior knowledge within in this area was non existing I considered it as an overwhelming task and the feasibility was very low. I did succeed with implementing the canny edge detector and the sobel operator in some manner, but the maximum fps that was acquired was not higher than a frame rate of 10 fps and that result is far from acceptable for real time applications. It takes a vast experience within the fields of computer vision and programming to construct a detection algorithm of a pattern for it to work robustly and efficiently.

ARToolkit contained all the tools that I needed for my detection and it was clearly beneficial that I had tried to implement a similar algorithm on my own which made it a lot easier to absorb the content within this computer vision library. However, ARToolkit has many flaws; one is that it is very hardware dependent both for the computer and the camera, and its best accuracy is when the marker is at 100 mm from the camera.

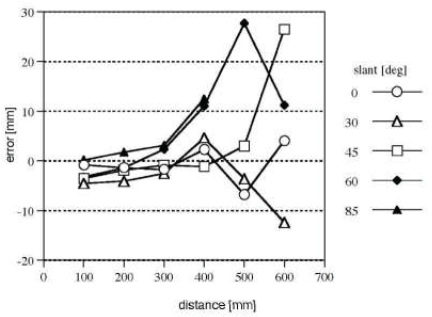

Figure 5.1 Performance test done by the developers for ARToolKit [5]

In a comparison with my evaluation of performance, see Figure 4.8, 4.9 and 4.10, my results seems highly plausible, considering that i calculated my own essential matrix and then translated the camera's coordinates to the reference camera. There will always be many different sources that will give rise to non-optimal condition for detection, such as the illumination sources, contours and contrasts and the most important factor calibration. I tried in some trails to use threshold filter in order to make a quicker and more robust detection.

The pattern size according to the developers of ARToolkit where a bigger marker size will improve the identification accuracy that is shown in Figure 5.2 matched with my results. The performance test done by the developers of ARToolkit has only an effective range up to 50 cm, whereas

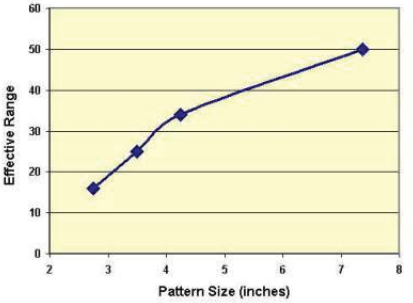

Figure 5.2 Performance test done by the developers for ARToolKit [5], where the Effective Range scale is in cm

my tests went from 60 up to 160 cm which are well above the ranges that have been documented in their tests.

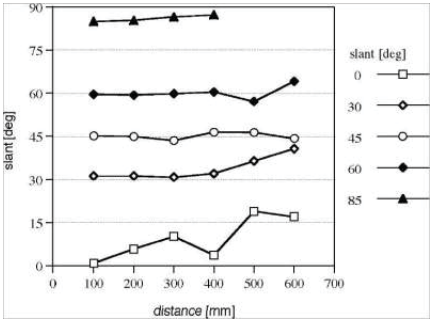

Figure 5.3 Performance test done by the developers for ARToolKit [5]

When reflecting over the matter that the essential matrix from the stereopsis calibration in matlab did not give me any satisfactory results, it must have been due to the fact that the toolbox uses another notation of the parameters than ARToolKit. The procedure that I used by using the equation that are explained in [15] and then by average the result in order to get representation that fits more with the reality. One could assume that the rotation matrix that is calculated within ARToolkit is constant when the marker is fixed, but that is not the case it constantly fluctuates. Perhaps that there is some deviation within the ARToolKit shell that it takes into account. I would like to emphasize again that the results are very dependent of the hardware of the camera and the quality of the calibration.

The calibration of the cameras was a time consuming and a very demanding process and it was quite hard to know what to do in order to

achieve good results when calibrating. Since it plays a major role for the calculus it is of the greatest importance that it is as precise as possible. When designing the sockets the choice of *nonblocking* UDP structure may not seem as the obvious choice, my motivation for my choice is; since the sampling is so high compared to the markers movement, when tracking e.g., robots with overhead cameras, its simpler solution has clearly more benefits than TCP sockets. After doing the overhead calibration according to Figure 2.2 which was hard to do between the cameras since it was a narrow area of view, the system behaved as I had anticipated even though it was a distance about 180 cm which is well over the documented range for ARToolkits performance. When working with the NXT it was not obvious of how to do the integration with my application. First I tried with the toolbox in Matlab that was provided from [13] but without any success. Since it was possible to use NXC on the existing firmware I also tried to write NXC applications that led nowhere. My main problem was; how to transmitt the data in real time to the NXT? I later tried to flash the firmware with LeJOS [10], where I could successfully connect through bluetooth with a socket and send direct commands to it. First I tried to integrated the new solution into my server application but I lost packages from the transmitting cameras due to some synchronization error since I introduced a new socket into the server application. This was solved by having the server creating files, one for each command, and then having the NXT application as a stand alone and then reading those files that the server created in real time. The NXT movement towards the reference marker can be considered as a Gain Scheduler where a initial static gain is applied that is lowered once the NXT reaches the satisfactory deviation between the reference marker and the tracking marker. The strength

Figure 5.4 Setup for the tracking of the nxt

with my result is not that you are able to track robots using computer vision, it is rather that the user is able to mount several cameras, calibrate them easily off-line with the help of the command option from the server where it bulks out data, and then load the essential matrix for the camera pairs. However, the most important feature is that it is possible

to give the tracking marker an absolute position in space and estimate its position with a better accuarcy, where one camera will always serve as a reference.

Future work

These are my thoughts on how to optimize my work within in this area for future student's.

- Experimentation with illumination and lighting of the environment for optimal condition.
- Implement optical flow and use that data to calculate the trajectory of the marker, e.g., using Kalman filter or other observer.
- Optimize the algorithms.

References

[1] David A. Forsyth, Jean Ponc. *Computer Vision: A MODDERN APPROACH*, ISBN 013608592X.

[2] Emanuele Trucco, Alessandro Verri, Prentice Hall (1998). *Introductory techniques for 3-D computer vision* , ISBN 0132611082.

[3] Gary Bradski, Adrian Kaehler - O'Reilly (2008), *Learning OpenCV: computer vision with the OpenCV library*, ISBN 0596516134.

[4] Libdc1394:
http://damien.douxchamps.net/ieee1394/libdc1394/ (2010)

[5] Hirokazu Kato, Mark Billinghurst, and Ivan Poupyrev. ARToolKit version 2.33 Manual, 2002.
http://www.hitl.washington.edu/artoolkit/ (2010)

[6] OpenCV:
http://opencv.willowgarage.com/wiki/ 2010)

[7] G. Bradski and A. Kaehler, *Learning OpenCV*, O'Reilly Media, Inc., 2008.

[8] Client Server sockets:
http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/index.jsp?topic=%2Frzab6%2Fhowdosockets.htm (2010)

[9] Lego mindstorms Group:
http://mindstorms.lego.com/en-us/Default.aspx (2012)

[10] LeJOS, java for Lego Mindstorm:
http://lejos.sourceforge.net/ (2012)

[11] OpenGL, http://www.opengl.org/ (2010)

[12] Matlab Camera Calibration Toolbox:
http://www.vision.caltech.edu/bouguetj/calib_doc/ (2010)

[13] RWTH - Mindstorms NXT Toolbox for MATLAB:
http://www.mindstorms.rwth-aachen.de/ (2012)

[14] Yuuki Matsumoto, Kunio Sakamoto, Shusaku Nomura Tetsuya Hirotomi, Kuninori Shiwaku and Masahito Hirakawa (2009) *Activity Replay System of Life Review Therapy Using Mixed Reality Technology* , ISBN: 978-988-17012-2-0

[15] Li Mei Song, Ming Ping Wang, Lu Lu, Huang Jing Huan (2006), *High precision camera calibration in vision measurement*, Province Key Laboratory of Robot Technique and Application, Information Engineering College, South West University of Science and Technology, SiChuan MianYang 621010, China

[16] Robert Lagani‘re, School of Information Technology and Engineering, University of Ottawa,Ottawa, Ont. CANADA K1N 6N5, *Morphological Corner Detection*

[17] Canny Edge detection algorithm
http://en.wikipedia.org/wiki/Canny_edge_detector (2010)

[18] Harris corner detection algorithm
http://en.wikipedia.org/wiki/Corner_detection (2010),C. Harris and M.Stephens (1988), *A combined corner and edge detector*, Proceedings of the 4th Alvey Vision Conference (pp.147-151).

[19] Gregory G. Slabaugh, *Computing Euler angles from a rotation matrix*, https://truesculpt.googlecode.com/hg-history/38000e9dfece971460473d5788c235fbbe82f31b/Doc/rotation_matrix_to_euler.pdf (2010)

[20] Magnus Johansson Bjärenstam, Michael Larsson, Master Thesis, LTH, Lund University, *Development of balancing robot with omni wheels (2012), ISBN-0280, ISRN LUTFD2/TFRT–5897–SE*

[21] Quickcam pro 9000, Logitech,
http://www.logitech.com/en-gb/support/3056?crid=405 (2013)

[22] C250, Logitech,
http://www.logitech.com/en-gb/support/5864?crid=405 (2013)

[23] Marlin, Allied Vision Technologies,
http://www.alliedvisiontec.com/us/products/cameras/firewire/marlin.html (2013)

[24] Lego NXT tracked with ARToolkit, Mikael Borg,
http://www.youtube.com/watch?v=tfsAU-z_RCQ (2012)

A. HowTo

serverCommandLine.c:
Starts the server and collects data from the client (the cameras).
-h for help
simplexx.c:
Download ARToolkit and replace the simple folder which is located in ARToolkit/ examples, and compile.
-h for help
Run the nxtTrack.c before you connect the cameras to the server since it needs some time to initiate connection. For calibration of the intrinsic, extrinsic and stereo extrinsic download Matlab camera calibration toolbox.

B. simpleTest.c

Source code for grabing frames with the cameras.

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "object.h"
#include <string.h>
#include <stdio.h>
#include<getopt.h>
#include <netdb.h>        /* netbd.h is needed for struct hostent =) */
#define PORT 12345        /* Open Port on Remote Host */
#define MAXDATASIZE 32000  /* Max number of bytes of data */
#define TRUE              1
#define FALSE             0
//
// Camera configuration.
//
#ifdef _WIN32
char                    *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                    *vconf = "v4l2src device=/dev/video0 use-fixed-fps=false
!  ffmpegcolorspace !capsfilter caps=video/x-raw-rgb,width=640,height=480,bpp=24 ! identity name=artoolkit ! fakesink";
#endif
/* Object Data */
char            *model_name = "Data/object_data2";
ObjectData_T    *object;
int             objectnum;
int             xsize, ysize;
int thresh = 115;
int             count = 0;
// Socket to server
int  fd;
float           r11, r12 ,r13 ,r21 ,r22 ,r23 , r31, r32, r33;
float           tx, ty, tz;
char           *cparam_name    = "Data/calib2306.dat";
ARParam         cparam;
static void   init(int videoNum);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static int draw( ObjectData_T *object, int objectnum );
static int  draw_object( int obj_id, double gl_para[16] );
typedef struct visionData
{
  int index;
  int trackMarkDetected;
  int refMarkDetected;
  float real;
  char string[20];
  float rot[3][3];
  float trans[3];
  float rotRef[3][3];
  float transRef[3];
} visionData_struct;
typedef struct pingData
{
  unsigned int id;
  int integer;
  float real;
  char string[20];
  struct timeval tv;
} pingData_struct;
visionData_struct con1;
pingData_struct con1Recv;
int pingData_size = sizeof(pingData_struct);
FILE *file;
int read_from_server(int fd,  pingData_struct *con)
{
  int left = sizeof(pingData_struct);
  char *ptr = (char *) con;
  printf("Trying to read from server, %d \n",left);
  while ( left >0 )
  {
    int bytes_read = read(fd,ptr,left);
    printf(" Read %d Left %d, socket %d, \n",bytes_read,left,fd);
    ptr+=bytes_read;
    if (bytes_read == 0)
    {
      printf("error occured on socket %d \n",fd);
      return 0;
    }
    left-=bytes_read;
    //printf(" Read %d Left %d, socket %d, boolean%d\n",bytes_read,left,fd,con1Recv.integer);
    if (left==0) {
      con1Recv.integer=TRUE;
      printf("boolean:%d\n", con1Recv.integer);
    }
  }
  return 1;
}
int scannedThresh=115;
int textFLAG=1;
int cameraNum;
char ip[20];
int main (int argc, char* argv[])
{
  while (1)
  {
    int option_index = 0;
    int optchr;
    static struct option long_options[4] = {
      {"Ip of server", 1,0,'i'},
      {"MountCamera", 1,0,'c'},
      {"Thresh",1,0,'t'},
```

```
      {"help",0,0,'h'},
    };
    optchr=getopt_long(argc, argv ,"ht:c:i:", long_options, &option_index);
    if (optchr==-1) {
      printf("%s -h to see help menu\n",argv[0]);
      break;
    }
    switch (optchr)
    {
    case 'i':
      strcpy(ip,optarg);
      printf("Ip enterd:%s\n", ip);
      break;
    case 'c':
      cameraNum=atoi(optarg);
      printf("Camera Num enterd %d \n",atoi(optarg));
      break;
    case 't':
      scannedThresh=atoi(optarg);
      printf("Threshold value enterd %d \n",atoi(optarg));
      break;
    case 'h':
      printf("Usage:\n");
      printf(" %s -i Ip adress of server\n",argv[0]);
      printf(" %s -c Camera number you want to mount\n",argv[0]);
      printf(" %s -t Threshold value\n",argv[0]);
      printf(" %s -h Display this help\n",argv[0]);
      return 0;
      break;
    }//end switch
  }//end while
  char *buff[512];
  glutInit(&argc, argv);
  if (cameraNum!=1) {
    textFLAG=2;
  }
  /*if (ip==NULL){
   printf("No ip adress of the server enterd\n");
   exit(0);
  }*/
  sprintf(buff,"%s%d %s","CAM",cameraNum,"transformation data.txt");
  file = fopen(&buff,"wt");
  init(cameraNum);
  connect_to_server();
  arVideoCapStart();
  argMainLoop( NULL, keyEvent, mainLoop );
  fclose(file);
  close(fd);
  return (0);
}
void connect_to_server()
{
  struct hostent *he;         /* structure that will get information about remote host */
  struct sockaddr_in server;  /* server's address information */
  if ((he=gethostbyname(ip))==NULL) {  /* calls gethostbyname() */
    printf("gethostbyname() error\n");
    exit(-1);
  }
  if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1) { /* calls socket() */
    printf("socket() error\n");
    exit(-1);
  }
  server.sin_family = AF_INET;
  server.sin_port = htons(PORT); /* htons() is needed again */
  server.sin_addr = *((struct in_addr *)he->h_addr);  /*he->h_addr passes "*he"'s info to "h_addr" */
  bzero(&(server.sin_zero),8);
  if (connect(fd, (struct sockaddr *)&server,sizeof(struct sockaddr))==-1) { /* calls connect() */
    printf("connect() error\n");
    exit(-1);
  }
  printf("Connected to server \n");
}
static void   keyEvent( unsigned char key, int x, int y)
{
  /* quit if the ESC key is pressed */
  if ( key == 0x1b ) {
    printf("*** %f (frame/sec) Average result:**********%f\n", (double)count/arUtilTimer());
    printf("*****************************************\n");
    printf("r11:%f r12:%f r13:%f \n",r11/count,r12/count,r13/count);
    printf("r21%f r22:%f r23:%f \n",r21/count,r22/count,r23/count);
    printf("r31:%f r32:%f r33:%f \n",r31/count,r32/count,r33/count);
    printf("*****************************************\n");
    printf("tx:%f ty:%f tz:%f \n",tx/count,ty/count,tz/count);
    cleanup();
    exit(0);
  }
}
/* main loop */
int count=0;
static void mainLoop(void)
{
  ARUint8         *dataPtr;
  ARMarkerInfo    *marker_info;
  int             marker_num;
  int             i,j,k;
  con1.refMarkDetected= FALSE;
  con1.trackMarkDetected= FALSE;
  struct timeval currentTime, currentTimeDel;
  double rot[3][3];
  double wa,wc ,wb;
  double pi = 3.14159265358979323846264643;
  /* grab a video frame */
  if ( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
    arUtilSleep(2);
    return;
  }
  if ( count == 0 ) arUtilTimerReset();
  count++;
  /*draw the video*/
  argDrawMode2D();
  argDispImage( dataPtr, 0,0 );
  glColor3f( 1.0, 0.0, 0.0 );
  glLineWidth(2.0);
  gettimeofday(&currentTime,NULL);
  /* detect the markers in the video frame */
  if (arDetectMarker(dataPtr, scannedThresh,
                     &marker_info, &marker_num) < 0 ) {
    cleanup();
    exit(0);
  }
  //for( i = 0; i < marker_num; i++ ) {   //Draws a square aroune the detected marker
  //argDrawSquare(marker_info[i].vertex,0,0);
  //}
  /* check for known patterns */
  for ( i = 0; i < objectnum; i++ ) {
    k = -1;
    for ( j = 0; j < marker_num; j++ ) {
```

```
      if ( object[i].id == marker_info[j].id) {
        /* you've found a pattern */
        //printf("Found pattern: %d ",patt_id);
        //glColor3f( 0.0, 1.0, 0.0 );
        //argDrawSquare(marker_info[j].vertex,0,0);
        if ( k == -1 ) k = j;
        else /* make sure you have the best pattern (highest confidence factor) */
          if ( marker_info[k].cf < marker_info[j].cf ) k = j;
      }
    }
    if ( k == -1 ) {
      object[i].visible = 0;
      continue;
    }
    object[i].visible = 1;
    /* get the transformation between the marker and the real camera */
    //arGetTransMatCont(&marker_info[k], patt_trans,patt_center, patt_width, patt_trans);//HISTORY BASED IE FASTER BUT LESS ACCURATE
    arGetTransMat(&marker_info[k],
                  object[i].marker_center, object[i].marker_width,
                  object[i].trans);
    if (object[i].visible==1 ) {
      count1++;
      printf("Calib_pat detected\n");
      con1.rotRef[0][0]=object[i].trans[0][0];
      con1.rotRef[0][1]=object[i].trans[0][1];
      con1.rotRef[0][2]=object[i].trans[0][2];
      con1.rotRef[1][0]=object[i].trans[1][0];
      con1.rotRef[1][1]=object[i].trans[1][1];
      con1.rotRef[1][2]=object[i].trans[1][2];
      con1.rotRef[2][0]=object[i].trans[2][0];
      con1.rotRef[2][1]=object[i].trans[2][1];
      con1.rotRef[2][2]=object[i].trans[2][2];
      con1.transRef[0]=object[i].trans[0][3];
      con1.transRef[1]=object[i].trans[1][3];
      con1.transRef[2]=object[i].trans[2][3];
      con1.index=count1;
      rot[0][0]=object[i].trans[0][0];
      rot[0][1]=object[i].trans[0][1];
      rot[0][2]=object[i].trans[0][2];
      rot[1][0]=object[i].trans[1][0];
      rot[1][1]=object[i].trans[1][1];
      rot[1][2]=object[i].trans[1][2];
      rot[2][0]=object[i].trans[2][0];
      rot[2][1]=object[i].trans[2][1];
      rot[2][2]=object[i].trans[2][2];
      con1.refMarkDetected= TRUE;
      //arGetAngle(rot,&wa,&wb,&wc); //Gets the euler angles of the rotation matrix
      printf("**********Calib_pat********************\n");
      printf("r11:%f r12:%f r13:%f \n",con1.rotRef[0][0],con1.rotRef[0][1],con1.rotRef[0][2]);
      printf("r21%f r22:%f r23:%f \n",con1.rotRef[1][0],con1.rotRef[1][1],con1.rotRef[1][2]);
      printf("r31:%f r32:%f r33:%f \n",con1.rotRef[2][0],con1.rotRef[2][1],con1.rotRef[2][2]);
      printf("************************************************\n");
      printf("tx:%f ty:%f tz:%f \n",con1.transRef[0],con1.transRef[1],con1.transRef[2]);
     /* fprintf(file,"%s%d%s%d%s %f %f %f %s %f %f %f %s %f %f %f %s %d %d %d %d%s\n",
              "x",textFLAG,"{",count1,"}=[",con1.rotRef[0][0],con1.rotRef[0][1],con1.rotRef[0][2],con1.transRef[0],";",
              con1.rotRef[1][0],con1.rotRef[1][1],con1.rotRef[1][2],con1.transRef[1],";",
              con1.rotRef[2][0],con1.rotRef[2][1],con1.rotRef[2][2],con1.transRef[2],";",
              0,    0   ,0    ,1,"];");
*/
      //printf("Wa:%03.3f, Wb:%03.3f, Wc:%03.3f\n", wa*180/pi, wb*180/pi, wc*180/pi);
    }
    if (object[0].visible==1) {
      printf("Hiro_pat detected\n");
      con1.trackMarkDetected= TRUE;
      con1.rot[0][0]=object[0].trans[0][0];
      con1.rot[0][1]=object[0].trans[0][1];
      con1.rot[0][2]=object[0].trans[0][2];
      con1.rot[1][0]=object[0].trans[1][0];
      con1.rot[1][1]=object[0].trans[1][1];
      con1.rot[1][2]=object[0].trans[1][2];
      con1.rot[2][0]=object[0].trans[2][0];
      con1.rot[2][1]=object[0].trans[2][1];
      con1.rot[2][2]=object[0].trans[2][2];
      con1.trans[0]=object[0].trans[0][3];
      con1.trans[1]=object[0].trans[1][3];
      con1.trans[2]=object[0].trans[2][3];
      printf("***********Hiro_pat*******************\n");
      printf("r11:%f r12:%f r13:%f \n",con1.rot[0][0],con1.rot[0][1],con1.rot[0][2]);
      printf("r21%f r22:%f r23:%f \n",con1.rot[1][0],con1.rot[1][1],con1.rot[1][2]);
      printf("r31:%f r32:%f r33:%f \n",con1.rot[2][0],con1.rot[2][1],con1.rot[2][2]);
      printf("************************************************\n");
      printf("tx:%f ty:%f tz:%f \n",con1.trans[0],con1.trans[1],con1.trans[2]);
     /* fprintf(file,"%s%d%s%d%s %f %f %f %s %f %f %f %s %f %f %f %s %d %d %d %d%s\n",
              "n",textFLAG,"{",count1,"}=[",con1.rot[0][0],con1.rot[0][1],con1.rot[0][2],con1.trans[0],";",
              con1.rot[1][0],con1.rot[1][1],con1.rot[1][2],con1.trans[1],";",
              con1.rot[2][0],con1.rot[2][1],con1.rot[2][2],con1.trans[2],";",
              0,    0   ,0    ,1,"];");
*/
    }
  if (object[0].visible!=1) {
 con1.trackMarkDetected= FALSE;
printf("trackmarkdetected false\n");
}
 if (object[1].visible!=1) {
 con1.refMarkDetected= FALSE;
printf("refmarkdetected false\n");
}
  }
  gettimeofday(&currentTimeDel,NULL);
  //arVideoCapNext();
  /* draw the AR graphics */
  draw( object, objectnum );
  int written =write(fd,&con1,sizeof(con1));
  /*swap the graphics buffers*/
  argSwapBuffers();
  /*int success=read_from_server(fd,&con1Recv);  //Reads pink pkt
   if(success==1){
   printf("Got from server %ld %ld %d\n", con1Recv.tv.tv_sec, con1Recv.tv.tv_usec,con1Recv.id);
   write(fd,&con1Recv,sizeof(con1Recv));   //Sends back ping pkt
   }
  */
  if (written==-1) {
    printf("Did not succesfully write to server \n");
    return 0;
  }
}
static void init(int videoNum )
{
  ARParam  wparam;
  char *buff[512];
  sprintf(buff,"%s%d %s","v4l2src device=/dev/video",videoNum,"use-fixed-fps=false
!  ffmpegcolorspace !capsfilter caps=video/x-raw-rgb,width=640,height=480,bpp=24 ! identity name=artoolkit ! fakesink");
```

```
  /* open the video path */
  if ( arVideoOpen( &buff ) < 0 ) exit(0);
  /* find the size of the window */
  if ( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
  printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);
  /* set the initial camera parameters */
  if ( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
    printf("Camera parameter load error !!\n");
    exit(0);
  }
  arParamChangeSize( &wparam, xsize, ysize, &cparam );
  /* Matlab calib parameters*/
  if (cameraNum==0) {
    cparam.mat[0][0] =  683.043561559270 52; //Focal length
    cparam.mat[0][1] =  0; //Skew
    cparam.mat[0][2] =  334.050181317925706;//Cx
    cparam.mat[1][2] = 217.477646403760247;//Cy
    cparam.mat[1][1] =  683.043561559270 52; //Focal lenght * aspect ratio
  }
  if (cameraNum==1) {
    //cparam.mat[0][0] = 681.427507297262196 ; //Focal length
    cparam.mat[0][1] =  2; //Skew
    //cparam.mat[0][2] =  319.161998371275274;//Cx
    cparam.mat[1][2] = 219.115459155136904;//Cy
    //cparam.mat[1][1] = 681.427507297262196; //Focal lenght * aspect ratio
  }
  if (cameraNum==2 || cameraNum==3) {
    cparam.mat[0][0] =   667.227223963916913; //Focal length
    cparam.mat[0][1] =  1; //Skew
    cparam.mat[0][2] =   334.050181317925706;//Cx
    cparam.mat[1][2] = 217.477646403760247;//219.340035671937954;//Cy
    cparam.mat[1][1] = 667.227223963916913; //Focal lenght * aspect ratio
  }
  arInitCparam( &cparam );
  printf("*** Camera Parameter ***\n");
  arParamDisp( &cparam );
  /* load in the object data - trained markers and associated bitmap files */
  if ( (object=read_ObjData(model_name, &objectnum)) == NULL ) exit(0);
  printf("Objectfile num = %d\n", objectnum);
  /* open the graphics window */
  argInit( &cparam, 1.0, 0, 0, 0, 0 );
}
/* cleanup function called when program exits */
static void cleanup(void)
{
  arVideoCapStop();
  arVideoClose();
  argCleanup();
}
/* draw the the AR objects */
static int draw( ObjectData_T *object, int objectnum )
{
  int     i;
  double  gl_para[16];
  glClearDepth( 1.0 );
  glClear(GL_DEPTH_BUFFER_BIT);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LEQUAL);
  glEnable(GL_LIGHTING);
  /* calculate the viewing parameters - gl_para */
  for ( i = 0; i < objectnum; i++ ) {
    if ( object[i].visible == 0 ) continue;
    argConvGlpara(object[i].trans, gl_para);
    draw_object( object[i].id, gl_para);
  }
  glDisable( GL_LIGHTING );
  glDisable( GL_DEPTH_TEST );
  return(0);
}
/* draw the user object */
static int  draw_object( int obj_id, double gl_para[16])
{
  GLfloat   mat_ambient[] = {0.0, 0.0, 1.0, 1.0};
  GLfloat   mat_ambient_collide[]     = {1.0, 0.0, 0.0, 1.0};
  GLfloat   mat_flash[] = {0.0, 0.0, 1.0, 1.0};
  GLfloat   mat_flash_collide[]       = {1.0, 0.0, 0.0, 1.0};
  GLfloat   mat_flash_shiny[] = {50.0};
  GLfloat   light_position[]  = {100.0,-200.0,200.0,0.0};
  GLfloat   ambi[]                    = {0.1, 0.1, 0.1, 0.1};
  GLfloat   lightZeroColor[]  = {0.9, 0.9, 0.9, 0.1};
  argDrawMode3D();
  argDraw3dCamera( 0, 0 );
  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixd( gl_para );
  /* set the material */
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
  if (obj_id == 0) {
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash_collide);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_collide);
    /* draw a cube */
    glTranslatef( 0.0, 0.0, 20.0 ); //Translations of how gl draws the cube on the marker
    glutWireCube(80);//Size of the drawn cube
  }
  else {
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    /* draw a cube */
    glTranslatef( 0.0, 0.0, 20.0 );
    glutWireCube(80);
  }
  argDrawMode2D();
  return 0;
}
```

C. servercommandline.c

The server for connecting the cameras and doing the calculation.

```
/*
This program starts up a server to communicate with a number of specified cameras. The cameras that connects
acts like clients and sends data to the server. All the data about the markers are processed in the server.
There are different options at the commandline from which the user can choose from.
Package/libraries needed to run this system is:
ARToolKit
OpenCV
Matlab camera calib toolbox
Concerning the rig in lab C Cam 0 is the mid cam, Cam 1 the furthest away, Cam 2 the closest to the entrance!!!
g++ -lnsl -lresolv -lm  -lhighgui -lcv -lcvaux -I/usr/local/include/opencv/ -o streamserverCommandlinev.10 serverCommandLinev.10.c
*/
#include <stdio.h>/* These are the usual header files */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <getopt.h>
#include "cv.h"
#include "highgui.h"
#include <cvaux.h>
#include <stdlib.h>
#include <math.h>
#define DEBUG
#ifdef DEBUG
#include <stdarg.h>
#include <errno.h>
#define PORT 12345 /* Port that will be opened */
#define BACKLOG 2 /* Number of allowed connections */
#define TRUE             1
#define FALSE            0
#define PROCESSED          2
void
debug (char *fmt, ...)
{
  va_list ap;
  va_start (ap, fmt);
  fprintf (stderr, "DEBUG: ");
  vfprintf (stderr, fmt, ap);
  va_end (ap);
}
#else
void
debug (char *fmt, ...)
{
  ;
}
#endif
typedef struct connInfo
{
  int sockfd;
  int remaining;
  int remainingDelay;
  int remainingRefMark;
  struct timeval lastSent;
  struct timeval lastRecv;
  struct sockaddr_in clientAddr;
  int sin_size;
  int index;
  int doneCamera;
  int doneRefMark;
  int doneDelay;
  unsigned int id;
} connInfo_struct;
typedef struct visionData
{
  int index;
  int trackMarkDetected;
  int refMarkDetected;
  float real;
  char string[20];
  float rot[3][3];
  float trans[3];
  float rotRef[3][3];
  float transRef[3];
} visionData_struct;
typedef struct pingData
{
  unsigned int id;
  int integer;
  float real;
  char string[20];
  struct timeval tv;
} pingData_struct;
int visionData_size = sizeof(visionData_struct);
int pingData_size = sizeof(pingData_struct);
#define MAXCONCON 10
int numConCons = 0;
int nxtUSE=1;//if use nxt=1 else 0
pingData_struct pingPkt[MAXCONCON];
visionData_struct visionPkt[MAXCONCON];
connInfo_struct visionCon[MAXCONCON];
connInfo_struct connectedCameras[MAXCONCON];
int pktCount=0;
int doTranstoCam1RefMark (visionData_struct rot1, CvMat* transFormMatrix, CvMat result3);
int invTranstoCam1 (visionData_struct rot1, visionData_struct rot2, CvMat result);
int doTranstoCam0TrackMark (visionData_struct rot2, CvMat* transFormMatrix1, CvMat result1);
int read_from_camera(connInfo_struct *con)
{
  char *ptr1 = (char *) con;
  int bytes_read1 = read(con->sockfd,
                         (char *)(&visionPkt[con->index])+
                         visionData_size - con->remaining ,
                         con->remaining);
  //printf(" Read %d Left %d, socket %d \n",bytes_read, visionData_size - con->remaining,con->sockfd);
  ptr1+=bytes_read1;
  if (bytes_read1 == -1)
  {
    //printf("socket %d read returned -1 \n",con->sockfd);
    if (errno==EAGAIN);
    //printf("nothing to read yet on socket %d \n",con->sockfd);
    return 1;
  }
  else  {
    con->remaining -= bytes_read1;
```

```
    //printf("Bytes read:%d",bytes_read);
  }
  if (con->remaining <= 0) {
    pktCount++;
    con->doneCamera=1;
    con->remaining = visionData_size;
  }
  else {
    con->doneCamera=0;
  }
  return 1;
}
int read_from_cameraDelay(connInfo_struct *con)
{
  char *ptr = (char *) con;
  int bytes_read = read(con->sockfd,
                        (char *)(&pingPkt[con->index])+
                        pingData_size - con->remainingDelay ,
                        con->remainingDelay);
  //printf(" Read %d Left %d, socket %d, pointer %x \n",bytes_read,left,fd,ptr);
  if (bytes_read == -1)
  {
    //printf("socket %d read returned -1 \n",con->sockfd);
    if (errno==EAGAIN);
    //printf("nothing to read yet on socket %d \n",con->sockfd);
    return 1;
  }
  else  {
    con->remainingDelay -= bytes_read;
    //printf("Bytes read:%d",bytes_read);
  }
  if (con->remainingDelay <= 0) {
    con->doneDelay=1;
    con->remainingDelay = pingData_size;
    //printf("Completed Packet\n");
  }
  else {
    con->doneDelay=0;
  }
  return 1;
}
int write_to_camera(connInfo_struct *con) {
  char *ptr = (char *) con;
  gettimeofday(&pingPkt[con->index].tv,NULL);
//printf("Writing to camera %ld %ld %d\n",pingPkt[con->index].tv.tv_sec, pingPkt[con->index].tv.tv_usec,con->id);
  pingPkt[con->index].id = con->id++;
  int bytes_read = write(con->sockfd,(char *)(&pingPkt[con->index]), pingData_size);
}
void
printMat (CvMat * A)
{
  int i, j;
  for (i = 0; i < A->rows; i++)
  {
    printf ("\n");
    switch (CV_MAT_DEPTH (A->type))
    {
    case CV_32F:
    case CV_64F:
      for (j = 0; j < A->cols; j++)
        printf ("%8.3f ", (float) cvGetReal2D (A, i, j));
      break;
    case CV_8U:
    case CV_16U:
      for (j = 0; j < A->cols; j++)
        printf ("%6d", (int) cvGetReal2D (A, i, j));
      break;
    default:
      break;
    }
  }
  printf ("\n");
}
//STUFF FOR EVALUATION!! Stores the data in a file just uncomment "SK 1 and SK 2 " to run
FILE *eval;
double r11t,r12t,r13t,r21t,r22t,r23t,r31t,r32t,r33t,txt,tyt,tzt;
int countEval=0;
int
main (int argc, char* argv[])
{
  FILE *file,*nxtDataCmd0, *nxtDataCmd1, *nxtDataCmd2, *nxtDataCmd3, *nxtDataCmd4, *nxtDataCmd5, *nxtDataCmd6, *nxtDataCmd7;
  FILE *p;
//eval=fopen("Evaldist220cm.m","wt"); //SK 2
  int numOfConnectedCams=0;
  char buff[1024];
  double init[4][4];
  double init2[4][4];
  int count=0;
  CvMat Rab = cvMat (4, 4, CV_64FC1, init);
  CvMat Tab = cvMat (4, 4, CV_64FC1, init);
  CvMat CamXO1 = cvMat (4, 4, CV_64FC1, init);
  CvMat *endResults[MAXCONCON];
  CvMat calibResults[MAXCONCON];
  CvMat Cam22Cam1RefMark = cvMat (4, 4, CV_64FC1, init);
  CvMat Cam22Cam1TrackMark = cvMat (4, 4, CV_64FC1, init);
  CvMat Cam22Cam1Track2Ref = cvMat (4, 4, CV_64FC1, init2);
  CvMat Cam32Cam1 = cvMat (4, 4, CV_64FC1, init);
  int readSuccessCameraCalib;
  int k=0;
  int calibMatExist = FALSE;
  double r11,r12,r13,r21,r22,r23,r31,r32,r33,tx,ty,tz;
  int fd, on = 1; /* file descriptors */
  int desc_ready, end_server = FALSE;
  struct sockaddr_in server; /* server's address information */
  struct sockaddr_in client1, client2, client3; /* client's address information */
  struct timeval timeout;
  fd_set master_set, working_set;
  int sin_size;
  int gnDebug=0;
  int numCams=2;
  int dataLoad;
  double pi = 3.14159265358979323846264643;
  sin_size = sizeof (struct sockaddr_in);
  float camDist, camDist2, tz11, alphaRad, alphaDeg, x, tempDeg, tz22, tzEst;
  int i;
  /*Declarations for transmitting data into nxtTrack.c*/
int nxtTrackXCam0=0;
int nxtTrackYCam0=0;
int nxtFlagXCam0 = 1;
int nxtFlagYCam0 = 1;
int nxtFlagXCam0Done = 1;
int nxtFlagYCam0Done = 1;
int nxtTrackXCam1=0;
int nxtTrackYCam1=0;
int nxtFlagXCam1 = 1;
int nxtFlagYCam1 = 1;
int nxtFlagXCam1Done = 1;
int nxtFlagYCam1Done = 1;
char buffDataNxtCmd0[512];
char buffDataNxtCmd1[512];
```

```
char buffDataNxtCmd2[512];
char buffDataNxtCmd3[512];
char buffDataNxtCmd4[512];
char buffDataNxtCmd5[512];
char buffDataNxtCmd6[512];
char buffDataNxtCmd7[512];
  while (1)
  {
    int option_index = 0;
    int optchr;
    static struct option long_options[5] = {
      {"Calib Mode", 0,0,'c'},
      {"Normal Mode ",0,0,'n'},
      {"Extract Data ",0,0,'e'},
      {"Number of Cameras",0,0,'q'},
      {"help",0,0,'h'}
    };
    optchr=getopt_long(argc, argv ,"hnNe:E:c:C:q:", long_options, &option_index);
    if (optchr==-1) {
      printf("%s -h to see help menu\n",argv[0]);
      break;
    }
    switch (optchr)
    {
    case 'q':
      numCams=atoi(optarg);
      printf("%d Cameras to be connected\n",numCams);
      break;
    case 'C':
    case 'c':
      printf("Entering Calib Mode\n");
      printf("Maximum number of Cameras:%d \n",MAXCONCON);
      printf("You have selected %d Cameras to connect\n",numCams);
      printf("Connect the Cameras\n");
      int calibCams;
      calibCams = atoi(optarg);
      printf("Calibrate betwen camera 0 and %d \n",calibCams);
      char buffCam[512];
      sprintf(buffCam,"%s%d%s","CalibCam0toCam",calibCams,".m");
      file = fopen(buffCam,"wt");
      if ((fd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
      { /* calls socket() */
        printf ("socket() error\n");
        exit (-1);
      }
      server.sin_family = AF_INET;
      server.sin_port = htons (PORT); /* Remember htons() from "Conversions" section? =) */
      server.sin_addr.s_addr = INADDR_ANY; /* INADDR_ANY puts your IP address automatically */
      bzero (&(server.sin_zero), 8); /* zero the rest of the structure */
      if (bind (fd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
      { /* calls bind() */
        printf ("bind() error\n");
        exit (-1);
      }
      if (listen (fd, numCams) == -1)
      { /* calls listen() */
        printf ("listen() error\n");
        exit (-1);
      }
      /* accepting all connections */
      for (i=0; i<numCams; i++) {
        visionCon[i].sockfd= accept(fd, (struct sockaddr *) &visionCon[i].clientAddr,
                                    (socklen_t *)&visionCon[i].sin_size);
        if (visionCon[i].sockfd!=-1) {
          connectedCameras[numOfConnectedCams]=visionCon[i];
          printf("Entered loop\n");
          connectedCameras[numOfConnectedCams].index = numOfConnectedCams;
          if (ioctl (connectedCameras[numOfConnectedCams].sockfd, FIONBIO, (char *) &on) == -1)
          {
            perror ("ioctl() failed");
            close (connectedCameras[numOfConnectedCams].sockfd);
            exit (-1);
          }
          connectedCameras[numOfConnectedCams].remaining = sizeof(visionData_struct);
          connectedCameras[numOfConnectedCams].remainingDelay = sizeof(pingData_struct);
          numOfConnectedCams++;
        }
        printf("Camera %d connected, socket %d \n",i,
               visionCon[i].sockfd);
        visionCon[i].index = i;
        if (ioctl (visionCon[i].sockfd, FIONBIO, (char *) &on) == -1)
        {
          perror ("ioctl() failed");
          close (visionCon[i].sockfd);
          exit (-1);
        }
        visionCon[i].remaining = sizeof(visionData_struct);
        visionCon[i].remainingDelay = sizeof(pingData_struct);
      }
      while (1)
      {
        for (i=0; i<numCams; i++) {
          int readSuccessCamera  = read_from_camera(&connectedCameras[i]);
          if (readSuccessCamera==0)
          {
            printf("bad read\n");
            return 0 ;
          }
        }
        if ( connectedCameras[0].doneCamera==1 && visionPkt[0].refMarkDetected==TRUE &&
             connectedCameras[1].doneCamera==1 && visionPkt[1].refMarkDetected==TRUE   ) {
          //Stores the data in a file for calibration in matlab
          fprintf(file,"%s%d%s %f %f %f %f %s %f %f %f %f %s %f %f %f %f %s %d %d %d %d%s\n",
                  "Cam0{",visionPkt[0].index,"}=[",
                  visionPkt[0].rotRef[0][0],visionPkt[0].rotRef[0][1],visionPkt[0].rotRef[0][2],visionPkt[0].transRef[0],";",
                  visionPkt[0].rotRef[1][0],visionPkt[0].rotRef[1][1],visionPkt[0].rotRef[1][2],visionPkt[0].transRef[1],";",
                  visionPkt[0].rotRef[2][0],visionPkt[0].rotRef[2][1],visionPkt[0].rotRef[2][2],visionPkt[0].transRef[2],";",
                  0,    0   ,0      ,1,"];");
          fprintf(file,"%s%d%s%d%s %f %f %f %f %s %f %f %f %f %s %f %f %f %f %s %d %d %d %d%s\n",
                  "Cam",calibCams,"{",visionPkt[1].index,"}=[",
                  visionPkt[1].rotRef[0][0],visionPkt[1].rotRef[0][1],visionPkt[1].rotRef[0][2],visionPkt[1].transRef[0],";",
                  visionPkt[1].rotRef[1][0],visionPkt[1].rotRef[1][1],visionPkt[1].rotRef[1][2],visionPkt[1].transRef[1],";",
                  visionPkt[1].rotRef[2][0],visionPkt[1].rotRef[2][1],visionPkt[1].rotRef[2][2],visionPkt[1].transRef[2],";",
                  0,    0   ,0      ,1,"];");
          fprintf(file,"%s","\n");
          invTranstoCam1(visionPkt[0], visionPkt[1],CamX01);
          r11=CV_MAT_ELEM(CamX01, double , 0,0);
          r12=CV_MAT_ELEM(CamX01, double , 0,1);
          r13=CV_MAT_ELEM(CamX01, double , 0,2);
          r21=CV_MAT_ELEM(CamX01, double , 1,0);
          r22=CV_MAT_ELEM(CamX01, double , 1,1);
          r23=CV_MAT_ELEM(CamX01, double , 1,2);
          r31=CV_MAT_ELEM(CamX01, double , 2,0);
          r32=CV_MAT_ELEM(CamX01, double , 2,1);
          r33=CV_MAT_ELEM(CamX01, double , 2,2);
          tx=CV_MAT_ELEM(CamX01, double , 0,3);
```

```
          ty=CV_MAT_ELEM(CamX01, double , 1,3);
          tz=CV_MAT_ELEM(CamX01, double , 2,3);
          count++;
          fprintf(file,"%s%s%d%s %f %f %f %f %s %f %f %f %f %s %f %f %f %f %s %d %d %d %d%s\n",
                  "x","[",count,"]=[",
                  r11,r12,r13,tx,";",
                  r21,r22,r23,ty,";",
                  r31,r32,r33,tz,";",
                  0,   0   ,0    ,1,"];");
          fprintf(file,"%s%d%s %d%s%d%s \n\n","indexes{",count,"}= [ ",visionPkt[0].index," ",
                  visionPkt[1].index," ];");
          if (count==500) {
            fprintf(file,"%s%d%s","tempString='Camera",calibCams,"';");
            fclose(file);
            break;
          }
          connectedCameras[0].doneCamera=PROCESSED;
          connectedCameras[1].doneCamera=PROCESSED;
        }
      }
      break;
    case 'e':
    case 'E':
      dataLoad = atoi(optarg);
      printf("You have selected to load %d calibration files files\n",dataLoad); //Read the specified calibration file
      for (i=1; i<=dataLoad; i++) {
        char buffLoadData[512];
        sprintf(buffLoadData,"%s%d%s","CalibDataCamera0Camera",i,".txt");
        if ((p=fopen(buffLoadData,"r"))==NULL) {
          printf("Cannot open file.\n");
          exit(1);
        }
        fscanf( p, "%s  ",buff);
        double r11 = atof(buff);
        fscanf(p,"%s ",buff);
        double r12 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r13 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r21 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r22 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r23 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r31 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r32 = atof(buff);
        fscanf( p, "%s  ",buff);
        double r33 = atof(buff);
        fscanf( p, "%s  ",buff);
        double tx = atof(buff);
        fscanf( p, "%s  ",buff);
        double ty = atof(buff);
        fscanf( p, "%s  ",buff);
        double tz = atof(buff);
        /*Data Loaded from Calib file*/
        printf("***********Extracted Data from file%d*****\n",i);
        printf("r11:%f r12:%f r13:%f \n",r11,r12,r13);
        printf("r21%f r22:%f r23:%f \n",r21,r22,r23);
        printf("r31:%f r32:%f r33:%f \n",r31,r32,r33);
        printf("*******************************************\n");
        printf("tx:%f ty:%f tz:%f \n",tx,ty,tz);
        CvMat calibMat;
        double calibMatHeader[4][4] =
          { r11, r12, r13, tx,
            r21, r22, r23, ty,
            r31, r32, r33, tz,
            0, 0, 0, 1
          };
        cvInitMatHeader (&calibMat, 4, 4, CV_64FC1, calibMatHeader, CV_AUTO_STEP);
        calibResults[i-1]=calibMat;
//cvReleaseMatHeader(&calibMat);
printMat(&calibResults[i-1]);
CvMat temp;
double dealloc[4][4] =
        { 0, 0, 0, 0,
    0, 0, 0, 0,
          0, 0, 0, 0,
          0, 0, 0, 1
        };
        cvInitMatHeader (&temp, 4, 4, CV_64FC1, dealloc, CV_AUTO_STEP);
calibMat=temp;
}
      calibMatExist=TRUE;
    case 'n':
    case 'N':
      printf("Entering normal mode\n");
      printf("Maximum number of Cameras:%d \n",MAXCONCON);
      printf("You have selected %d Cameras to connect\n",numCams);
      printf("Connect the Cameras\n");
      if ((fd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
      { /* calls socket() */
        printf ("socket() error\n");
        exit (-1);
      }
      server.sin_family = AF_INET;
      server.sin_port = htons (PORT); /* Remember htons() from "Conversions" section? =) */
      server.sin_addr.s_addr = INADDR_ANY; /* INADDR_ANY puts your IP address automatically */
      bzero (&(server.sin_zero), 8); /* zero the rest of the structure */
      if (bind (fd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
      { /* calls bind() */
        printf ("bind() error\n");
        exit (-1);
      }
      if (listen (fd, numCams) == -1)
      { /* calls listen() */
        printf ("listen() error\n");
        exit (-1);
      }
      for (i=0; i<numCams; i++) {
        visionCon[i].sockfd= accept(fd, (struct sockaddr *) &visionCon[i].clientAddr,
                                    (socklen_t *)&visionCon[i].sin_size);
        if (visionCon[i].sockfd!=-1) {
          connectedCameras[numOfConnectedCams]=visionCon[i];
          printf("Entered loop\n");
          connectedCameras[numOfConnectedCams].index = numOfConnectedCams;
          if (ioctl (connectedCameras[numOfConnectedCams].sockfd, FIONBIO, (char *) &on) == -1)
          {
            perror ("ioctl() failed");
            close (connectedCameras[numOfConnectedCams].sockfd);
            exit (-1);
          }
          connectedCameras[numOfConnectedCams].remaining = sizeof(visionData_struct);
          connectedCameras[numOfConnectedCams].remainingDelay = sizeof(pingData_struct);
          numOfConnectedCams++;
        }
        printf("Camera %d connected, socket %d \n",i,
```

```
              visionCon[i].sockfd);
        visionCon[i].index = i;
        if (ioctl (visionCon[i].sockfd, FIONBIO, (char *) &on) == -1)
        {
          perror ("ioctl() failed");
          close (visionCon[i].sockfd);
          exit (-1);
        }
        visionCon[i].remaining = sizeof(visionData_struct);
        visionCon[i].remainingDelay = sizeof(pingData_struct);
      }
/* Declarations for transmitting data to nxt */
sprintf(buffDataNxtCmd0,"%s","nxtDataCmd0.txt");
sprintf(buffDataNxtCmd1,"%s","nxtDataCmd1.txt");
sprintf(buffDataNxtCmd2,"%s","nxtDataCmd2.txt");
sprintf(buffDataNxtCmd3,"%s","nxtDataCmd3.txt");
sprintf(buffDataNxtCmd4,"%s","nxtDataCmd4.txt");
sprintf(buffDataNxtCmd5,"%s","nxtDataCmd5.txt");
sprintf(buffDataNxtCmd6,"%s","nxtDataCmd6.txt");
sprintf(buffDataNxtCmd7,"%s","nxtDataCmd7.txt");
nxtDataCmd0 = fopen(buffDataNxtCmd0,"wt");
nxtDataCmd1 = fopen(buffDataNxtCmd1,"wt");
nxtDataCmd2 = fopen(buffDataNxtCmd2,"wt");
nxtDataCmd3 = fopen(buffDataNxtCmd3,"wt");
nxtDataCmd4 = fopen(buffDataNxtCmd4,"wt");
nxtDataCmd5 = fopen(buffDataNxtCmd5,"wt");
nxtDataCmd6 = fopen(buffDataNxtCmd6,"wt");
nxtDataCmd7 = fopen(buffDataNxtCmd7,"wt");
      while (1)
      {
        for (i=0; i<numOfConnectedCams; i++) {
          int readSuccessCamera  = read_from_camera(&connectedCameras[i]);
/*  if (visionCon[i].doneCamera==1)
              {
                int pingSuccess = write_to_camera(&visionCon[i]); //Send the delay pkt to client
                int readSuccessDelay  = read_from_cameraDelay(&visionCon[i]);//recieve the delay pkt from client
                struct timeval currentTime;
                gettimeofday(&currentTime,NULL);
                //printf("Reading from camera %d %ld %ld, id %d , curr time %ld %ld\n",i, pingPkt[1].tv.tv_sec, pingPkt[1].tv.tv_usec,
pingPkt[1].id,currentTime.tv_sec,currentTime.tv_usec);
              }
              if (readSuccessCamera==0)
              {
                printf("bad read\n");
                return 0 ;
              }
            */
  //printf("Cam1 tx:%f, ty:%f, tz:%f\n",visionPkt[0].trans[0], visionPkt[0].trans[1], visionPkt[0].trans[2]);
        }
        for ( i=1 ; i<numOfConnectedCams; i++) {
if ( connectedCameras[i].doneCamera==1 && visionPkt[i].trackMarkDetected==TRUE    ) {
connectedCameras[i].doneCamera=PROCESSED; //pkt been proccesed
                        doTranstoCam0TrackMark(visionPkt[i],&calibResults[i-1], Cam22Cam1TrackMark);
//Translates trackMark from camera[i]'s CS to the referens camera's CS
  }
          if ( connectedCameras[i].doneCamera==1 && visionPkt[i].refMarkDetected==TRUE    ) {
            if (calibMatExist==TRUE) {
              //doTranstoCam1RefMark(visionPkt[i],calibResults[i-1],Cam22Cam1RefMark);
//Translates refMark from camera[i]'s Coordinate System to the referens camera's CS
              //doTranstoCam1TrackMark(visionPkt[1],calibResults[i-1], Cam22Cam1TrackMark);
//Translates trackMark from camera[i]'s CS to the referens camera's CS
                if (visionPkt[0].trackMarkDetected==TRUE && connectedCameras[0].doneCamera==1) {
                // trans2Marker2MarkerrelationBeteen2Cams(visionPkt[i],visionPkt[0],calibResults[i-1], Cam22Cam1Track2Ref,i);
//Translates camera[i]'s track marker to the referns camera CS and compares the relative positions between them
                  //printMat(&calibResults[i-1]);
                  //trans2Marker2Markerrelation(visionPkt[i],calibResults[i-1], Cam22Cam1Track2Ref,i);
//Translates both markers from camera[i]'s to the  referens camera coordinate system.
                  //calibResults[i-1]=Cam22Cam1Track2Ref;
                  //printMat(&calibResults[numCams-1]);
                  //  printf ("***Trans to refMark from trackMark*****\n");
                  //printMat(&Cam22Cam1Track2Ref);
                }
              }
            }
          }
if ( nxtUSE==1){
if(nxtTrackXCam0== 0 && visionPkt[0].trackMarkDetected==TRUE && visionPkt[0].refMarkDetected==TRUE)
{
if(nxtFlagXCam0==1){
printf("Sending cmd1\n"); //Debug outprint
fprintf(nxtDataCmd1,"%s","1");
fclose(nxtDataCmd1);
nxtFlagXCam0=0;
}
}
if(  visionPkt[0].trans[0] > visionPkt[0].transRef[0]-100){
if(nxtFlagXCam0Done==1){
printf("Sending cmd2\n");//Debug outprint
nxtTrackXCam0=1;
nxtFlagXCam0Done=0;
printf("Tracking of first marker around x done\n");
fprintf(nxtDataCmd0,"%s","1");
fclose(nxtDataCmd0);
}
}
if(  visionPkt[0].trans[1] > visionPkt[0].transRef[1] && nxtTrackXCam0== 1 && nxtFlagYCam0==1){
if(nxtFlagYCam0==1){
printf("Sending cmd3\n");//Debug outprint
nxtFlagYCam0=0;
nxtTrackYCam0=1;
fprintf(nxtDataCmd2,"%s","1");
fclose(nxtDataCmd2);
}
}
if(  visionPkt[0].trans[1] < visionPkt[0].transRef[1]+150 && nxtTrackYCam0==1){
if(nxtFlagYCam0Done==1){
printf("Sending Cmd4\n");//Debug outprint
nxtFlagYCam0Done=0;
fprintf(nxtDataCmd3,"%s","1");
fclose(nxtDataCmd3);
}
}
if(nxtTrackXCam1== 0 && visionPkt[1].trackMarkDetected==TRUE && visionPkt[1].refMarkDetected==TRUE &&   nxtFlagYCam0Done==0)
{
if(nxtFlagXCam1==1){
printf("Sending cmd5\n"); //Debug outprint
fprintf(nxtDataCmd4,"%s","1");
fclose(nxtDataCmd4);
nxtFlagXCam1=0;
}
}
if(  visionPkt[1].trans[0] > visionPkt[1].transRef[0]-150 && nxtFlagYCam0Done==0
&& nxtFlagXCam1==0 && visionPkt[1].trackMarkDetected==TRUE){
if(nxtFlagXCam1Done==1){
printf("Sending cmd6\n");//Debug outprint
nxtTrackXCam1=1;
```

```
nxtFlagXCam1Done=0;
printf("Tracking of second marker around x done\n");
//printf("trackmark ty:%f, refmark ty:%f\n",visionPkt[1].trans[0],visionPkt[1].transRef[0]);
fprintf(nxtDataCmd5,"%s","1");
fclose(nxtDataCmd5);
}
}
if(  visionPkt[1].trans[1] < visionPkt[1].transRef[1] && nxtTrackXCam1== 1 && nxtFlagYCam1==1){
if(nxtFlagYCam1==1){
printf("Sending cmd7\n");//Debug outprint
nxtFlagYCam1=0;
nxtTrackYCam1=1;
fprintf(nxtDataCmd6,"%s","1");
fclose(nxtDataCmd6);
}
}
if(  visionPkt[1].trans[1] > visionPkt[0].transRef[1] && nxtTrackYCam1==1 &&  nxtFlagYCam1==0){
if(nxtFlagYCam1Done==1){
printf("Sending Cmd8\n");//Debug outprint
nxtFlagYCam1Done=0;
fprintf(nxtDataCmd7,"%s","1");
fclose(nxtDataCmd7);
}
}
        }
      }
      break;
    case 'h':
      printf("Usage:\n");
      printf(" %s -q Number of cameras to connect (default 2)\n",argv[0]);
      printf(" %s -c Calibration Mode:Input which camera u want to calibrate with respecte to Cam 0\n",argv[0]);
      printf(" %s -e Extract Data:Input how many files u want to load\n",argv[0]);
      printf(" %s -n Normal Mode\n",argv[0]);
      printf(" %s -h Display this help\n",argv[0]);
      return 0;
      break;
   }//end switch
  }//end while
  return 0;
}
int
invTranstoCam1 (visionData_struct rot1, visionData_struct rot2, CvMat result)
{
  CvMat Tm1;
  CvMat Tm2;
  CvMat x2;
  int i;
  double resultInit[4][4];
  double resultInit1[4][4];
  CvMat inv = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat result1 = cvMat (4, 4, CV_64FC1, resultInit1);
  double matheaderTm1[4][4] =
    { rot1.rotRef[0][0], rot1.rotRef[0][1], rot1.rotRef[0][2], rot1.transRef[0],
      rot1.rotRef[1][0], rot1.rotRef[1][1], rot1.rotRef[1][2], rot1.transRef[1],
      rot1.rotRef[2][0], rot1.rotRef[2][1], rot1.rotRef[2][2], rot1.transRef[2],
      0, 0, 0, 1
    };
  double matheaderTm2[4][4] =
    { rot2.rotRef[0][0], rot2.rotRef[0][1], rot2.rotRef[0][2], rot2.transRef[0],
      rot2.rotRef[1][0], rot2.rotRef[1][1], rot2.rotRef[1][2], rot2.transRef[1],
      rot2.rotRef[2][0], rot2.rotRef[2][1], rot2.rotRef[2][2], rot2.transRef[2],
      0, 0, 0, 1
    };
  cvInitMatHeader (&Tm1, 4, 4, CV_64FC1, matheaderTm1, CV_AUTO_STEP);
  cvInitMatHeader (&Tm2, 4, 4, CV_64FC1, matheaderTm2, CV_AUTO_STEP);
  printf ("Tm1:\n");
  printMat (&Tm1);
  printf ("**********************************\n");
  printf ("Tm2: \n");
  printMat (&Tm2);
  printf ("**********************************\n");
  cvInvert (&Tm2, &inv, CV_LU);
  cvMatMul (&Tm1,&inv, &result);
  printf ("**************Tm1*Tm2************\n");
  printMat(&result);
}
int
doTranstoCam1RefMark (visionData_struct rot1, CvMat* transFormMatrix, CvMat result3)
{
  CvMat T1;
  CvMat T2;
  double matheaderT1[4][4] =
    { rot1.rotRef[0][0], rot1.rotRef[0][1], rot1.rotRef[0][2], rot1.transRef[0],
      rot1.rotRef[1][0], rot1.rotRef[1][1], rot1.rotRef[1][2], rot1.transRef[1],
      rot1.rotRef[2][0], rot1.rotRef[2][1], rot1.rotRef[2][2], rot1.transRef[2],
      0, 0, 0, 1
    };
  cvInitMatHeader (&T1, 4, 4, CV_64FC1, matheaderT1, CV_AUTO_STEP);
  cvMatMul (transFormMatrix, &T1,  &result3);
  //printf ("**************Trans to refMark***\n");
  //printMat(&result3);
}
int
doTranstoCam0TrackMark (visionData_struct rot2, CvMat* transFormMatrix1, CvMat result1)
{
  CvMat T11;
  int i;
  double matheaderT11[4][4] =
    { rot2.rot[0][0], rot2.rot[0][1], rot2.rot[0][2], rot2.trans[0],
      rot2.rot[1][0], rot2.rot[1][1], rot2.rot[1][2], rot2.trans[1],
      rot2.rot[2][0], rot2.rot[2][1], rot2.rot[2][2], rot2.trans[2],
      0, 0, 0, 1
    };
  //printMat(transFormMatrix1);//Tog bort & som stod innan, &transFromMatrix1, till transFormMatrix
  cvInitMatHeader (&T11, 4, 4, CV_64FC1, matheaderT11, CV_AUTO_STEP);
  cvMatMul (transFormMatrix1,&T11,  &result1);
  //printf ("**************Trans to trackMark***\n");
 // printMat(&result1);
  /*        countEval++;
              txt=CV_MAT_ELEM(result1, double , 0,3); //FOR EVALUATION
              tyt=CV_MAT_ELEM(result1, double , 1,3);
              tzt=CV_MAT_ELEM(result1, double , 2,3);
   fprintf(eval,"%s%d%s %f %f %f%s\n",  //SK 3
                     "x220{",countEval,"}=[",
                     txt,tyt,tzt,"];");
  if(countEval==500){
  fclose(eval);
  }
  */
}
int
trans2Marker2Markerrelation (visionData_struct rot1, CvMat transFormMatrix, CvMat result, int indexCam)
{
  CvMat transRef;
  CvMat transMark;
  CvMat x2;
  int i;
  double resultInit[4][4];
  double resultInit1[4][4];
  CvMat inv1 = cvMat (4, 4, CV_64FC1, resultInit);
```

```
  CvMat result1 = cvMat (4, 4, CV_64FC1, resultInit1);
  double matheaderRefTrans[4][4] =
    { rot1.rotRef[0][0], rot1.rotRef[0][1], rot1.rotRef[0][2], rot1.transRef[0],
      rot1.rotRef[1][0], rot1.rotRef[1][1], rot1.rotRef[1][2], rot1.transRef[1],
      rot1.rotRef[2][0], rot1.rotRef[2][1], rot1.rotRef[2][2], rot1.transRef[2],
      0, 0, 0, 1
    };
  double matheaderMarkTrans[4][4] =
    { rot1.rot[0][0], rot1.rot[0][1], rot1.rot[0][2], rot1.trans[0],
      rot1.rot[1][0], rot1.rot[1][1], rot1.rot[1][2], rot1.trans[1],
      rot1.rot[2][0], rot1.rot[2][1], rot1.rot[2][2], rot1.trans[2],
      0, 0, 0, 1
    };
  cvInitMatHeader (&transRef, 4, 4, CV_64FC1, matheaderRefTrans, CV_AUTO_STEP);
  cvInitMatHeader (&transMark, 4, 4, CV_64FC1, matheaderMarkTrans, CV_AUTO_STEP);
  cvMatMul (&transFormMatrix,&transRef, &result1);
  printf ("**********Trans to RefMark***Cam%d\n",indexCam);
  printMat(&result1);
  cvMatMul (&transFormMatrix,&transMark, &result1);
  printf ("**********Trans to TrackMark***Cam%d\n",indexCam);
  printMat(&result1);
  cvInvert (&transMark, &inv1, CV_LU);
  cvMatMul (&inv1,&transRef, &result);
  printf ("**Trans to refMark from tracMark****Cam%d\n",indexCam);
  printMat(&result);
}
int
trans2Marker2MarkerrelationBeteen2Cams(visionData_struct rot1, visionData_struct rot2, CvMat transFormMatrix, CvMat result, int indexCam)
{
  CvMat transRef;
  CvMat transMark;
  CvMat x2;
  int i;
  double resultInit[4][4];
  double resultInit1[4][4];
  CvMat inv1 = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat inv2 = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat result1 = cvMat (4, 4, CV_64FC1, resultInit1);
  CvMat result2 = cvMat (4, 4, CV_64FC1, resultInit1);
  double matheaderRefTrans[4][4] =
    { rot1.rotRef[0][0], rot1.rotRef[0][1], rot1.rotRef[0][2], rot1.transRef[0],
      rot1.rotRef[1][0], rot1.rotRef[1][1], rot1.rotRef[1][2], rot1.transRef[1],
      rot1.rotRef[2][0], rot1.rotRef[2][1], rot1.rotRef[2][2], rot1.transRef[2],
      0, 0, 0, 1
    };
  double matheaderMarkTrans[4][4] =
    { rot2.rot[0][0], rot2.rot[0][1], rot2.rot[0][2], rot2.trans[0],
      rot2.rot[1][0], rot2.rot[1][1], rot2.rot[1][2], rot2.trans[1],
      rot2.rot[2][0], rot2.rot[2][1], rot2.rot[2][2], rot2.trans[2],
      0, 0, 0, 1
    };
  cvInitMatHeader (&transRef, 4, 4, CV_64FC1, matheaderRefTrans, CV_AUTO_STEP);
  cvInitMatHeader (&transMark, 4, 4, CV_64FC1, matheaderMarkTrans, CV_AUTO_STEP);
  cvMatMul (&transFormMatrix,&transRef, &result1);
  printf ("**********Trans to RefMark***Cam%d\n",indexCam);
  printMat(&result1);
  printf ("**********Trans to TrackMark***Cam0\n");
  printMat(&transMark);
  //cvInvert (&transMark, &inv1, CV_LU);
  //cvMatMul (&inv1,&result1, &result);
  cvInvert (&result1, &inv1, CV_LU);//ORIGINAL
  cvMatMul (&inv1,&transMark, &result);//ORIGINAL
  printf ("**Trans to refMark from tracMark****Cam%d\n",indexCam);
  printMat(&result);
  /*          countEval++;
              txt=CV_MAT_ELEM(result, double , 0,3); //FOR EVALUATION
              tyt=CV_MAT_ELEM(result, double , 1,3);
              tzt=CV_MAT_ELEM(result, double , 2,3);
   fprintf(eval,"%s%d%s %f %f %f%s\n",  //SK 1
                  "x140{",countEval,"}=[",
                  txt,tyt,tzt,"];");
  if(countEval==500){
  fclose(eval);
  }
  */
}
int
mark2Trans2mark1 (visionData_struct rot1, visionData_struct rot2, CvMat result)
{
  CvMat Ta;
  CvMat Tb;
  CvMat RaI;
  CvMat RbI;
  double resultInit[4][4];
  CvMat Tm1 = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat Tm2 = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat result1 = cvMat (4, 4, CV_64FC1, resultInit);
  CvMat result2 = cvMat (4, 4, CV_64FC1, resultInit);
  double matheaderTm1[4][4] =
    { rot1.rotRef[0][0], rot1.rotRef[0][1], rot1.rotRef[0][2], rot1.transRef[0],
      rot1.rotRef[1][0], rot1.rotRef[1][1], rot1.rotRef[1][2], rot1.transRef[1],
      rot1.rotRef[2][0], rot1.rotRef[2][1], rot1.rotRef[2][2], rot1.transRef[2],
      0, 0, 0, 1
    };
  double matheaderTm2[4][4] =
    { rot2.rot[0][0], rot2.rot[0][1], rot2.rot[0][2], rot2.trans[0],
      rot2.rot[1][0], rot2.rot[1][1], rot2.rot[1][2], rot2.trans[1],
      rot2.rot[2][0], rot2.rot[2][1], rot2.rot[2][2], rot2.trans[2],
      0, 0, 0, 1
    };
  cvInitMatHeader (&Tm1, 4, 4, CV_64FC1, matheaderTm1, CV_AUTO_STEP);
  cvInitMatHeader (&Tm2, 4, 4, CV_64FC1, matheaderTm2, CV_AUTO_STEP);
}
```

D. rbotClient.c

Source code for direct commands to the nxt.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //socket lib
#include <sys/types.h>
#include <bluetooth/bluetooth.h> //bluettoh lib
#include <bluetooth/rfcomm.h>
#define max_message_size 59
//gcc -lbluetooth -o nxtTrack nxtTrack.c
int nxtsocket; //on declare la socket
void init_bluetooth(char *btAddress)
{
struct sockaddr_rc addr={0};//structure du type sockaddr_rc
int status;
nxtsocket = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
addr.rc_family = AF_BLUETOOTH;
addr.rc_channel = (uint8_t) 1;
str2ba(btAddress, &addr.rc_bdaddr);// convertit un string en adresse
status = connect(nxtsocket, (struct sockaddr *)&addr, sizeof(addr) );
//connection
}
void nxt_forward_x(void)
{
char cmd1[15]={0x0D, 0x00, 0x00 ,0x04, 0x0, 0xA ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor A
write(nxtsocket, cmd1, 15); //send PlayTone command
char cmd3[15]={0x0D, 0x00, 0x00 ,0x04, 0x2, -0xC ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor C
write(nxtsocket, cmd3, 15); //send PlayTone command
}
void nxt_forward_y(void)
{
char cmd1[15]={0x0D, 0x00, 0x00 ,0x04, 0x0, 0xA ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor A
write(nxtsocket, cmd1, 15); //send PlayTone command
char cmd3[15]={0x0D, 0x00, 0x00 ,0x04, 0x1, -0xA ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor B
write(nxtsocket, cmd3, 15); //send PlayTone command
}
void nxt_forward_y2(void)
{
char cmd1[15]={0x0D, 0x00, 0x00 ,0x04, 0x1, 0xA ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor A
write(nxtsocket, cmd1, 15); //send PlayTone command
char cmd3[15]={0x0D, 0x00, 0x00 ,0x04, 0x2, -0xA ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor B
write(nxtsocket, cmd3, 15); //send PlayTone command
}
void nxt_sprint(void)
{
char cmd1[15]={0x0D, 0x00, 0x00 ,0x04, 0x0, 0x14 ,0x01 ,0x00,0x00, 0x00, 0x4D, 0x2, 0x00,0x00,0x00};//setting output to motor A
write(nxtsocket, cmd1, 15); //send PlayTone command
char cmd3[15]={0x0D, 0x00, 0x00 ,0x04, 0x2, -0x14 ,0x01 ,0x00,0x00, 0x00, 0x4D, 0x2, 0x00,0x00,0x00};//setting output to motor C
write(nxtsocket, cmd3, 15); //send PlayTone command
}
void nxt_stop(void)
{
char cmd1[15]={0x0D, 0x00, 0x00 ,0x04, 0xFF, 0x00 ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor ABC
write(nxtsocket, cmd1, 15); //send PlayTone command
//char cmd3[15]={0x0D, 0x00, 0x00 ,0x04, 0x2, 0x00 ,0x01 ,0x00,0x00, 0x00, 0x00, 0x00, 0x00,0x00,0x00};//setting output to motor C
//write(nxtsocket, cmd3, 15); //send PlayTone command
write(nxtsocket, cmd1, 15); //send PlayTone command
}
int main (void) {
FILE *nxtDataCmd0, *nxtDataCmd1, *nxtDataCmd2, *nxtDataCmd3, *nxtDataCmd4, *nxtDataCmd5, *nxtDataCmd6, *nxtDataCmd7;
int blevel, cmd0 = 1, cmd1 = 1, sprint1Done=1, cmd2= 1, cmd3=1, cmd4=1, cmd5=1, cmd6=1, cmd7=1;
char buffLoadData1[512];
char buffLoadData2[512];
char buffLoadData3[512];
char buffLoadData4[512];
char buffLoadData5[512];
char buffLoadData6[512];
char buffLoadData7[512];
char buffLoadData8[512];
char buff1[1024];
char buff2[1024];
char buff3[1024];
char buff4[1024];
char buff5[1024];
char buff6[1024];
char buff7[1024];
char buff8[1024];
sprintf(buffLoadData1,"%s","nxtDataCmd0.txt");
sprintf(buffLoadData2,"%s","nxtDataCmd1.txt");
sprintf(buffLoadData3,"%s","nxtDataCmd2.txt");
sprintf(buffLoadData4,"%s","nxtDataCmd3.txt");
sprintf(buffLoadData5,"%s","nxtDataCmd4.txt");
sprintf(buffLoadData6,"%s","nxtDataCmd5.txt");
sprintf(buffLoadData7,"%s","nxtDataCmd6.txt");
sprintf(buffLoadData8,"%s","nxtDataCmd7.txt");
//int cmd1 = atof(buff);
//fscanf( nxtData, "%s",buff);
//int cmd2 = atof(buff);
char btaddress[18] = "00:16:53:0A:30:F5"; //NXT adress
init_bluetooth(btaddress);
nxtDataCmd0=fopen(buffLoadData1,"wt");
nxtDataCmd1=fopen(buffLoadData2,"wt");
nxtDataCmd2=fopen(buffLoadData3,"wt");
nxtDataCmd3=fopen(buffLoadData4,"wt");
nxtDataCmd4=fopen(buffLoadData5,"wt");
nxtDataCmd5=fopen(buffLoadData6,"wt");
nxtDataCmd6=fopen(buffLoadData7,"wt");
nxtDataCmd7=fopen(buffLoadData8,"wt");
//Reseting the commands before run
fseek ( nxtDataCmd0 , 0 , SEEK_SET );
fprintf(nxtDataCmd0,"%s","0");
fseek ( nxtDataCmd1 , 0 , SEEK_SET );
fprintf(nxtDataCmd1,"%s","0");
fseek ( nxtDataCmd2 , 0 , SEEK_SET );
fprintf(nxtDataCmd2,"%s","0");
fseek ( nxtDataCmd3 , 0 , SEEK_SET );
fprintf(nxtDataCmd3,"%s","0");
fseek ( nxtDataCmd4 , 0 , SEEK_SET );
fprintf(nxtDataCmd4,"%s","0");
fseek ( nxtDataCmd5 , 0 , SEEK_SET );
fprintf(nxtDataCmd5,"%s","0");
fseek ( nxtDataCmd6 , 0 , SEEK_SET );
fprintf(nxtDataCmd6,"%s","0");
fseek ( nxtDataCmd7 , 0 , SEEK_SET );
fprintf(nxtDataCmd7,"%s","0");
fclose(nxtDataCmd0);
fclose(nxtDataCmd1);
fclose(nxtDataCmd2);
```

```
fclose(nxtDataCmd3);
fclose(nxtDataCmd4);
fclose(nxtDataCmd5);
fclose(nxtDataCmd6);
fclose(nxtDataCmd7);
while(1){
nxtDataCmd0=fopen(buffLoadData1,"r");
nxtDataCmd1=fopen(buffLoadData2,"r");
nxtDataCmd2=fopen(buffLoadData3,"r");
nxtDataCmd3=fopen(buffLoadData4,"r");
nxtDataCmd4=fopen(buffLoadData5,"r");
nxtDataCmd5=fopen(buffLoadData6,"r");
nxtDataCmd6=fopen(buffLoadData7,"r");
nxtDataCmd7=fopen(buffLoadData8,"r");
fscanf( nxtDataCmd0, "%s ",buff1);
//printf("Cmd0:%d\n",(int)atof(buff1)); //Debug printing
if( (int)atof(buff1)==1 && cmd0==1){
nxt_stop();
cmd0=0;
printf("Motors stopped for x Cam0\n");
}
fscanf(nxtDataCmd1, "%s",buff2);
//printf("Cmd1:%d\n",(int)atof(buff2)); //Debug printing
if( (int)atof(buff2)==1 && cmd1==1){
nxt_forward_x();
cmd1=0;
printf("Motors started for x Cam0\n");
fseek ( nxtDataCmd1 , 0 , SEEK_SET );
fprintf(nxtDataCmd1,"%s","0");
}
fscanf(nxtDataCmd2, "%s",buff3);
//printf("Cmd1:%d\n",(int)atof(buff2)); //Debug printing
if( (int)atof(buff3)==1 && cmd2==1){
nxt_forward_y();
cmd2=0;
printf("Motors started for y Cam0 \n");
fseek ( nxtDataCmd2 , 0 , SEEK_SET );
fprintf(nxtDataCmd2,"%s","0");
}
fscanf(nxtDataCmd3, "%s",buff4);
//printf("Cmd1:%d\n",(int)atof(buff2)); //Debug printing
if( (int)atof(buff4)==1 && cmd3==1){
nxt_stop();
cmd3=0;
printf("Motors stopped for y Cam0\n");
fseek ( nxtDataCmd3 , 0 , SEEK_SET );
fprintf(nxtDataCmd3,"%s","0");
}
if(cmd3==0 && cmd2==0 && sprint1Done==1){
nxt_sprint();
printf("sprinting to Cam 1\n");
sprint1Done=0;
}
fscanf( nxtDataCmd4, "%s ",buff5);
//printf("Cmd0:%d\n",(int)atof(buff1)); //Debug printing
if( (int)atof(buff5)==1 && cmd4==1){
nxt_forward_x();
cmd4=0;
printf("Motors started for x Cam1 \n");
}
fscanf( nxtDataCmd5, "%s ",buff6);
//printf("Cmd0:%d\n",(int)atof(buff1)); //Debug printing
if( (int)atof(buff6)==1 && cmd5==1){
nxt_stop();
cmd5=0;
printf("Motors stopped for x Cam1 \n");
}
fscanf( nxtDataCmd6, "%s ",buff7);
//printf("Cmd0:%d\n",(int)atof(buff1)); //Debug printing
if( (int)atof(buff7)==1 && cmd6==1){
nxt_forward_y2();
cmd6=0;
printf("Motors started for x Cam1\n");
}
fscanf( nxtDataCmd7, "%s ",buff8);
//printf("Cmd0:%d\n",(int)atof(buff1)); //Debug printing
if( (int)atof(buff8)==1 && cmd7==1){
nxt_stop();
cmd7=0;
printf("Motors stopped for y Cam1\n");
}
fclose(nxtDataCmd0);
fclose(nxtDataCmd1);
fclose(nxtDataCmd2);
fclose(nxtDataCmd3);
fclose(nxtDataCmd4);
fclose(nxtDataCmd5);
fclose(nxtDataCmd6);
fclose(nxtDataCmd7);
}
//open connection
close(nxtsocket);
//direct command
return 0;
}
```

E. Calib.m

The calculation for the essential matrix between two cameras.

```
lengthOfIndex=length(indexes);
clear rotnew;
t=length(rotnew);
yplot=[1:3];%% 21 22 23 31 32 33];
r11=0;
r12=0;
r13=0;
r21=0;
r22=0;
r23=0;
r31=0;
r32=0;
r33=0;
tx=0;
ty=0;
tz=0;
for i=1:500;
    Cam0Index=indexes{i}(1,1);
    Cam1Index=indexes{i}(1,2);
    Compare=rotCompare{i};
    temp{i} = [Cam0{Cam0Index}(1:3,1:3)*(Cam1{Cam1Index}(1:3,1:3))];
    Result=temp{i};
    r11= r11+Result(1,1);
    r12= r12+Result(1,2);
    r13= r13+Result(1,3);
    r21= r21+Result(2,1);
    r22= r22+Result(2,2);
    r23= r23+Result(2,3);
    r31= r31+Result(3,1);
    r32= r32+Result(3,2);
    r33= r33+Result(3,3);
    tx=tx+x{camIndex{i}}(1:1,4:4);
    ty=ty+x{camIndex{i}}(2:2,4:4);
    tz=tz+x{camIndex{i}}(3:3,4:4);
end
count;
 %%Remove the the -abs() !!!to check signs and ad abs() after and - if needed :)
transformMat=[(r11/t) (r12/t) r13/t tx/t;
            (r21/t) (r22/t) (r23/t) ty/t;
            (r31/t) (r32/t) r33/t tz/t;
             0 0 0 1]
%buffer=char('CalibDataCamera0');
buffer=sprintf('CalibDataCamera0%s.txt',tempString);
file_1 = fopen(buffer,'wt');
fprintf(file_1,' %8.7f',transformMat(1,1));
fprintf(file_1,' %8.7f',transformMat(1,2));
fprintf(file_1,' %8.7f',transformMat(1,3));
fprintf(file_1,' %8.7f',transformMat(2,1));
fprintf(file_1,' %8.7f',transformMat(2,2));
fprintf(file_1,' %8.7f',transformMat(2,3));
fprintf(file_1,' %8.7f',transformMat(3,1));
fprintf(file_1,' %8.7f',transformMat(3,2));
fprintf(file_1,' %8.7f',transformMat(3,3));
fprintf(file_1,' %8.7f',transformMat(1,4));
fprintf(file_1,' %8.7f',transformMat(2,4));
fprintf(file_1,' %8.7f',transformMat(3,4));
fclose(file_1);
```

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
March 2013

Document Number
ISRN LUTFD2/TFRT--5914--SE

Author(s)
Mikael Borg

Supervisor
Vladimeros Vladimerou, Dept. of Automatic Control, Lund University, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Overhead Localization of Mobile Robots

Abstract

The objective of this thesis has been to develop a system that detects a mobile robot with tracking and controlling its position with the help of one or more cameras. This has been done by implementing different algorithms that are able to detects markers. By using several cameras one can track over a bigger area and estimate the markers position with better precision by using triangulation where the marker has been detected by at least two cameras. A tracking marker is mounted on a robot which enables the camera to detect the robot and a reference marker is placed on the ground in order for the mobile robot to move towards the reference marker. When the tracking and reference marker has been detected the direct commands are then sent to the mobile robot for controlling its position. Topics that are of interest in this report are image analysis in general, image analysis algorithms, camera calibration, different libraries for implementation, Lego mindstorms NXT, sockets TCP/UDP, bluetooth.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-53

Recipient's notes

Security classification