

# Lego Mindstorms NXT Camera

Semester thesis by Leo den Hartog

1. Advisor: Wolfgang Haid
2. Advisor: Matthias Woehrle

Professor: Prof. Dr. Lothar Thiele

8/31/2008



**Abstract**

In this semester thesis, an embedded camera is integrated into the Lego Mindstorms set. Two different cameras are evaluated and the CMUCam3 is chosen. The camera is connected to the Lego Mindstorms NXT through a self-made connection board. A communication protocol is established and implemented on the different interfaces. The system as a whole is evaluated through a prototype robot, which was programmed to recognize one or two balls of specific colors. All requirements are met and the system described in this semester thesis is fully functional.



## Contents

<b>1. Introduction .....</b>	<b>7</b>
1.1. Overview .....	7
1.2. Goals .....	8
1.3. Requirements .....	8
1.4. Related work .....	8
<b>2. System Overview .....</b>	<b>9</b>
2.1. Lego Mindstorms background .....	9
2.2. Lego Mindstorms NXT brick .....	10
2.3. Connection board and communication protocol .....	10
2.4. Camera .....	11
2.5. Conclusion .....	11
<b>3. Evaluation of Cameras .....</b>	<b>13</b>
3.1. CMUCam3 .....	14
3.2. LeanXcam .....	15
3.3. Decision .....	15
<b>4. Implementation .....</b>	<b>17</b>
4.1. Components .....	17
4.1.1. Connection board .....	17
4.1.2. Communication protocol .....	17
4.1.3. Image processing .....	18
4.2. Facts .....	19
4.2.1. I <sup>2</sup> C Protocol .....	19
4.2.2. RS-232 Protocol .....	19
4.2.3. NXT – I <sup>2</sup> C .....	20
4.2.4. PIC – I <sup>2</sup> C .....	22
4.2.5. PIC – RS-232 .....	25
4.2.6. CMUCam3 – RS-232 .....	27
4.2.7. CMUCam3 – Image processing .....	28
4.3. Implementation .....	29
4.3.1. Connection board .....	29

4.3.2. Communication protocol.....	30
4.4. Debugging .....	31
<b>5. Evaluation .....</b>	<b>33</b>
5.1. Object recognition .....	34
5.1.1. Implementation .....	34
5.1.2. Problems and Conclusion .....	35
5.2. Evaluation – Move to red ball .....	36
5.3. Evaluation – Distinguish red and blue ball .....	36
<b>6. Conclusion.....</b>	<b>37</b>
6.1. Future Work.....	37
<b>7. Appendix.....</b>	<b>39</b>
A. Code .....	39
A.1. CMUCam3 firmware .....	39
A.2. Connection board firmware.....	42
A.3. NXT Evaluation – Move to red ball.....	47
A.4. NXT Evaluation – Distinguish red and blue ball .....	50
B. Assignment.....	53
<b>8. References.....</b>	<b>59</b>

## 1. Introduction

For several years, the Computer Engineering and Network Laboratory at the ETH has been working with students on building and programming small robots using the Lego Mindstorms set. Several projects have been realized on the system [1], but they were always limited due to the lack of a camera. Although there exists an ultrasound sensor for distance measurements and a simple light sensor that can detect differences in brightness, a camera with image processing functionalities has not been implemented. The main idea of this semester thesis is to integrate a third-party camera into the Lego Mindstorms set in a way that it can be used in the same way as the existing sensors. This would greatly improve the potential of the existing set for future projects.

The reason to choose this semester thesis is that it included all aspects from a hardware implementation up to the very interesting topic of computer vision. After years of theoretical courses it felt right to complete a practical project from the design concept all the way to the final implementation and deployment.

### 1.1. Overview

This semester thesis is about integrating a commercial, embedded camera into the Lego Mindstorms set. For this, a connection board was built to enable the camera to communicate with the NXT, the Lego Mindstorms Processor, as shown in Fig.1. A microcontroller on the board is used to connect the different interfaces of the NXT and the camera and establish a simple communication protocol. Both interfaces have been adapted in a way that the camera is implemented as a fully functional, additional sensor to the existing set, without adding any additional constraints to the design flexibility.

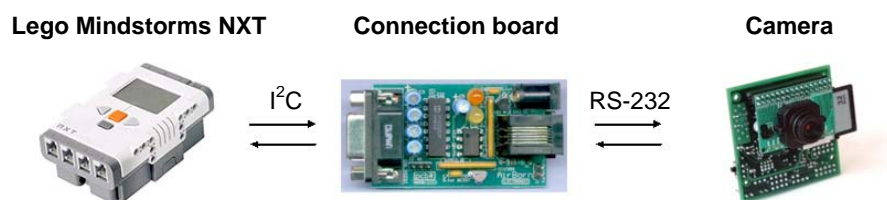


Figure 1: Concept

Evaluation of the system has been done by enabling a robot to detect a ball with a certain color on a white table and point the camera directly at it. For the evaluation, a library on the camera for object recognition based on color has been implemented and the functionality of the connection board has been fully proven. The evaluation has shown that through solely adapting the cameras existing libraries, it is feasible to perform all basic robotic tasks with the designed solution.

## 1.2. Goals

The goal of this semester thesis is the implementation of a camera into the Lego Mindstorms set to enable NXT-based robots to perform object recognition. The project should enhance the Lego Mindstorms set and implement the camera in a way that it fits into the existing set. Further should it be possible to easily program the camera to perform all basic image processing tasks for maximized flexibility in future projects.

According to the assignment [Appendix B], this consists of three main goals:

- Design and implementation of the hardware interface between the camera platforms and the NXT
- Design and implementation of the software libraries for obtaining data from the camera (both on the camera platforms and the NXT)
- Implementation of software libraries for typical tasks to be performed by the camera when used in conjunction with the NXT

## 1.3. Requirements

The camera has to be implemented into the set on the same programming environment as is used today. Additional libraries can be integrated, but no changes should be made that narrow the present design flexibility regarding programming and use of sensors. No additional constraints regarding power consumption should be tolerated. Sensors already in use with the Lego Mindstorms set should not be influenced by the integration of the camera.

It should be possible to easily program the camera, in order to be able to adapt any existing library to the needs of a specific application. Some basic image processing tasks should already be implemented or at least should it be feasible to integrate other libraries into an existing camera framework.

## 1.4. Related work

An existing integration of a camera into the Lego Mindstorms set is the vision subsystem [21]. It connects a preprogrammed camera to a sensor port and enables object tracking based on color. It would fulfill the needs for the evaluation, but because it is not feasible to change the image processing functionalities it does not meet the requirements for programmability.

One open-source implementation of a camera into robotics is the Surveyor SRV-1 [22]. It applies a fully programmable camera to a small robot and controls the robot through its own processor.



## 2. System Overview

The main task is to implement the camera into the system in a way, that further projects can use it similar to all other sensors. Due to the limited processing power and insufficient communication bandwidth of the I<sup>2</sup>C port on the Lego Mindstorms set, all image processing needs to be done on the camera. Other connection possibilities, like Bluetooth, RS485 or USB, are available on the NXT, but because I<sup>2</sup>C is the Lego Mindstorms standard protocol for digital sensors, the camera should optimally be connected in a similar way. This implies creating a hardware bridge from the I<sup>2</sup>C port to a port compatible with the camera. In order to be flexible in the design of the communication protocol, the connection board should be self-made rather than bought as an existing solution. By adapting the interface on the camera as well as on the Lego Mindstorms system, this hardware setup allows an implementation of a simple message protocol through the connection board.

The different tasks of programming the interfaces as well as the firmware for the hardware bridge are independent, which has the beneficial effect, that every subtask can be handled and evaluated separately.

### 2.1. Lego Mindstorms background

Lego Mindstorms has been used at the Computer Engineering and Networks Laboratory for several years in a PPS project for first year students. On top of that, a few semester theses [2,3] have been written on evaluating or enhancing the existing system sold by Lego.

The core of the Lego Mindstorms set is a programmable Lego brick called NXT, depicted in Fig.2, which is battery powered and connects to the PC via Bluetooth or USB. It has three ports for connecting motors and another four ports to which sensors can be connected. The NXT can be programmed via the standard LabView software by Lego, but due to the very open design there exist also various other programming environments for the NXT. The PPS uses a programming language called NXC (Not eXactly C) [1], which is a high level, C-like language for the original NXT firmware. The connection between the PC and the NXT Brick is handled by a program called BricxCC, which also compiles NXC programs.



Figure 2: Lego Mindstorms NXT Brick

In addition to the NXT, Lego sells various sensor types and motors compatible with the NXT. All original sensors can be interfaced by LabView as well as NXC, which allows an easy integration into the system. For further details on the sensors, the reader is referred to a Semester Thesis by Claudia Frischknecht and Thomas Other [2].

## 2.2. Lego Mindstorms NXT

The NXT supports several different protocols, of which we choose I<sup>2</sup>C [23] for maximal flexibility and most stable implementation. Another option would be the RS-485 port, but the NXT driver of that port is not yet implemented in the NXT set. Since I<sup>2</sup>C is the Lego Mindstorms standard protocol for communication with digital sensors, this facilitates further work.

The software library of the NXT should be adapted in a way to allow access to the camera with simple commands similar to using existing sensors. From the latter concern, we conclude that the only reasonable choice for a programming platform is NXC, which is presently used in the PPS, rather than switching the entire system to a different programming environment. The newest version of NXC has even implemented the standard functions for I<sup>2</sup>C communication, since it is also used by the software to communicate with the digital ultrasonic sensor. So the communication interface on the NXT side is a matter of initializing an I<sup>2</sup>C port and sending the packets with the appropriate I<sup>2</sup>C -address and the right setting for the amount of bytes expected of the answer.

## 2.3. Connection board and communication protocol

The connection board is used to connect the camera to the NXT and to implement a communication protocol that can be designed independent of existing protocols. One way to implement this hardware bridge is through the use of a microcontroller, which supports both protocols and can be programmed to relay the packets between the two. Through LEDs and possibly an interface that can connect to a PC, the board should be able to offer some debugging possibilities for initial programming.

Due to hardware constraints, all image processing is done on the camera, so only a very limited amount of data needs to be sent. Therefore the communication protocol should be kept rather simple. The idea is to implement a protocol, that implements a basic request and answer functionality, possibly with single byte packages. With the NXT as the core of the system, it polls the camera for data rather than receiving a constant data stream, which is more efficient for processed image information. Due to the I<sup>2</sup>C requirement of an address byte, the request will consist of at least two bytes, but the microcontroller can interpret the address byte and cut it off. This design allows for all basic tasks to be accomplished, assuming that the library on the camera is adapted accordingly and all communication is initiated by the NXT brick.

## 2.4. Camera

The camera should enable the Lego Mindstorms set to perform basic image processing functionalities like object recognition. For that the camera should be easily programmable and its image processing functions should be adaptable to the Lego Mindstorms set. A variety of libraries should already exist on the camera, so the major task is to integrate a communication concept in a way that is adaptable for all the libraries. All image processing is done on the camera, so the camera should have the processing power to perform those tasks locally. For best use in the Lego Mindstorms project, the camera should be based on an open-source system, in order to be able to adapt all functions to the specific environment. The transmission should be done through a protocol, which can be implemented into the communication board. So the general implementation would need to react to requests from the NXT by sending information, decoded into packets according to the communication protocol.

For our evaluation task, we need a library on the camera that can recognize objects based on color and send the coordinates of the objects to the NXT. Color tracking is implemented in most standard graphic libraries, so it could be easily integrated into any framework if not available by default.

## 2.5. Conclusion

After evaluating two different cameras, the CMUCam3 [5] and the LeanXcam [6], we decided to use the CMUCam3, a camera developed for use in embedded systems. The choice was made for the CMUCam3 mainly because it has several graphic libraries already implemented. The LeanXcam [6] generally is performance-wise the superior camera, but it lacked support for general graphics applications and it does not support an appropriate communication protocol. The evaluation is discussed in detail in chapter 3.

In order to be able to connect the CMUCam3 to the NXT, a bridge between a RS-232 port on the camera and an I<sup>2</sup>C port on the NXT is needed. The solution was a connection board with a PIC16F690 microcontroller at its core, which was programmed to handle the translation between the protocols. The communication protocol between the camera and the NXT brick was kept deliberately simple. The NXT has been adapted to send requests to the camera, coded into one byte instructions. To interpret these packets, the software of the CMUCam3 has been adapted to send the coordinates of recognized objects, also in the form of one byte packages, to the NXT on request. The standard graphics library on the CMUCam3 is able to perform all general image processing tasks. The final implementation of the system is explained in chapter 4.

Finally, the system was evaluated through a simple robot, which was intended to recognize one or more balls of different colors. The implementation enables color tracking on the base of predefined RGB-colors through the adaption of an existing library for the communication protocol. This test was designed to act as an example for the further use of the camera as well as a measurement for the overall performance of the system, which is discussed in chapters 5. Finally, the semester thesis is concluded in chapter 6.



### 3. Evaluation of Cameras

The main requirements for the cameras are the possibilities of easily programming the camera and the feasibility of integrating it into the Lego Mindstorms system. There are of course some constraints regarding size, power consumption and prize, which led to the choice of the two cameras evaluated in this chapter.

The goal of this semester thesis is to integrate a camera into a robotics system, aimed at building small to medium sized robots. This only makes sense, if the camera can operate autonomously regarding processing as well as power. Power sources are available either through the NXT with 4.3V or through several different battery kits ranging from 3V up to 12V. Regarding processing power, it is important to keep as much of the calculation on the camera rather than on the NXT. The NXT's processor is already very limited, and there is no sense in starting time expensive image processing on it, so the camera must be completely capable of handling the images itself.

Because none of the later cameras have the I<sup>2</sup>C protocol implemented by default, they need to be connected through a connection board to the NXT. So for use with the Lego Mindstorms set, the cameras need to have a port which is able to connect to a microcontroller as discussed in the design concept of the connection board.

Camera	CMUCam3	LeanXcam
Designed by	Carnegie Mellon University	Super Computing Systems
Operating system	Individual Firmware	Linux Microcontroller OS
Framework	C-based, GCC	C-based, GCC
Debugging	Leds, RS-232, local emulation	RS-232, Ethernet, JTAG
Communication	RS-232, unshifted RS-232	I <sup>2</sup> C (only master, Ethernet)
Resolution	352x288 pixels	752x480 pixels
Frame Rate	26 fps	60 fps
Color depth	8 bit	10 bit
Processor	ARM7TDMI	Blackfin ADSP-BF537
RAM	64 KB	64 MB SDRAM
Power supply	6-15 V; 150 mA	4.5 V; 700 mA

Table 1: Camera comparison

### 3.1. CMUCam3

The CMUCam3 [5,8] is a low-cost, embedded camera using a complete open-source system. It has been specifically designed for the use in small embedded systems as for example robotics. For programming, it can be connected to a PC through a RS-232 port and has as other ports one unshifted RS-232 (0V - 5V) and four servo ports, intended to control motors. A direct connection with the NXT is impossible, a separate connection board is necessary. The CMUCam3 works on 6V to 15V and needs at least 150mA, therefore an external power supply is required for use with the NXT brick.

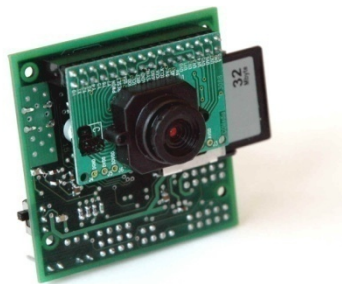


Figure 3: CMUCam3 - front

The CMUCam3 works on an ARM7TDMI processor with 64kb RAM and has a maximum resolution of 352x288 pixels, the standard image processing libraries work on 176x144 pixel though. In contrast to its non-programmable predecessor, which functioned as a preconfigured black-box with a standard interface, the CMUCam3 is designed in a complete open-source environment, in order to be able to adapt the camera to specific tasks. The software development framework of the camera consists of a C-based programming environment with several libraries already implemented. The collection of libraries includes functions such as face recognition, color tracking, free space localization and many more, which are all extensively tested through the wide spread use of the CMUCam3. The framework also includes tools for emulating the camera locally, with which debugging based on previously recorded images is possible. All functions needed for object recognition based on color are implemented in the standard graphic library, although only at a resolution of 176x144 pixels. Still, for basic object-recognition that resolution is sufficient.

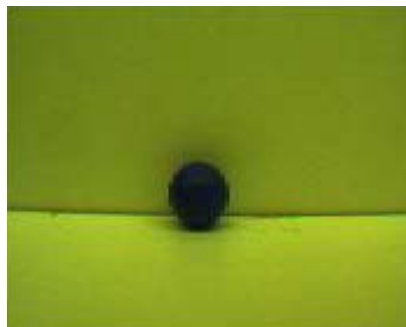


Figure 4: CMUCam3 - Sample Image

### 3.2. LeanXcam

The LeanXcam targets the same applications as the CMUCam3, but in a much more advanced configuration [6]. It runs a fully functional Linux on a Blackfin ADSP-BF537 processor with 64 MB SDRAM, which illustrates the difference to the CMUCam3 already quite impressively. Initial configuration of the camera is done via a RS-232 interface, but in normal operation it is intended to use an Ethernet port. On top of that, a special version we obtained for testing purposes is equipped with an I<sup>2</sup>C -Port working in master mode. Since the NXT brick also runs I<sup>2</sup>C only in master mode, that interface would only make sense with a special driver emulating a slave mode on the camera. A different option would be to make the RS-232 port accessible under Linux. In continuous capture mode, the LeanXcam uses up to 700mA at a voltage level of 4.5V. The NXT brick is not able to deliver that current, so the LeanXcam has to be powered externally.

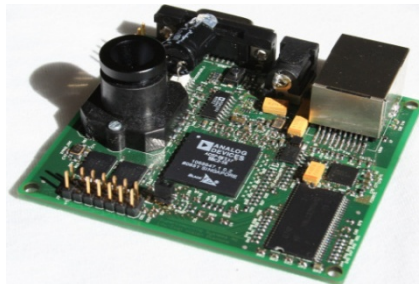


Figure 5: LeanXcam

On the LeanXcam, a C-based framework for implementing specific functions exists. The interface is configured through an Apache web server running on the camera and all preprogrammed libraries operate on an html interface. Since the LeanXcam is still in a development phase, no image processing libraries are implemented. Up to date, the only library available downloads pictures from the camera and outputs them on a webpage. Programming and debugging can be done in a pre-configured sandbox, which runs a regular Linux distribution, has the compiler and framework set up and is able to emulate all functions of the camera locally.

### 3.3. Decision

Programming some simple image processing functions in any C-based environment is nowadays a pretty straightforward task, therefore the missing libraries on the LeanXcam are not a problem. Connecting the cameras to the NXT brick is the big issue to be considered. For the CMUCam3, a RS-232 to I<sup>2</sup>C converter is needed. This is feasible with a PIC Microcontroller and has already been done before [8]. For the LeanXcam, on the other hand, a complete I<sup>2</sup>C driver including a port sensing capability would have to be implemented in order to connect the LeanXcam to the NXT brick. Thus we decided to use the CMUCam3 for this semester thesis. It includes all necessary functions on a simple platform that can be easily adapted to different applications.





## 4. Implementation

The implementation is done according to the design concept of chapter 2 as shown in Fig.6. In section 4.1, the main components are explained, with the specific facts of the individual implementations in section 4.2. The technical details on the connection board and the communication protocol are in section 4.3. In section 4.4, there is a description of the debugging environment used.



Figure 6: Components

### 4.1. Components

#### 4.1.1. Connection board

One main concern is the implementation of the two protocols, I<sup>2</sup>C and RS-232 on the PIC microcontroller. On the camera and NXT side, initialization and transmitting is implemented in the high-level programming language C, but on the PIC, all functions need to be programmed in assembler. Further on, debugging is quite difficult on low-level programming languages, because it has to be done on signal level rather than by a logical debugger. The protocols are integrated on the PIC, so no analog signal decoding is necessary, but due to several problems in the implementation, programming the firmware for the PIC is a major task of this semester thesis.

#### 4.1.2. Communication protocol

The main idea of the communication protocol is to keep it as simple as possible. In the concept, the constraints for the communication protocol are defined as a basic request and answer communication. Due to some delays in the transmission, several errors might emerge, like packet loss and sequence errors, which are of course not handled by the most basic version of a single byte request and answer protocol. Furthermore, there are some issues with the need of I<sup>2</sup>C to include addresses in master-slave communication, which need to be addressed in the microcontroller.

#### 4.1.3. Image processing

Although the CMUCam3 includes several standard libraries for image processing, integrating them into the Lego Mindstorms system is a main task. The camera was designed to act as the core of a robotics system itself, so all information needs to be further processed to adjust to the communication protocol. The main idea is to only send information to the NXT brick similar to the information that would be sent from the camera to the motors. This information needs to be processed on the camera according to the communication protocol and the use in the specific implementation for use with the NXT. So for the camera to be integrated, libraries would need to be modified to output information at a different level of the processing stages of the existing libraries.

4.2. Facts

4.2.1. I<sup>2</sup>C Protocol

The I<sup>2</sup>C [23] uses two lines for data transmission, SDA for data and SCL as clock, and needs a common ground level. I<sup>2</sup>C is a communication between a master and possible many slaves. One data line (SCL) is reserved as clock, which the master controls and keeps at high level. Every communication is initiated by the master through the clock level, whilst sending data parallel on the other data line (SDA). Depending on the data sent, it expects an answer and releases control of the clock. Figure 7 depicts a transmission of a three byte package.

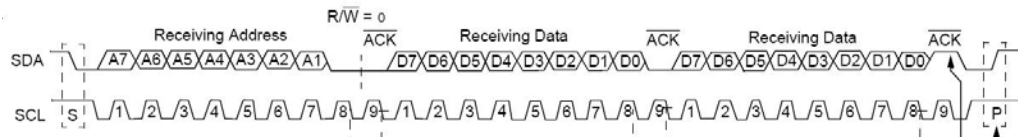


Figure 7 [14]: I<sup>2</sup>C Address and Data reception

4.2.2. RS-232 Protocol

The RS-232 protocol [24] is a communication protocol with a packet size of one byte. Communication is established through two unidirectional wires and a common ground level. One data wire is used for transmission (TX), the other for receiving (RX). A sender keeps its TX line at a low level and initiates a communication through an initial high start bit. The start bit is followed by 8 data bits and completed through a low stop bit, in Fig.8, such a data packet is shown. The sampling rate is fixed at a value between 2'400 and 115'200 bits/s. The voltage levels used high and low differ between the unshifted (low: 0 V; high +5 V) and the shifted version (low: -12 to -3 V; high: +3 to + 15 V). A standard serial connection to a PC uses shifted RS-232.

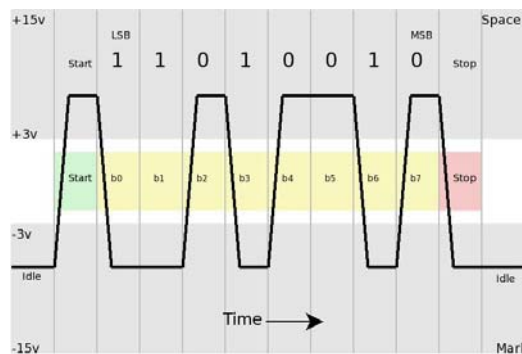


Figure 8 [17]: RS-232 Data packet

#### 4.2.3. NXT-I<sup>2</sup>C

According to the design concept and semester thesis goals, all functionality on the NXT brick should be integrated into the existing framework. The TIK Lego Mindstorms project uses the NXC programming language for all of their applications, so the communication interface should be primarily integrated into that system. The NXC language has implemented some I<sup>2</sup>C functionality in its newest version, so the main task was to implement an appropriate function for all communication tasks. In accordance with the communication concept, the communication function is set up to send one request byte and expects one byte as answer. The I<sup>2</sup>C address is hardwired into the setup, to minimize errors.

```

03: #define I2C_PORT S1

08: int err;
09: byte send[] = {0x40, 0xfe};
10: byte recv[] = {1};

74: SetSensorType(I2C_PORT, SENSOR_TYPE_LOWSPEED);
75: SetSensorMode(I2C_PORT, IN_MODE_RAW);
76: ResetSensor(I2C_PORT);

```

Listing 1: Initialization of I<sup>2</sup>C communication:

In Listing 1, line 3 defines on what port the camera is connected, where S1 refers to input port 1. Two variables are needed for communication, a third one is initialized for error checking purposes (Lines 8-10). Afterwards the I<sup>2</sup>C Port is initialized and reset to be ready for use.

```

20: sub sendrecvI2c (byte data)
21: {

24:     send[1] = data;

26:     while (I2CCheckStatus(I2C_PORT) != 0);
27:     err = I2CWrite(I2C_PORT, 1, send);

29:     while (I2CCheckStatus(I2C_PORT) != 0);

33:     while (I2CBytesReady(I2C_PORT) < 1);
34:     I2CRead(I2C_PORT, 1, recv);

38:     Wait(500);
39: }

```

Listing 2: Communication interface

The communication function, shown in List.2, is called `sendrecv2c(byte data)` and takes a single byte as input. On Line 24, it writes the input into the send buffer. Afterwards it checks if the port is ready and transmits the send buffer as soon as it is ready (Line 26-27). Then it checks whether the buffer is cleared on the receiving end and then waits for an answer to be transmitted to the buffer (lines 29-33). On line 34, the answer is written into the receive buffer.

This function can be implemented into any NXT program that does not use input port 1 for different purposes. The function is completely independent of what is actually sent. It can be used for receiving coordinates of an object, like in the code used for the evaluation of the whole system in chapter 5, but theoretically also for low-level commands like directions if an application requires all calculation to be done on the camera.

#### 4.2.4. PIC – I<sup>2</sup>C

On the PIC, two specific ports, RB4 and RB6, are assigned to I<sup>2</sup>C communication. They are initialized by configuring both ports as digital output.

```

08: #define          NODE_ADDR  0x40  ; I2C address,64 decimal

27: ;Switch to Register Bank 1
28: bsf              status,rp0

34: ;Set Port B0-B3 and B7 to output and B4-B6 to input
35: ;B4 and B6 used for I2C; B5 is RS-232 RX, B7 is TX
36: movlw   b'01110000'
37: movwf   TRISB
38: ;Switch to Register Bank 2
39: bcf     status,rp0
40: bsf     status,rp1
41: ;Set all ports to digital
42: movlw   0x00
43: movwf   ANSEL
44: movlw   0x00
45: movwf   ANSELH
46:
47: ;Osccon
48: movlw  B'01110001'
49: movwf  OSCCON
50: ;Comparator off
51: clrf   CM1CON0
52: clrf   CM2CON0
53: ;ECCP off
54: clrf   CCP1CON
55:
56: ;Switch to Register Bank 1
57:       bsf     status,rp0
58: bcf     status,rp1

63: ;ADC off
64: bcf     ADCON0, 0

77: ;Set i2c address
78: movlw   NODE_ADDR  ; set Node Address
79: movwf   SSPADD

82: ;Switch to Register Bank 0
83: bcf     STATUS,RP0

91: ;initialize i2c
92: movlw   b'00111001'      ; Setup SSP module for 7-bit
93: movwf   SSPCON           ; address, slave mode
94: movlw   b'00110110'      ; Setup SSP module for 7-bit
95: movwf   SSPCON           ; address, slave mode

```

Listing 3: Initialization of the I<sup>2</sup>C protocol

Listing 3 shows the initialization of the I<sup>2</sup>C protocol. In line 8, the I<sup>2</sup>C slave address of the PIC is set to 0x40. Lines 27 to 45 again initialize the port, with B4 being the data line (SDI) and B6 the clock (SCL). Both ports have to be configured as input because the PIC functions as slave, therefore the master controls both lines. In the event of a transmission, the ports are switched automatically. In lines 45 to 64, some general settings are disabled, for details on the individual setting refer to the PIC data sheet [14]. The I<sup>2</sup>C address is fixed by writing it to the SSPADD register. It is very important, that this is done before I<sup>2</sup>C initialization. After that, the I<sup>2</sup>C protocol is initialized. The reason for the first, seemingly useless, initialization is a bug in the PIC, which writes the address only to the appropriate register if initialized this way, no matter what mode is used later on [19].

```

150: WaitrxI2c
151: ;Write Status on output for debugging purposes
152: bsf          STATUS,RP0      ;Bank 1
153: movf  sspstat,w
154: bcf          STATUS,RP0      ;Bank 0
155: movwf PORTC

157: ;Wait for received byte on I2C
158: btfsc PIR1,sspif
159: goto  i2cnew
160: ;Repeat Loop
161: goto WaitrxI2c

164: i2cnew
165: banksel      SSPSTAT
166: movf  SSPSTAT,W  ; Get the value of SSPSTAT
167: bcf          STATUS,RP0      ;Bank 0
168: andlw b'00101101' ; Mask out unimportant bits
169: banksel      Temp          ; Put masked value in Temp
170: movwf Temp          ; for comparision checking.

172: State1          ; Handle address byte
173: movlw b'00001001'
174: xorwf Temp,W
175: btfss STATUS,Z    ; Address received?
176: goto  State2      ; No, handle data byte
177: banksel      SSPBUF
178: movf  SSPBUF,W    ; Get the addr and throw it away
179: bcf          pir1,sspif
180: ;Repeat Loop
181: goto WaitrxI2c

184: State2          ; Handle data byte
185: banksel      SSPBUF
186: movf  SSPBUF,W    ; Get the byte into W

198: bsf STATUS,RP0      ;Bank 1
199: btfsc SSPSTAT, 2    ;Check if answer is exp.
200: goto WriteI2C
201: bcf          STATUS,RP0      ;Bank 0
202: goto WaitrxI2c

```

Listing 4: Receive function

```

204: WriteI2C
205: banksel SSPSTAT
206: btfsc      SSPSTAT,BF      ; Is the buffer full?
207: goto  WriteI2C      ; Yes, keep waiting.
208: banksel SSPCON      ; No, continue.
209: DoI2CWrite
210: bcf      SSPCON,WCOL ; Clear the WCOL flag.

212: movwf      Delay0
213: movlw      h'03'
214: movwf      Delay1
215: DelayLoopSend2
216: decfsz     Delay0,f
217: goto      DelayLoopSend2
218: movlw      h'20'
219: movwf      Delay0
220: decfsz     Delay1,f
221: goto      DelayLoopSend2

224: movf  rx_data,w
225: movwf SSPBUF      ; Write the byte in WREG
226: btfsc SSPCON,WCOL ; Collision?
227: goto  DoI2CWrite
228: bsf  SSPCON,CKP      ; Release the clock.

254: bcf      pirl,sspif

```

Listing 5: Send function

Listings 4 and 5 are the I<sup>2</sup>C send and receive functions of the PIC. For debugging purposes, the Status register is put on the output port (Lines 150-155) in the waiting loop (157-161), in order to be able to check all status bits through an oscilloscope. In lines 164 to 170, the status register is saved in a temp variable, because if an address byte is received, the status differs from a data byte, so the two states can be distinguished (Lines 172-176). If the received byte is an address, the data is discarded (Lines 177-180), otherwise it is saved in the buffer w (Lines 184-186). After the data byte is received, lines 198 to 202 check whether an answer is expected or not. This is actually unnecessary for the communication protocol between NXT and CMUCam3, but is implemented here for completeness reasons. If an answer byte is expected, the code first checks for a buffer overflow, then takes control of the clock. Afterwards it delays communication through a loop for about 200ms, in order to write the register rx\_data without problems on the SSPBUF, which is used for sending through the I<sup>2</sup>C protocol. If the send command was successful, the program releases the clock and returns; otherwise it tries to resend the data.



#### 4.2.5. PIC – RS-232

Due to the design of the PIC16F690, initialization is similar to the I<sup>2</sup>C protocol. Some lines, particularly the port initialization, are a repetition from chapter 4.2.4., but because they are important to both protocols, they are stated here as well.

```

27: ;Switch to Register Bank 1
28:          bsf          status,rp0

34: ;Set Port B0-B3 and B7 to output and B4-B6 to input
35: ;B4 and B6 used for I2C; B5 is RS-232 RX, B7 is TX
36:  movlw    b'01110000'
37:  movwf    TRISB

38: ;Switch to Register Bank 2
39:  bcf     status,rp0
40:  bsf     status,rp1
41: ;Set all ports to digital
42:  movlw   0x00
43:  movwf   ANSEL
44:  movlw   0x00
45:  movwf   ANSELH

56: ;Switch to Register Bank 1
57:          bsf     status,rp0
58:  bcf     status,rp1

67: ;Uart settings BAUD RATE: 19200
68:  movlw   0
69:          movwf   SPBRGH
70:          movlw   d'12'
71:          movwf   SPBRG
72: ;Uart mode 8bit, no parity
73:  movlw   b'00100110'
74:          movwf   TXSTA
75:  movlw   b'00000111'
76:          movwf   ADCON1

83: ;Switch to Register Bank 0
84:  bcf     STATUS,RP0
85: ;Uart mode 8bit, no parity on reception
86:          movlw   b'10010000'
87:          movwf   RCSTA

```

Listing 6: UART initialization

Whilst programming, the ANSEL settings in line 42-45, List.6 posed some problems, because if improperly set, no digital input can be received and the port acts virtually deaf. These settings in interaction with the port settings (lines 36-37) determine the basic operation mode of the port, specific applications like UART communication are activated later. Baud rate and Transmission settings (lines 67-76) are set according to the calculation explained in detail in the USART-sheet [18] by Microchip.

The following code examples for RS-232 transmission are general functions for sending and receiving through an UART communication port.

```

264: Waittx
265:     btfss PIR1,TXIF
266:     goto Waittx
267:     movf tx_data,w
268:     movwf TXREG

```

Listing 7: RS-232 sending

The data to be sent in the send function, shown in List.7, is stored in the tx\_data and send by writing into the TXREG register. Lines 265-266 check for the port to be ready.

```

272: ser_in
273:
274: uart_ready
275:     btfss pir1,rcif
276:     goto ser_in
277:
278: ;catch overflow error
279:     btfsc rcsta,oerr
280:     goto overerror
281: ;catch framing error
282:     btfsc rcsta,ferr
283:     goto frameerror
284: ;Write data to receive register
285:     movf rcreg,w
286:     movwf rx_data
287:     goto received
288:
289: overerror
290:     bcf rcsta,cren ;pulse cren off...
291:     movf rcreg,w ;flush fifo
292:     movf rcreg,w ; all three elements.
293:     movf rcreg,w
294:     bsf rcsta,cren ;turn cren back on.
295: ;this pulsing of cren
296: ;will clear the oerr flag.
297:     goto ser_in
298:
299: frameerror
300:     movf rcreg,w ;reading rcreg clears ferr
301: goto ser_in

```

Listing 8: RS-232 receiving

In the receive loop (List.8), two errors are caught. Overrun errors are caused by too much input and framing errors, which appear when the stop bit is not seen at the expected time, due to wrong settings or bad transmissions.

#### 4.2.6. CMUCam3 – RS-232

Because the idea of the communication protocol was to only accept one byte requests and answer in one byte packages, the communication interface is kept very simple. The camera waits for a character to be received and answers accordingly. The UART communication settings, the general protocol for the RS-232 port, of course need to be configured in the same way as the connection board:

- Baud rate: 19'200 bits/s
- Parity bit: None
- Data bits: 8
- Stop bits: 1
- Flow Control: Off

Listings 9 to 11 show the code for communication with the connection board. The full code is attached in appendix A.1.

```
13: // initialize variables for communication
14: uint32_t val;
15: char input;

23: // configure uarts
24: cc3_uart_init (0, CC3_UART_RATE_19200,
25: CC3_UART_MODE_8N1, CC3_UART_BINMODE_TEXT);

27: // Make stdout and stdin not buffered
28: val = setvbuf (stdout, NULL, _IONBF, 0);
29: val = setvbuf (stdin, NULL, _IONBF, 0);
```

Listing 9: Initialization of the serial port

```
62: //wait for receive
63: input = getchar();
```

Listing 10: Sensing on received byte

```
56: uint16_t red_x, red_y;

67: if (input=='a')
68: {

95:     printf("%c",my_x);

101: }
```

Listing 11: Sending answer byte according to input

The input is coded as a character rather than a binary variable for simpler handling in the code. Tables for ASCII-codes [12] translate 8-bit commands easily into characters.

#### 4.2.7. CMUCam3 – Image processing

As mentioned in the evaluation, the CMUCam3 framework is a C-based programming environment, which is available as a pre-configured version for cygwin [9]. The camera can be adapted for specific use through individualization of the firmware. The concept of the CMUCam3 is, different than its predecessors, to create a complete open-source firmware, than can be adapted for different applications. The CMUCam3 framework uses the Code sourcery G++ compiler [10] and has many firmware samples already included into the distribution [11]. A different way of usage is the installation of a firmware emulation of the CMUCam2 on the CMUCam3. The previous versions of the camera had pre-programmed functionalities, which could be accessed through the standardized interface that is shown in Fig. 10. This is a valuable tool for debugging all problems related to brightness settings and color fine-tuning.

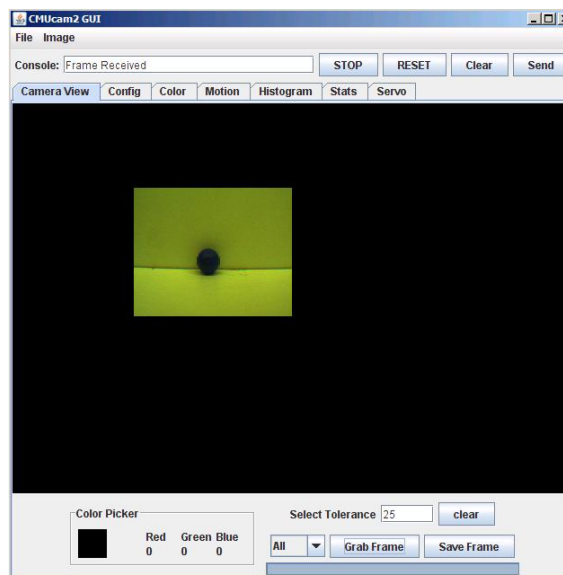


Figure 10: Screenshot CMUCam2 tool

### 4.3. Implementation

#### 4.3.1. Connection board

An adequate microcontroller is the PIC16F690 [14], because it supports both RS-232 and I<sup>2</sup>C and is sold in a package that includes an interface to a PC. In order to be able to use the shifted RS-232 port on the camera as well as a standard serial port on a PC, the RS-232 port of the PIC needs to be wired through a level shifter, like a MAX232 [13], to adjust the voltage levels from unshifted (low: 0V; high +5V) out of the PIC to shifted (low: -12V to -3V; high: +3V to + 15V). The camera would have a unshifted RS-232 port, but for debugging purposes it is more suitable to use the port which also connects to the PC.

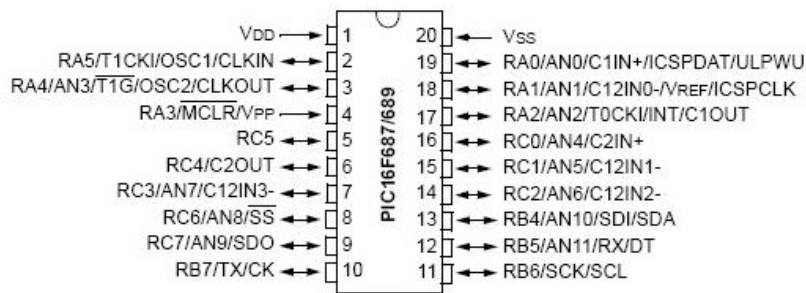


Figure 11: PIC16F690 Diagram, I<sup>2</sup>C on ports RB4 & RB6, RS-232 on ports RB5 & RB7

The connection board is powered through the CMUCam3, which has a voltage-controlled output port of 5V (see Figure 12), which is within the voltage range the PIC can handle.

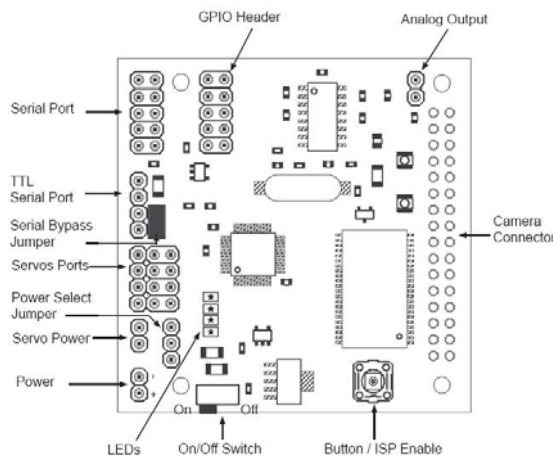


Figure 12 [7]: CMUCam3 - layout

According to the datasheets and the design concept requirements, the connection board has been setup in the manner shown in the circuit diagram in Fig. 13. On the prototype board, several LED's have been connected to the ports (used and unused ones) in order to be more flexible in debugging.

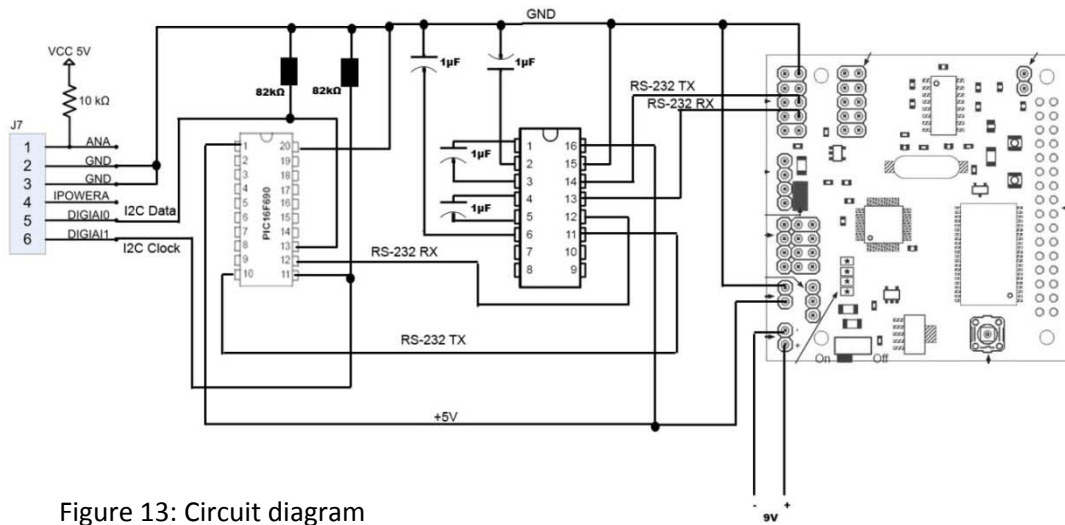


Figure 13: Circuit diagram

#### 4.3.2. Communication protocol

In accordance with the concept, the communication protocol is kept as simple as possible. The PIC expects a I<sup>2</sup>C packet of two bytes, first is always the constant address of the connection board (0x40), secondly the data is transmitted. The connection board relays the data to the camera and expects a single byte answer. This answer is responded to the NXT's request. The round trip time for a request and answer cycle is about 80ms on the NXT side.

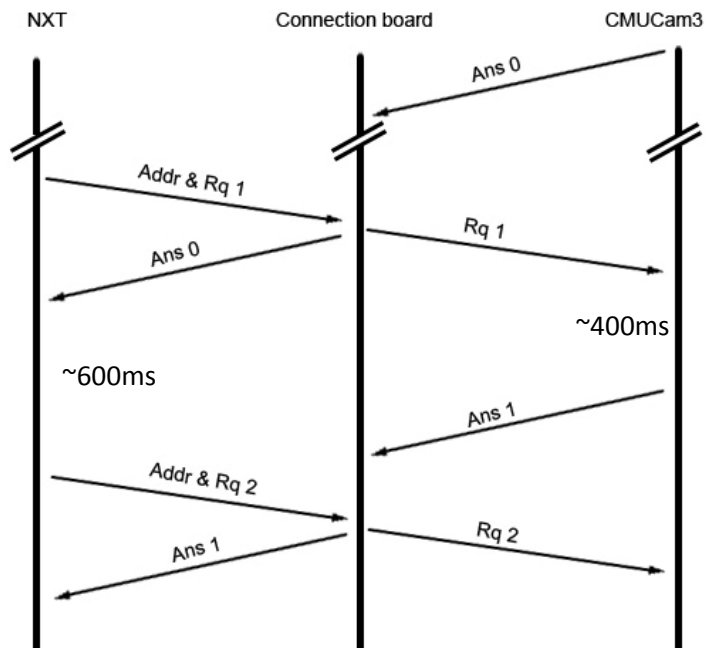


Figure 14: Communication protocol

The connection board cannot wait for the camera to answer a request of the NXT, because that would cause a Timeout in the protocol. This means that for every request, the answer of the request before is received, as is shown in Fig. 14. The delay on the camera side (~400ms) is due to the image processing, which is a rather extensive calculation. On the NXT side, this delay needs to be taken into account through the use of a waiting loop (~600ms) to make sure the CMUCam3 has enough time to answer the previous request. Delaying the answer in the I<sup>2</sup>C protocol leads to the abortion of the communication by the PIC, therefore this issue needs to be handled in the NXT code for communication, similar to the examples in the evaluations (5.3. & 5.4.). On initialization of the system, the camera sends a dummy answer (Ans 0) to the communication board in order to fill the buffer and avoid an abortion of the communication on the first request.

#### 4.4. Debugging

In order to be able to fully understand all operations, the PIC Programming has been done completely in Assembler. Microchip MPLAB IDE 8.10 [15] acted as programming environment, with MPLINK set up to compile to files. The connection board is connected to the PC via the PICKIT interface and the firmware of the PIC is flashed through PICKIT 2 [16]. All assembler code shown in this semester thesis is invariant to upper and lower case, so if the code is reused for different projects, this needs to be taken into account in the compiler.

For debugging purposes, the connection board can also be powered through the PICKIT in order to be more flexible with power supply. Further an oscilloscope has been connected to a unused output port of the PIC to monitor several operation and data transfer bits whilst debugging by redirecting important status registers to the output. Figure 15 shows an example of an RS-232 transmission on an oscilloscope.

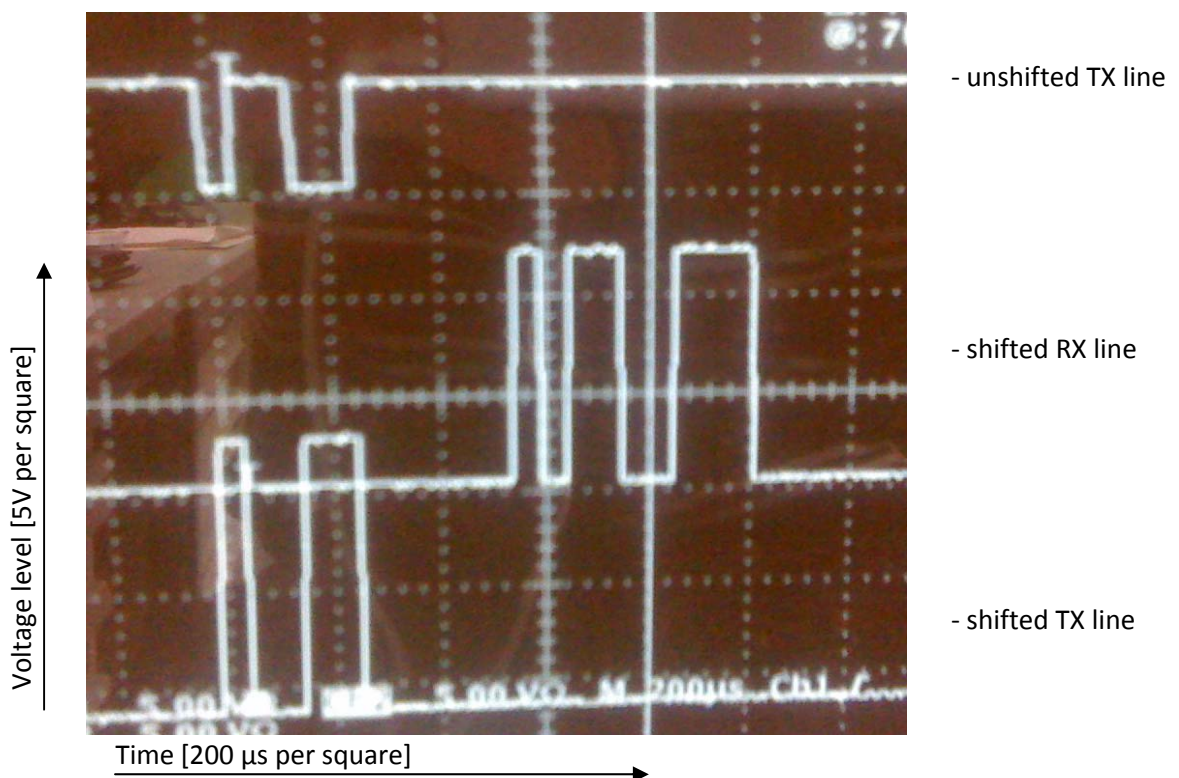


Figure 15: Sample RS-232 communication on oscilloscope





## 5. Evaluation

In the goals of this semester thesis, the evaluation target has been defined as enabling a Lego Mindstorm robot to recognize objects. In order to approach that target, we defined a white surface of  $\sim 3\text{m} \times 3\text{m}$  surrounded by walls of  $\sim 0.4\text{m}$  as evaluation environment. The robots starting point is somewhere in the middle, pointed at a random direction, as is shown for example in Fig. 16.

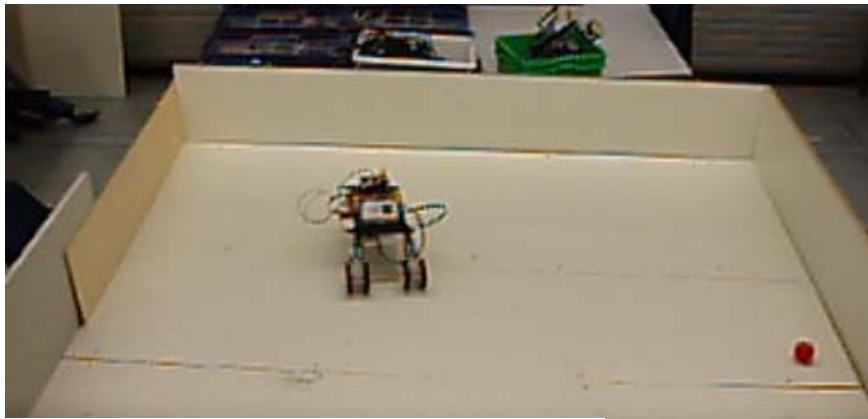


Figure 16: robot in test environment

The robot used for the following tests, is a simple robot with two separate motors at the back and a single wheel up front, which can rotate, in order to enable the robot turn around its own axis on spot. Figure 17 is an image of the robot with the camera to the left, the connection board in the middle and the NXT to the right.

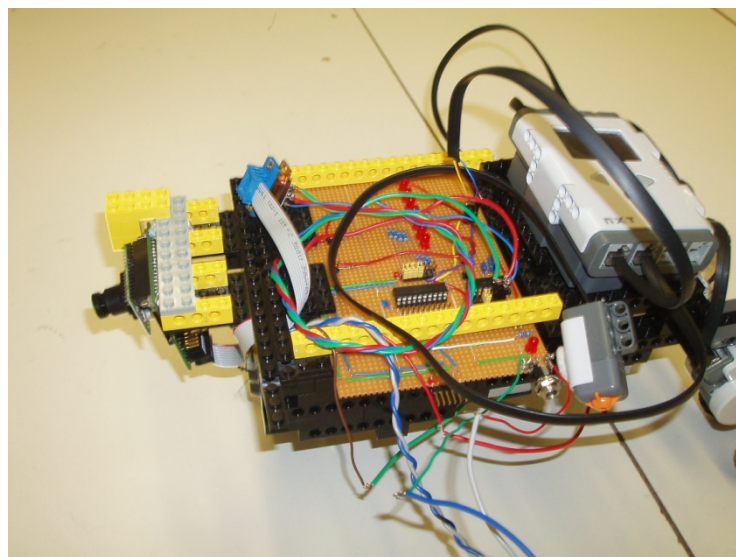


Figure 17: Picture of robot used for evaluation

## 5.1. Object recognition

### 5.1.1. Implementation

Basic image processing functions are implemented in the standard graphic libraries of the cc3 framework.

```

31: // Initialize Camera
32: cc3_camera_init ();
33: cc3_camera_set_colorspace (CC3_COLORSPACE_RGB);
34: cc3_camera_set_resolution (CC3_CAMERA_RESOLUTION_LOW);
35: cc3_camera_set_auto_white_balance (true);
36: cc3_camera_set_auto_exposure (true);

43: // setup an image structure
44: cc3_pixbuf_load ();
45: img.channels = 3;
46: img.width = cc3_g_pixbuf_frame.width;
47: // image will hold just 1 row for scanline processing
48: img.height = 1;
49: img.pix = cc3_malloc_rows (1);

```

Listing 12: Initialization of image processing

```

104: // This tells the camera to grab a new frame into the
105: // fifo and reset any internal location information.
106: cc3_pixbuf_frame_set_coi(CC3_CHANNEL_GREEN);
107: cc3_pixbuf_load ();

109: min_green = 0;
110: my_x = 0;
111: my_y = 0;
112: y = 0;
113: while (cc3_pixbuf_read_rows (img.pix, 1)) {
114: // read a row into the image memory from the camera
115:   for (uint16_t x = 0; x < img.width; x++) {
116:     uint8_t green = ((uint8_t *) img.pix)[x];
117:     if (green > min_green) {
118:       min_green = green;
119:       my_x = x;
120:       my_y = y;
121:     }
122:   }
123:   y++;
124: }
125: // printf ("Found min green value %d at %d, %d\n",
min_green, my_x, my_y);

```

Listing 13: Find coordinates of small red ball

Lines 104 to 112 in List.13 set up the frame buffer and initialize the channel green. From line 113 to 124, the darkest green spot in the image is searched and saved into the three variables:

- Min\_green: Brightness
- my\_x: X-coordinate of brightest spot
- my\_y: Y-coordinate of brightest spot

With brightness, it is only referred to brightness in the green-channel, therefore meaning the intensity of green at the individual pixel in a range of 0 to 255. The X- and Y-coordinates are absolute pixel values with x ranging from 1 to 176 and y ranging from 1 to 144.

### 5.1.2. Problems and Conclusion

With this color tracking library, it is feasible to track a red ball across a white background. One big problem that aroused is the relatively high red part of any lighting setting. Therefore if a precise tracking of any object should be done, the threshold for distinguishing between background noise and an actual object needs to be carefully adjusted to the specific setting. The best way to evaluate the thresholds is through the CMUCam2 tool by looking at the images produced by the different color channels. (see figure 17)

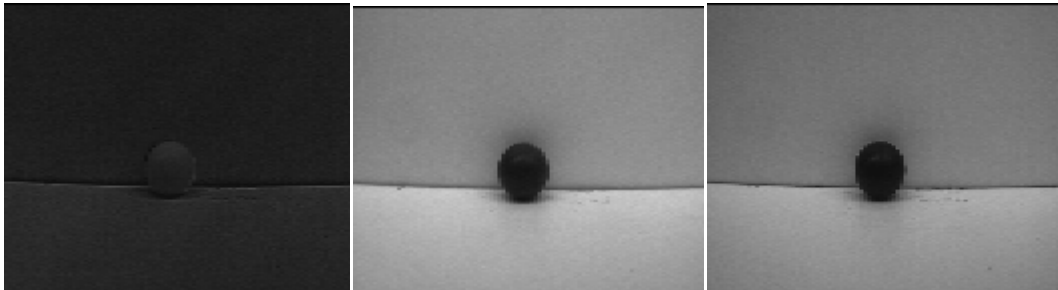


Figure 18: Picture of ball through color channels blue/green/red

## 5.2. Evaluation – Move to red ball

The evaluation is done in two steps: First the robot is set in the test environment, explained in Chapter 5, and programmed to center the camera on the red ball and move towards it. The code is shown in Appendix A.3., the NXT sends requests for the x-coordinate of the red ball and acts according to the answers it receives.

If the robot cannot see the ball, it is programmed to turn 20 degrees clockwise. As soon as the ball is visible, it switches to smaller rotations according to the relative coordinates received from the camera. As soon as the ball is in a centered position, the robot moves forward and reiterates the process.

The robot is able to find the ball without problems, as long as the camera is adjusted to a downward angle, so that only objects inside the evaluation environment are in sight. Some dark spots on the white surface were minor concerns at first, but adjusted to tighter brightness thresholds, the evaluation worked fine.

Message transmission round trip time is 1200ms, the according action afterwards has a duration of 80 to 300ms, so one movement takes the robot about 1500ms. With the designed rotation of 20 degrees clockwise if the ball is out of sight, execution time of the evaluation depends gravely on the initial setup. As soon as the ball is in sight, it is immediately found and centered. For a setup with a difference of 120 degrees clockwise, the robot is able to center the ball within 20s, which adds up to 13 movements.

## 5.3. Evaluation – Distinguish red and blue ball

The second evaluation was designed to localize two different objects and act according to their position. The robot is programmed to find one of the balls and center it, this time without moving towards it. On pressing the button attached to the NXT, it should look for the other ball. Every time the button is pressed, the NXT should switch its target.

With the code displayed in Appendix A.4., the robot is able to distinguish the two balls and easily center them. The only problem that came up was the background noise of the lighting settings in the test environment. Optimally, the red ball would have been traced through the blue channel and the blue ball vice versa, but because the light in the test setting had a very low blue spectrum, the blue channel could not be used very well for color tracking. The problem was solved by adjusting the green channel to the red ball with very narrow margins.

Execution times for the second evaluation were the same as for the one with just one ball. Message round trip time as well as movement pattern were identical, so only the initial setup of the evaluation environment has an impact on the performance.

## 6. Conclusion

All major goals of the original assignment have been reached. The concept has clearly proven to be working and a prototype has been built and tested. The communication board is fully functional and provides the necessary functions specified by the design concept. A library for object localization exists and can be used by further project without major interference with the existing Lego Mindstorms set. With the examples of the evaluations performed, the adaption to a future project is a simple task, smoothly fitting into the present system used by the PPS. The evaluation of a self-localization feature is solely a task of adjusting an existing library on the camera to the communication protocol.

### 6.1. Future Work

There are several directions, future work could focus on. One interesting topic is certainly the integration of the LeanXcam into the Lego Mindstorms set. This semester thesis has solved a lot of aspects, so the main focus could be on programming a driver or adapting a solution provided by the manufacturer in a final release.

Another important topic is the collection of libraries. As the CMUCam3 starts to be used in different types of projects, it would be of great benefit to collect the different firmwares used for different tasks and make them available to the public at a consolidated place. One approach would be to keep the homepage [20] accompanying this semester thesis up to date with any enhancements or additions to the code supplied by this thesis.

A last thing that should not be forgotten is the newly gained availability of a fully functional RS-232 port on the NXT. Especially older sensors only have RS-232 as a communication interface, so there are possibilities of easily integrating different types of sensors to the existing Lego Mindstorms set using the same design of the connection board.



## 7. Appendix

### A. Code

#### A.1. CMUCam3 firmware

```

#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <ctype.h>
#include <cc3.h>
#include <cc3_ilp.h>

int main (void)
{
    // initialize variables for communication
    uint32_t val;
    char input;

    cc3_image_t img;

    // init filesystem driver
    cc3_filesystem_init ();

    // configure uarts
    cc3_uart_init (0, CC3_UART_RATE_19200, CC3_UART_MODE_8N1,
                  CC3_UART_BINMODE_TEXT);

    // Make it so that stdout and stdin are not buffered
    val = setvbuf (stdout, NULL, _IONBF, 0);
    val = setvbuf (stdin, NULL, _IONBF, 0);

    //initialize camera
    cc3_camera_init ();
    cc3_camera_set_colorspace (CC3_COLORSPACE_RGB);
    cc3_camera_set_resolution (CC3_CAMERA_RESOLUTION_LOW);
    cc3_camera_set_auto_white_balance (true);
    cc3_camera_set_auto_exposure (true);

    cc3_led_set_state (0, false);
    cc3_led_set_state (1, false);
    cc3_led_set_state (2, false);

    // setup an image structure
    cc3_pixbuf_load ();
    img.channels = 3;
    img.width = cc3_g_pixbuf_frame.width;
    // image will hold just 1 row for scanline processing
    img.height = 1;
    img.pix = cc3_malloc_rows (1);

```

```

// printf ("Now we will use image data...\n");
val = 0;
int y;
    uint16_t my_x, my_y;
    uint8_t min_green, min_red, min_blue;
uint16_t red_x, red_y;
uint16_t blue_x, blue_y;

while (1) {
    cc3_led_set_state (0, true);

//wait for receive
    input = getchar();

    cc3_led_set_state (1, true);

if(input=='a')
    {
        // This tells the camera to grab a new frame into the fifo
and reset
        // any internal location information.
        cc3_pixbuf_frame_set_coi(CC3_CHANNEL_RED);
        cc3_pixbuf_load ();

        min_red = 255;
        red_x = 0;
        red_y = 0;
        y = 0;

        while (cc3_pixbuf_read_rows (img.pix, 1)) {
            // read a row into the image picture memory from the
camera
            for (uint16_t x = 0; x < img.width; x++) {
                uint8_t red = ((uint8_t *) img.pix)[x];
                if (red < min_red) {
                    min_red = red;
                    red_x = x;
                    red_y = y;
                }
            }
            y++;
        }
        // printf ("Found min green value %d at %d, %d\n",
min_green, my_x, my_y);

        if(min_red<27)
            {
                printf("%c",red_x);
            }
            else
            {
                printf("%c",250);
            }
        }
else if(input=='b')
    {

```



```

    // This tells the camera to grab a new frame into the fifo
and reset
    // any internal location information.
    cc3_pixbuf_frame_set_coi(CC3_CHANNEL_GREEN);
    cc3_pixbuf_load ();

    min_green = 255;
    my_x = 0;
    my_y = 0;
    y = 0;
    while (cc3_pixbuf_read_rows (img.pix, 1)) {
        // read a row into the image picture memory from the
camera
        for (uint16_t x = 0; x < img.width; x++) {
            uint8_t green = ((uint8_t *) img.pix)[x];
            if (green < min_green) {
                min_green = green;
                my_x = x;
                my_y = y;
            }
        }
        y++;
    }
    // printf ("Found min green value %d at %d, %d\n",
min_green, my_x, my_y);
    if(min_green<17)
    {
        printf("%c",my_x);
    }
    else
    {
        printf("%c",250);
    }
}
else if(input=='c')
{
    // This tells the camera to grab a new frame into the fifo
and reset
    // any internal location information.
    cc3_pixbuf_frame_set_coi(CC3_CHANNEL_BLUE);
    cc3_pixbuf_load ();

    min_blue = 255;
    blue_x = 0;
    blue_y = 0;
    y = 0;
    while (cc3_pixbuf_read_rows (img.pix, 1)) {
        // read a row into the image picture memory from the
camera
        for (uint16_t x = 0; x < img.width; x++) {
            uint8_t blue = ((uint8_t *) img.pix)[x];
            if (blue < min_blue) {
                min_blue = blue;
                blue_x = x;
                blue_y = y;
            }
        }
    }
}

```

```

        }
        Y++;
    }
    // printf ("Found min green value %d at %d, %d\n",
min_green, my_x, my_y);
    if(min_blue<20)
    {
        printf("%c",blue_x);
    }
    else
    {
        printf("%c",250);
    }
}
else
{
printf("%c",249);
}
}
free (img.pix);           // don't forget to free!

return 0;
}

```

## A.2. Connection board firmware

```

; Leo den Hartog
; Semester Thesis
; Version 1.0
; 08-08-26

#include <p16F690.inc>
__config (_INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF &
_MCLRE_OFF & _CP_OFF & _BOR_OFF & _IESO_OFF & _FCMEN_OFF)
#define NODE_ADDR 0x40 ; I2C address, 64 decimal

    cblock 0x20
Delay0
Delay1
Delay2
Delay3
rx_data
tx_data
testoutput
temp
    endc

    bcf          status,rp0
    clrf sspcon
    clrf pir1
    clrf pir2

;Switch to Register Bank 1

```

```

    bsf          status,rp0
;Set Port A0-A5 to input and A6-A7 to output
    movlw b'00111111'
    movwf TRISA

    movwf WPUA          ;Weak-PullUp an
;Set Port B0-B3 and B7 to output and B4-B6 to input
;B4 and B6 used for I2C; B5 is RS-232 RX, B7 is TX
    movlw b'01110000'
    movwf TRISB
;Switch to Register Bank 2
    bcf status,rp0
    bsf status,rp1
;Set all ports to digital
    movlw 0x00
    movwf ANSEL
    movlw 0x00
    movwf ANSELH

;Osccon
    movlw B'01110001'
    movwf OSCCON
;Comparator off
    clrf CM1CON0
    clrf CM2CON0
;ECCP off
    clrf CCP1CON

;Switch to Register Bank 1
    bsf status,rp0
    bcf status,rp1
;Set Port C0-C7 to output
    movlw b'00000000'
    movwf TRISC

;ADC off
    bcf          ADCON0, 0

;Uart settings BAUD RATE: 19200
    movlw 0
        movwf SPBRGH
        movlw d'12'
        movwf SPBRG
;Uart mode 8bit, no parity on transmission
    movlw b'00100110'
        movwf TXSTA
    movlw b'00000111'
        movwf ADCON1

;Set i2c address
    movlw NODE_ADDR ; set Node Address
    movwf SSPADD

    clrf pie1
;Switch to Register Bank 0

```

```

        bcf          STATUS,RP0
;Uart mode 8bit, no parity on reception
        movlw      b'10010000'
        movwf      RCSTA
;Disable interrupts
        bcf          INTCON, GIE
        bcf          INTCON, PEIE
;initialize i2c
        movlw      b'00111001'          ; Setup SSP module for 7-bit
        movwf      SSPCON              ; address, slave mode
        movlw      b'00110110'          ; Setup SSP module for 7-bit
        movwf      SSPCON              ; address, slave mode

        movlw      h'20'
        movwf      Delay0
        movlw      h'20'
        movwf      Delay1
        movlw      h'20'
        movwf      Delay2
        movlw      h'20'
        movwf      Delay3
DelayLoopStart
        decfsz     Delay0,f
        goto      DelayLoopStart
        movlw      h'20'
        movwf      Delay0
        decfsz     Delay1,f
        goto      DelayLoopStart
        movlw      h'20'
        movwf      Delay1
        decfsz     Delay2,f
        goto      DelayLoopStart
        movlw      h'20'
        movwf      Delay2
        decfsz     Delay3,f
        goto      DelayLoopStart

        movlw      0
        movwf      TXREG

        movf      rcreg,w
        movf      rcreg,w
        movf      rcreg,w

        clrf      porta
        clrf      portb
        clrf      portc
        clrf      rx_data

        movlw      b'01100001'
        movwf      testoutput

        movlw      b'01101100'
        movwf      rx_data

```

```

;START PROGRAM

MainLoop
    goto Receivei2c
    goto MainLoop

Receivei2c

Waitrx12c
;Write Status on output for debugging purposes
    bsf     STATUS,RP0      ;Bank 1
    movf   sspstat,w
    bcf    STATUS,RP0      ;Bank 0
    movwf  PORTC

;Wait for received byte on I2C
    btfsc  PIR1,sspif
    goto  i2cnew
;Repeat Loop
    goto  Waitrx12c

i2cnew
    banksel  SSPSTAT
    movf   SSPSTAT,W      ; Get the value of SSPSTAT
    bcf    STATUS,RP0      ;Bank 0
    andlw  b'00101101' ; Mask out unimportant bits
    banksel  Temp
    movwf  Temp          ; Put masked value in Temp
    movwf  Temp          ; for comparision checking.

State1      ; Handle address byte
    movlw  b'00001001'
    xorwf  Temp,W
    btfss  STATUS,Z      ; Address received?
    goto  State2          ; No, handle data byte
    banksel  SSPBUF
    movf   SSPBUF,W      ; Get the addr and throw it away
    bcf    PIR1,sspif
;Repeat Loop
    goto  Waitrx12c

State2      ; Handle data byte
    banksel  SSPBUF
    movf   SSPBUF,W      ; Get the byte into W
    movwf  txreg
    bcf    PIR1,sspif

Waiting
    btfss  PIR1,sspif
    goto  Waiting

Waiting2
    bsf     STATUS,RP0      ;Bank 1
    btfsc  SSPSTAT, 2      ;Check if answer is exp.
    goto  WriteI2C
    bcf    STATUS,RP0      ;Bank 0
    goto  Waitrx12c
    
```

```

WriteI2C
    banksel SSPSTAT
    btfsc      SSPSTAT,BF          ; Is the buffer full?
    goto      WriteI2C           ; Yes, keep waiting.
    banksel SSPCON
                                ; No, continue.
DoI2CWrite
    bcf      SSPCON,WCOL          ; Clear the WCOL flag.

    movlw    h'0f'
    movwf    Delay0
    movlw    h'03'
    movwf    Delay1
DelayLoopSend2
    decfsz   Delay0,f
    goto     DelayLoopSend2
    movlw    h'20'
    movwf    Delay0
    decfsz   Delay1,f
    goto     DelayLoopSend2

    movf     rx_data,w
    movwf    SSPBUF                ; Write the byte in WREG
    btfsc    SSPCON,WCOL          ; Collision?
    goto     DoI2CWrite
    bsf      SSPCON,CKP           ; Release the clock.
    movlw    h'20'
    movwf    Delay0
    movlw    h'20'
    movwf    Delay1
    movlw    h'20'
    movwf    Delay2
    movlw    h'02'
    movwf    Delay3
DelayLoopSend
    decfsz   Delay0,f
    goto     DelayLoopSend
    movlw    h'20'
    movwf    Delay0
    decfsz   Delay1,f
    goto     DelayLoopSend
    movlw    h'20'
    movwf    Delay1
    decfsz   Delay2,f
    goto     DelayLoopSend
    movlw    h'20'
    movwf    Delay2
    decfsz   Delay3,f
    goto     DelayLoopSend
    goto     ReceiveLoop
received
    bcf      pir1,sspif
    movf     rcreg,w
    movf     rcreg,w
    movf     rcreg,w
    bsf      STATUS,RP0          ;Bank 1

```

```

        bcf          SSPSTAT, 2
        bcf          STATUS,RP0      ;Bank 0,
        goto MainLoop

SendLoop
Waittx
        btfss PIR1,TXIF
        goto Waittx
        movf        tx_data,w
        movwf       TXREG

ReceiveLoop
ser_in

uart_ready
        btfss pir1,rcif
        goto ser_in

;catch overflow error
        btfsc rcsta,oerr
        goto overerror
;catch framing error
        btfsc rcsta,ferr
        goto frameerror
;Write data to receive register
        movf rcreg,w
        movwf rx_data
        goto received

overerror
        bcf rcsta,cren ;pulse cren off...
        movf rcreg,w      ;flush fifo
        movf rcreg,w      ; all three elements.
        movf rcreg,w
        bsf rcsta,cren ;turn cren back on.
                                ;this pulsing of cren
                                ;will clear the oerr flag.

        goto ser_in

frameerror
        movf rcreg,w      ;reading rcreg clears ferr
        goto ser_in

end
    
```

### A.3. NXT Evaluation – Move to red ball

```

#include "NXCDefs.h"

#define I2C_PORT S1
#define MOVE_TIME 300
#define TURN_TIME 75
#define BIG_TURN_TIME 250

int err;
    
```

```

byte send[] = {0x40, 0xfe};
byte recv[] = {1};

sub turn_right()
{
    OnFwd(OUT_A, 60);
    OnRev(OUT_C, 55);
    Wait(TURN_TIME);
    Off(OUT_AC);
}

sub sendrecvi2c (byte data)
{
    ClearScreen ();
    send[0] = 0x40;
    send[1] = data;

    while (I2CCheckStatus(I2C_PORT) != 0);
    err = I2CWrite(I2C_PORT, 1, send);

    while (I2CCheckStatus(I2C_PORT) != 0);
    TextOut (0, LCD_LINE4, "Clear");

    while (I2CBytesReady(I2C_PORT) < 1);
    I2CRead(I2C_PORT, 1, recv);
    TextOut (0, LCD_LINE5, "Read");
    NumOut(60, LCD_LINE5, recv[0]);

    Wait(500);
}

sub turn_left()
{
    OnFwd(OUT_C, 60);
    OnRev(OUT_A, 55);
    Wait(TURN_TIME);
    Off(OUT_AC);
}

sub turn_big_right()
{
    OnFwd(OUT_A, 60);
    OnRev(OUT_C, 55);
    Wait(BIG_TURN_TIME);
    Off(OUT_AC);
}

sub turn_big_left()
{
    OnFwd(OUT_C, 60);
    OnRev(OUT_A, 55);
    Wait(BIG_TURN_TIME);
    Off(OUT_AC);
}

```



```

sub straight()
{
  OnFwd(OUT_AC, 60);
  Wait(MOVE_TIME);
  Off(OUT_AC);
}

task main()
{
  SetSensorType(I2C_PORT, SENSOR_TYPE_LOWSPEED);
  SetSensorMode(I2C_PORT, IN_MODE_RAW);
  ResetSensor(I2C_PORT);
  SetSensor(IN_2, SENSOR_TOUCH);

  while(true)
  {

    sendrecvi2c(0x62);

    Wait(100);
    sendrecvi2c(0x62);

    if(recv[0]>247)
    {
      TextOut (0, LCD_LINE2, "Error");
      turn_big_right();
    }
    else if(recv[0]>90)
    {
      TextOut (0, LCD_LINE2, "Turn Right");
      turn_right();
    }
    else if(recv[0]<75)
    {
      TextOut (0, LCD_LINE2, "Turn Left");
      turn_left();
    }
    else
    {
      TextOut (0, LCD_LINE2, "Ball found and centered");
      straight();
    }
    Wait(100);

    if(SENSOR_2)
    {
      break;
    }

  }
  ClearScreen ();
  TextOut (0, LCD_LINE2, "Done");
  Wait(10000);
}

```

#### A.4. NXT Evaluation – Distinguish red and blue ball

```

#include "NXCDefs.h"

#define I2C_PORT S1
#define MOVE_TIME 300
#define TURN_TIME 75
#define BIG_TURN_TIME 250

int err;
byte send[] = {0x40, 0xfe};
byte recv[] = {1};
int tosend = 97;

sub turn_right()
{
  OnFwd(OUT_A, 60);
  OnRev(OUT_C, 55);
  Wait(TURN_TIME);
  Off(OUT_AC);
}

sub turn_left()
{
  OnFwd(OUT_C, 60);
  OnRev(OUT_A, 55);
  Wait(TURN_TIME);
  Off(OUT_AC);
}

sub turn_big_right()
{
  OnFwd(OUT_A, 60);
  OnRev(OUT_C, 55);
  Wait(BIG_TURN_TIME);
  Off(OUT_AC);
}

sub turn_big_left()
{
  OnFwd(OUT_C, 60);
  OnRev(OUT_A, 55);
  Wait(BIG_TURN_TIME);
  Off(OUT_AC);
}

sub straight()
{
  OnFwd(OUT_AC, 60);
  Wait(MOVE_TIME);
  Off(OUT_AC);
}

sub sendrecvI2c (byte data)

```

```

{

    send[0] = 0x40;
    send[1] = data;

    while (I2CCheckStatus(I2C_PORT) != 0);
    err = I2CWrite(I2C_PORT, 1, send);

    while (I2CCheckStatus(I2C_PORT) != 0);
    TextOut (0, LCD_LINE4, "Clear");

    while (I2CBytesReady(I2C_PORT) < 1);
    I2CRead(I2C_PORT, 1, recv);

        Wait(500);
}

task main()
{
SetSensorType(I2C_PORT, SENSOR_TYPE_LOWSPEED);
SetSensorMode(I2C_PORT, IN_MODE_RAW);
ResetSensor(I2C_PORT);
SetSensor(IN_2, SENSOR_TOUCH);

while(true)
{

if(SENSOR_2)
{
    ClearScreen ();

if(tosend==97)
{
tosend=98;
TextOut (0, LCD_LINE1, "Search Red ball");
}

else
{
TextOut (0, LCD_LINE1, "Search Blue ball");
tosend=97;
}

}

sendrecvi2c(tosend);

Wait(100);
ClearScreen ();
    sendrecvi2c(tosend);
TextOut (0, LCD_LINE5, "Read");
NumOut(60, LCD_LINE5, recv[0]);

if(recv[0]>247)

```

```
{
TextOut (0, LCD_LINE2,"Error");
turn_big_right();
}
else if(recv[0]>90)
{
TextOut (0, LCD_LINE2,"Turn Right");
turn_right();
}
else if(recv[0]<75)
{
TextOut (0, LCD_LINE2,"Turn Left");
turn_left();
}
else
{
if(tosend==97)
{
TextOut (0, LCD_LINE2,"Blue ball found and centered");
}

else
{
TextOut (0, LCD_LINE2,"Red ball found and centered");
}

}
Wait(100);

}
ClearScreen ();
TextOut (0, LCD_LINE2,"Done");
Wait(10000);
}
```



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Institut für  
Technische Informatik und  
Kommunikationsnetze

Semester Thesis at the  
Department of Information Technology and  
Electrical Engineering

for

**Leo Den Hartog**

**Lego Mindstorms NXT Camera**

**Advisors:** Wolfgang Haid  
Matthias Woehrle

**Professor:** Prof. Dr. Lothar Thiele

**Handout Date:** 09. 06. 2008

**Due Date:** 11. 08. 2008

# 1 Introduction

Lego Mindstorms NXT is a versatile robotics platform that is widely used for teaching purposes at schools and universities. The Computer Engineering and Networks Lab at ETH offers a course on Mindstorms NXT for first and second semester students, too [7]. The core of the Mindstorms NXT platform is a programmable unit, referred to as the *NXT*, see Fig. 1.

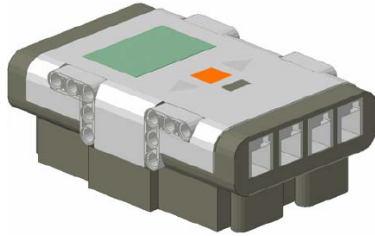


Figure 1: NXT: The programmable unit of the Lego Mindstorms NXT platform.

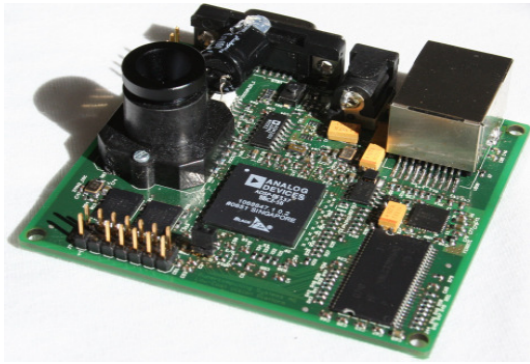
Unfortunately, the visual capabilities of the standard Mindstorms NXT platform are very limited because only a simple light-dependent transistor is available as a sensor for visual input. This severely narrows the scope of robotic applications because other available sensors, like the ultrasound distance sensor and touch sensors, cannot compensate for the missing visual capabilities. To overcome this problem, we aim to equip the NXT with a video camera, thereby opening up a huge range of new applications.

## 2 Project Goals

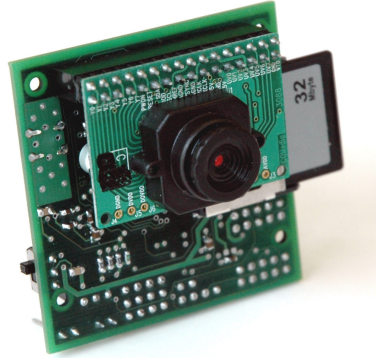
The goal of this project is to couple a video camera with the Mindstorms NXT, enabling NXT-based robots to orient themselves in their environment, extract information from the environment, and react to it. In this project, two existing camera platforms will be evaluated, namely the low-cost vision (LCV) platform developed by Supercomputing Systems AG Zurich, see Fig. 2(a), and the CMUCam3 developed at the Robotics Institute of Carnegie Mellon University, see Fig. 2(b). Both platforms are fully programmable embedded computer vision systems. This allows to analyze the video stream within the vision system and communicate only the results of the analysis to the connected system, that is, the NXT.

More concretely, the project will be split up into three main tasks:

- Design and implementation of the hardware interface between the camera platforms and the NXT
- Design and implementation of the software libraries for obtaining data from the camera (both on the camera platforms and the NXT)
- Implementation of software libraries for typical tasks to be performed by the camera when used in conjunction with the NXT



(a) LCV Camera Platform.



(b) CMUCam3 Camera Platform.

Figure 2: Camera Platforms.

- Evaluation of the system

The biggest challenge in the project will be to develop an elaborate concept and implement it using existing resources and not to come up with a large circuit or write lots of new code. The criteria to consider are online and offline reconfigurability, reusability, and debugging facilities:

- *Configurability:* Cameras can be used in robots for numerous purposes: object detection, object tracking, feedback for robot control, system calibration, etc. An example is, for instance, a robot which approaches an object (object detection and tracking) and uses then a vision-controlled arm to lift it (feedback for robot control). Configurability refers to the possibility to use a camera for these different purposes without reprogramming it.
- *Usability:* When the application scenario changes, the robot needs to be reprogrammed. It is desirable that new vision algorithms can be easily implemented or existing ones easily modified. There are many hardware and software components available for the Mindstorms NXT platform. Solutions based on widely-used “standard” components are preferable. It should be noted that target programmers are first and second semester students with limited programming skills.
- *Debugging Facilities:* Testing and debugging of an entire embedded system is much more involved compared to testing and debugging of software. The setup is usually time-consuming, the used tools are not as familiar as the ones used in software development, and the employed testing and debugging techniques and approaches are different. Testing and debugging of algorithms should be possible on a standard PC. A vital part of vision algorithms is the processing of an actual image. The work should consider facilitating the design and implementation by providing a method to easily simulate the processing of a given image and output the result for inspection. An effortless method to acquire images from the camera should be implemented.

## 3 Tasks

The project will be split up into several subtasks, as described below.

### 3.1 Familiarization with the Environment

In the beginning of the project the focus is on getting acquainted with the available hardware and software environment. In terms of hardware, the three systems used in this project (NXT, LCV, CMUCam3) need to be analyzed with respect to their basic system design and the available input and output interfaces. In terms of software, the three systems need to be analyzed with respect to available programming languages, the firmware and compiler support for these languages, the available APIs for the NXT (especially concerning the support of connected hardware devices), and development environments.

In addition to the familiarization with the hardware used in this project, a literature search about related projects should be carried out.

### 3.2 Definition of Requirements and Hardware/Software Design Concept

Based on the insights gained in the first project phase, a list of requirements is defined. Tailored towards these requirements, a hardware/software design concept is compiled specifying in detail the implementation of the system. The following aspects need to be taken into consideration:

#### Hardware

- selection of appropriate interfaces for communication between LCV/ CMUCam3 and NXT
- design of connection board for coupling LCV/CMUCam3 and NXT
- power supply for camera and connection board on mobile NXT-based robots

#### Software

- selection of programming language (candidates are NXC [4], Lejos [1], pbLua [5], Matlab [2] and possibly others)
- selection of external graphics libraries, if necessary
- specification of software libraries for communication between LCV/ CMUCam3 and NXT

### 3.3 Implementation

During the implementation phase, the system is implemented according to the design concept. In particular, a prototype version of the connection board is assembled and the necessary software libraries are written.



### 3.4 Evaluation

To evaluate the system, the following two capabilities should be demonstrated:

1. *Self-Localization*: A robot will be placed into a (predefined) area which is clearly marked (using black electrical tape on a white surface, for instance). By analyzing the visual input, the robot derives its position within this area.
2. *Object-Localization*: Different objects of known size, color and shape are placed into the visual field of a robot. By analyzing the visual input, the robot derives the position of the objects relative to its own position.

Based on these two capabilities, finally a complete robot is implemented. The task is to develop a robot that could take part in one of the disciplines typically found in robot competitions, such as “Robot Sumo” or “Puck Collect” [3, 6].

### 3.5 Documentation

Documentation is a vital part of this project, especially due to two reasons: First, the camera will be used in classroom. Students should be able to understand the system by just reading the documentation. Second, there is a large interest in Mindstorms NXT related topics on the one hand, but rather little good documentation on the other hand. In order to make a lasting contribution to the community, besides offering a good technical solution a good documentation needs to be provided.

The project documentation is expected to consist of three parts:

- *Report*: The report describes the background of the work, the technical details of the implementation, and the results. This report addresses a technically skilled audience that is interested in replicating or extending the work.
- *Tutorial*: The tutorial explains how the camera set-up for the NXT looks like (when using a camera with pre-installed software) and how the camera can be used by the NXT. In addition, the tutorial explains what is necessary to modify or write new programs. The tutorial is addressed at the first and second semester students attending the Mindstorms NXT class offered at our lab and thus should include an introductory example application.
- *Web Page*: The web page presents the main results of the project in a concise form. The web page will be integrated in our lab’s Mindstorms NXT page [7] (HTML only).

## 4 Project Organization

The guidelines for semester and master projects carried out at the Computer Engineering and Networks Lab are available at:

[https://www.tik.ee.ethz.ch/intranet/download/students/student\\_thesis\\_guidelines\\_ge.pdf](https://www.tik.ee.ethz.ch/intranet/download/students/student_thesis_guidelines_ge.pdf).

Supercomputing Systems AG provides the documentation of the LCV platform under a non-disclosure agreement. Publishing the documentation or technical details about this platform is not permitted. The own work carried out using the platform is not affected by the non-disclosure agreement.

There will be a weekly meeting to discuss the project's progress. A revision of the working document should be provided the day before.

Two hardcopies of the report are to be turned in. All copies remain the property of the Computer Engineering and Networks Laboratory.

A copy of the developed software needs to be handed in on CD or DVD at the end of the project.

## References

- [1] P. Andrews, J. Stuber, L. Griffiths, B. Bagnall, M. P. Scholz, T. Rinkens, J. A. B. na Moral, and J. Solorzano. Lejos — Java for Lego Mindstorms. <http://lejos.sourceforge.net/>.
- [2] L. Atorf, A. Behrens, A. Knepper, R. Schwann, B. Neumann, R. Schnitzler, J. Balle, T. Herold, and A. Telle. RWTH-Mindstorms NXT Toolbox for MATLAB. <http://www.mindstorms.rwth-aachen.de/>.
- [3] D. Calkins and S. Davalos. RoboGames. <http://www.robogames.net/>.
- [4] J. Hansen. Next Byte Codes & Not eXactly C. <http://bricxcc.sourceforge.net/nxc/>.
- [5] R. Hempel. pbLua Home Page. <http://www.hempeldesigngroup.com/lego/pbLua/>.
- [6] InnoC.at. RobotChallenge. <http://www.robotchallenge.at/>.
- [7] M. Woehrle and W. Haid. PPS Lego Mindstorms. <http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/>.

Zürich, June 2, 2008

## 8. References

- [1] TIK Lego Mindstorms, [www.tik.ee.ethz.ch/mindstorms/](http://www.tik.ee.ethz.ch/mindstorms/)
- [2] NXC, <http://bricxcc.sourceforge.net/nbc/>
- [3] Lego Mindstorms NXT, 2006, Claudia Frischknecht and Thomas Other
- [4] Bluetooth Anbindung für Lego Mindstorms, 2002, Andres Erni and Stefan Reichmuth
- [5] CMUCam3, <http://www.cmucam.org/>
- [6] LeanXcam Datasheet, Super computing systems
- [7] CMUCam3 Datasheet, 2006, Carnegie Mellon University
- [8] PIC-NXT Interface, <http://www.extremenxt.com/picnxt.html>
- [9] cygwin, <http://www.cygwin.com/>
- [10] Code sourcery G++, <http://www.codesourcery.com/>
- [11] cc3 distribution, <http://www.cmucam.org/wiki/Downloads/cc3.zip?format=raw>
- [12] ASCII-code, <http://en.wikipedia.org/wiki/ASCII>
- [13] MAX232 Data Sheet, 2002, Texas Instruments
- [14] PIC16F690 Data Sheet, 2006, Microchip
- [15] MPLAB IDE 8.10, Microchip
- [16] PICKIT 2 v2.11, Microchip
- [17] RS-232 Transmission, <http://en.wikipedia.org/wiki/RS-232>
- [18] USART, 2001, Microchip
- [19] PIC16F690 Errata, 2007, Microchip
- [20] "Lego Mindstorms NXT Camera" webpage, 2008,  
<http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/camera>
- [21] Vision Subsystem v2 for NXT, <http://www.mindsensors.com>
- [22] Surveyor SRV-1, <http://www.surveyor.com>
- [23] I<sup>2</sup>C Standard, <http://www.i2c-bus.org/>
- [24] RS-232 Standard, ANSI/EIA/TIA-232-F-1997