PROJEKTPRAKTIKUM COMPUTATIONAL NEURO ENGINEERING: MY KEEPON MOBILIZATION

eingereichtes Projektpraktikum von:

Imen Bouzouita

Anna Koch

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer/-in:	M.Sc. Christian Denk
Beginn:	01.05.2013
Zwischenbericht:	24.05.2013
Abgabe:	18.06.2013

Abstract

In this project, we designed and built a moving robot consisting of the My KeepOn, the event-based vision sensor and the OmniRob robot platform. We changed also the existing My KeepOn setup in order to create a communication user interface and to develop a demo application showing the interaction of the various parts of the entire system.

Zusammenfassung

In diesem Projekt haben wir einen mit einem eDVS Sensor ausgestatteten beweglichen Roboter entworfen und zusammengebaut. Dieser Roboter besteht aus dem My KeepOn und dem OmniRob Roboter-Plattform. Außerdem haben wir den bestehenden My KeepOn Setup geändert um eine Kommunikations-Benutzeroberfläche zu schaffen und um eine Demo Anwendung zu entwickeln. Diese Anwendung hat als Ziel, die Interaktion zwischen den verschiedenen Teilen zu evaluieren.

Contents

1 Introduction

1.1 Motivation and Objectives

My KeepOn is a commercial toy robot, which was built for educational reasons. The user can interact with it via buttons to select a desired mode of use or via push sensors to accomplish certain actions like making particular sounds. Furthermore, it has some degrees of freedom which allow it to perform articulated motion e.g. rotation and bending.
The robot was originally constructed operating in sealed stand-alone mode. For research reasons, it was provided with a microcontroller LPC1769 (see fig. 2.3) and its setup was already changed in order to make use of its sensors and actuators and enable higher level control of the motors.
Our goal in this project is to build a mobile robot, consisting of the My KeepOn on top of an OmniRob robot platform and equipped with an event-based vision sensor (see fig. 1.1). This entire system is controlled via a microcontroller LPC4337 (see fig. 2.5). A further objective was to evaluate the whole constructed system by a demo application demonstrating the interaction of the different parts, e.g. tracking and approaching a visual stimulus.
To achieve our goals, we started with modifying the existing My KeepOn setup and creating a communication user interface. After that, we created a possible design for interfacing the My KeepOn with the OmniRob and the eDVS. The next step was to assemble the hardware and Software of the entire system. And finally we started to develop the demo application.

Figure 1.1: Entire system with My KeepOn, OmniRob and eDVS

1.2 Outline

This report is organized into 4 chapters.
After the introduction, chapter 2 describes the hardware components of the entire system. In chapter 3 the implementation approaches for interfacing and interconnecting the system and developing the evaluation demo application are presented. Finally we deduce conclusions and propose some possible future work.

2 Hardware

In this chapter, we aim first to describe the existing hardware components of our system, i.e. the MyKeepOn robot, the Omnirob platform as well as the event-based vision sensor (eDVS). Furthermore, we will propose a possible design for interconnecting the entire system.

2.1 Description of the existing components

2.1.1 Vision Sensor eDVS

The eDVS includes a Silicon Retina chip (DVS), a Microcontroller (NXP LPC2106), an external Communication Interface (UART) and a Lens (BB193) (see fig.2.1). The eDVS with 128x128 Pixel transmits local relative intensity pixel changes. Therefore the eDVS is not working like a conventional sensor that sends entire images with pixel information in fixed frame rates, but it only quantizes brightness changes caused by movement, which leads to a much lower data rate. This provides us with high time resolution as well as an enormous reduction of data storage memory, computational requirements and size [1]. The data of the each pixel is converted by the Microcontroller and could be transferred to external components using a UART connection.

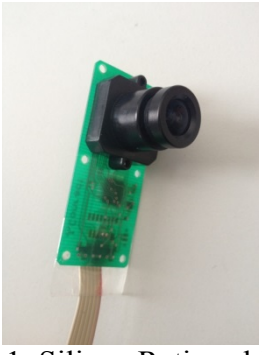

Figure 2.1: Silicon Retina chip eDVS

2.1.2 My KeepOn

My KeepOn is a toy robot which can perform several actions such as emitting sounds, rotating or even dancing to Music. It has different interaction sensors: two of them are

the front buttons responsible for selecting a mode (sleep, dance, touch) and the rest of the sensors are inside its body. The robot has 3 motors that provide four degrees of freedom in movement. One is used for the rotation around the Y-Axis, which allows the head to bend forwards and backwards. It can also bend around the X-Axis to the left and to the right. Additionally, the head can turn around the Z-Axis or it can also move up and down (this function is called pon later).
The circuit board inside the platform includes two FPGAs (Padauk P234CS20, Padauk P232CS14) which are connected via an I^2C Bus. The P232, which is the master on the I^2C bus, is controlling the motor action. The P234 is the slave and handles the sounds and the buttons' interaction (see fig.2.2).

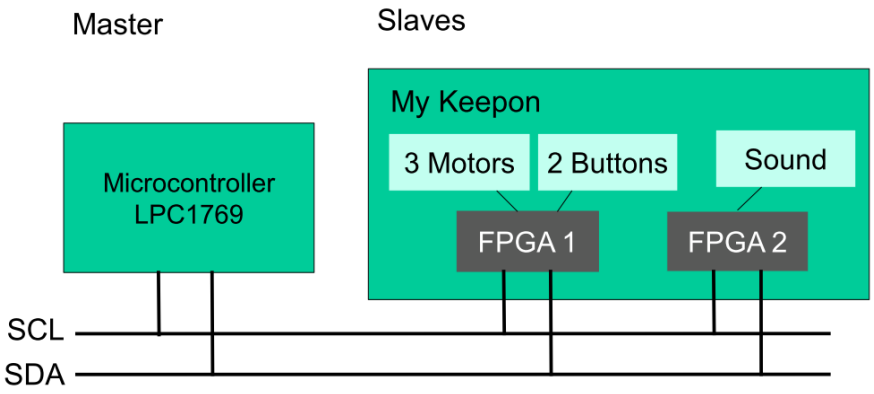

Figure 2.2: Model of the circuit board inside My KeepOn

Thanks to the 4 pins existing on this circuit board and allowing an external intervention, My KeepOn could be hacked using an I^2C bus and a LPC1769 microcontroller (see fig. 2.2 and fig 2.3).

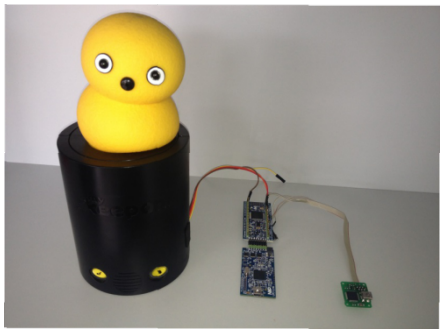

Figure 2.3: Hacked My KeepOn

In order to take control over the two FPGAs, the I^2C bus has to be connected shortly to ground during the power up phase of the My KeepOn. After that, the LPC1769

microcontroller will be the master and it will make both FPGAs to slaves. This microcontroller is equipped with an ARM Cortex/M3 CPU for embedded applications. It provides various peripheral components: in our case we used one I^2C interface and two UARTs buses. Later, we will use this microcontroller to implement the communication user interface between the PC and My KeepOn, and also to prepare the toy robot for the evaluation demo application.

2.1.3 OmniRob

The OmniRob is a moving robot platform with 3 omnidirectional wheels (see fig.. These wheels allow the robot to start at its current position in any direction without having to rotate first. Later My KeepOn will be installed on top. Under the platform there is enough space for batteries, microcontrollers and interfaces for the connection with other devices.

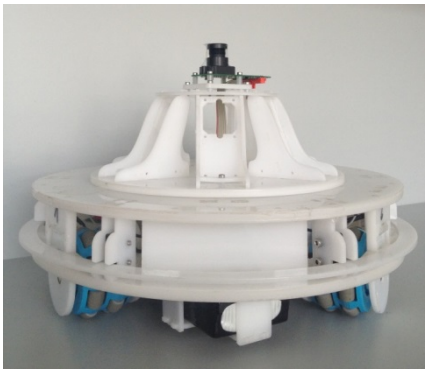

Figure 2.4: OmniRob

2.2 Interconnecting the entire system

To build the whole system consisting of the My KeepOn on top of the OmniRob robot platform and equipped with an event-based vision sensor, we connected all those components via UART to a microcontroller LPC4337. After that, we considered two possible designs. The first one is the autonomous solution, where the system is only controlled by the LPC4337. But for better performance, we could also connect this microcontroller optionally via WLAN to a PC, in order to be able later to develop much more complex demo applications (see fig. 2.5). Finally, we decided to go for the autonomous option. To realize this solution, we constructed a vertical platform on top of the Omnirob and we put My KeepOn inside of it. In order to place the eDVS camera on the head of the toy robot, we chose for symmetry reasons to remove its nose so we can place the eDVS instead of it. The entire system is to be seen on fig.1.1.

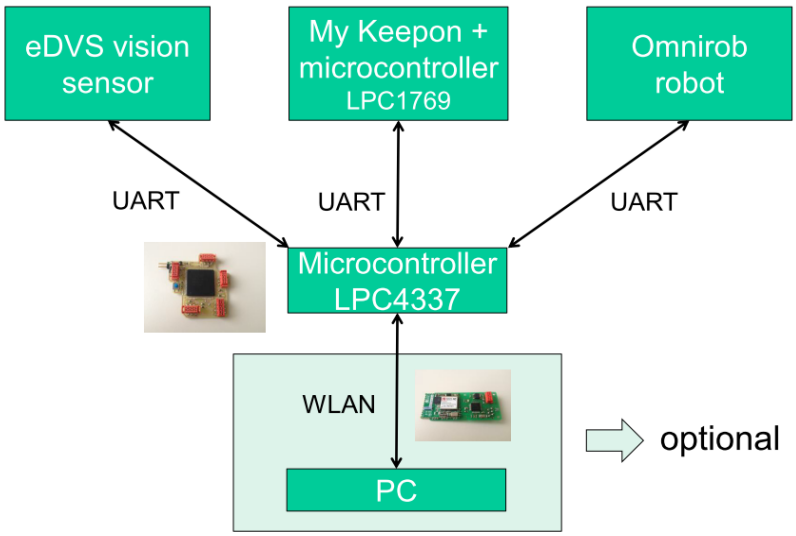

Figure 2.5: Entire system model

3 Implementation

In this chapter, we will present the implementation approaches that we applied for designing the software. But before talking about modifying the existing My KeepOn setup and creating the communication user interface, we will start with explaining the I^2C communication protocol used to hack the My KeepOn.

3.1 I^2C Communication Protocol

3.1.1 General basics

The I²C (Inter-Integrated Circuit) is a multimaster serial single-ended computer bus which is used in our system for the connection between the Microcontroller LPC1769 and the My KeepOn. The physical I^2C bus consists of two main wires: the clock line SCL and the data line SDA. The SDA is responsible for the transferring the data and the SCL synchronizes this process over the bus. In addition, this bus has a ground wire and if necessary a 5V wire. All the devices, in the hacked system are connected with the SCL and SDA and they could have the role of a master or of slaves. Thus, the LPC1769 is the master of the system and the devices of the MyKeepOn, which includes 2 FPGAs, are the slaves. Only the master can control the data transfer and therefore drives the SCL clock line. On the other hand, the slaves are only responding to the commands of the master. The I^2C Physical Protocol begins if the master wants to initiate a data transfer over the I^2C bus. Therefore, the master will issue a start sequence. There are two special sequences defined for the I^2C bus, the start and the stop sequence. During these special sequences, the SDA can change while the SCL is high. During the data transfer the SDA is only allowed to change while SCL is high. One data sequence has a size of 8 bits. After every 8 bits, a 9^{th} bit is sent back by the receiving device (see fig.3.1). With that acknowledge (ACK) bit the slave confirms if it has indeed received the data and is able to accept another data sequence or not.

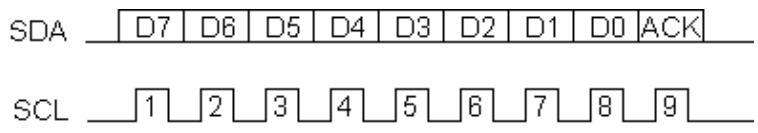

Figure 3.1: Data Transfer over time via an I2C bus

The slaves' addresses consist of 7 bit. For that reason, up to 128 devices can be installed on the I^2C bus. While sending a certain address from the master to a certain slave, a 9th Read/Write (R/W) bit would be required, in order to let the slave know if the master wants to read or to write from it.

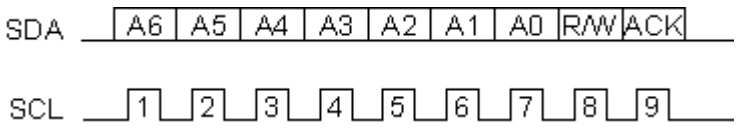

Figure 3.2: Sending an address via an I2C Bus

Hereinafter, the whole procedure of a data transfer between master and slave is described: First if the master wants to transmit data it will release a start sequence. Thereby all slaves will notice that a transfer is started. The master now will send the device address with a Write bit at the end of the sequence and the slave with this address will continue the transfer by sending back an acknowledge bit. Meanwhile, the other devices are not reacting. After the received ACK bit the master will send the data sequence, which includes the internal location or register number inside the slave where it wants to read from or write in. If the master wants to write in the slave, it can send further data bytes and after every byte the slave will send back an ACK bit. The transaction will stop if the master is issuing a stop sequence. If the master wants to read, it will send another start sequence and the device address ending with a read bit. Then it is allowed to read the data from the slave. The transfer will also terminate with a stop sequence. For reading the R/W bit must be 0 and for writing it is 1. The common standard I^2C speed is 100 kbit/s and the low-speed is 10kbit/s [2].

Write to a slave device:
1. Send a start sequence
2. Send the I^2C address of the slave with the R/W bit low for reading (even address)
3. Send the internal register number you want to write to
4. Send the data byte
5. [Optionally, send any further data bytes]
6. Send the stop sequence.

Reading from the Slave:
1. Send a start sequence
2. Send the I^2C address of the slave with the R/W bit low for reading (even address)
3. Send the internal register number you want to read from
4. Send a start sequence again (repeated start)
5. Send the I^2C address of the slave with the R/W bit high for writing (odd address)

6. Read data byte from slave
7. Send the stop sequence.

The bit sequence will look like this:

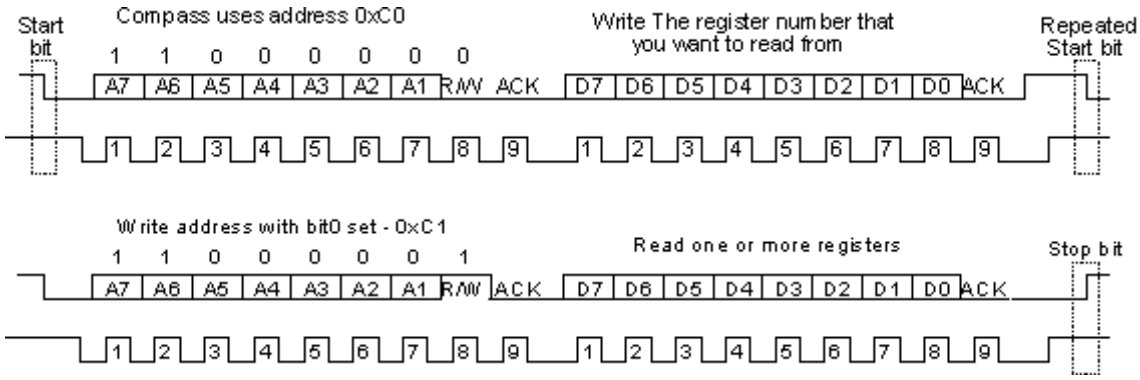

Figure 3.3: Examples of sequences for reading from or writing to a slave

3.1.2 I²C Application in our system

The I²C0 is used for the connection between the LPC1769 and My KeepOn. The pins of I²C0 on the microcontroller require an open-drain pad with an external pull-up and the register settings provide a frequency of 400 kHz (Fast Mode Plus)
For the interface between the My KeepOn and the Microcontroller, the LPC1769 works in the Master Transmitter mode. In this mode the address of the slave and a number of data bytes are transmitted to the slave receiver. There are several states describing a transfer from master to slave. The state 08H defines the start sequence. After the transmission of the I²C address of the slave with a low R/W bit the acknowledge bit tends to the state 18H. The state 28H is defining the acknowledge bit after every data byte (see fig. 3.4). The states 20H and 30H show that the acknowledge bit was not received.

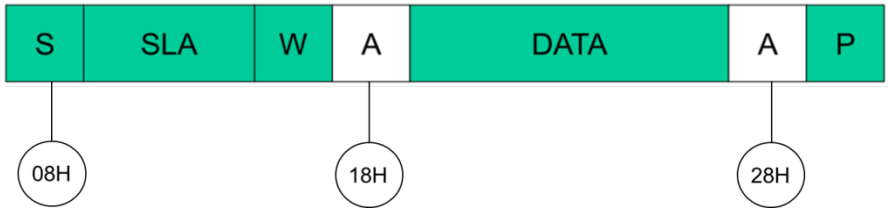

Figure 3.4: Data Transfer between My KeepOn and LPC1769

The My KeepOn slave addresses are 0xAA for the FPGA (P232) which controls the motors' actions and 0xA4 for the FPGA (P234) that controls the sounds. For the possible actions of the actuators, different functions are implemented in the code for the LPC 1769. The internal register addresses are 0x00, 0x02, 0x04 and 0x06 for the motor actions pon, bendTo, turnTo and stop. For playing sounds the register address is 0x01 (see fig. 3.5).

Possible Functions	Possible Inputs	I2C Master Buffer
• playSound(sound)	13 sounds={wakeup, beep, sigh,squatting..}	S xA4 | first byte of data: function address x01 | sound x0-xFF
• bendTo(position)	positions={certain angle, initial, forwards, backwards}	M xAA | bend address x02 | position x0-xFF
• turnTo(position)	positions={certain angle, initial, clockwise, anti-clockwise}	M xAA | rotation address x04 | position x0-xFF
• pon(index)	indexes={up/down fast, up/down slow, left/right}	M xAA | pon address x00 | index x01, x80, xFF
• stop()	no input needed	M xAA | stop address x06 | x10

Figure 3.5: Possible Functions and Inputs in the I2C Master Buffer

For the rotation and bending, values from 0x00 to 0xFF are allowed. Furthermore, the data that has to be send to the slave for bending backwards and forwards needs a value>0x80. On the other hand, for bending to the left/right a value<0x80 has to be send. Furthermore, My KeepOn can turn up to 160 degrees clockwise and up to 160 degrees anti-clockwise. Therefore it has a range of 320 degrees (see fig. 3.6). Besides it is possible to set the motors' speed between the values 1(slow) to 255(fast).

3.2 Communication User Interface

In order to create a communication interface between the user and My KeepOn, we modified the hacked system setup using UART sending and receiving proprieties.
This interface looks like a selection menu displaying all the possible actions that could be chosen. The user could decide which action is to be executed by sending commands to the microcontroller and addressing the desired functions with particular inputs (e.g. bendTo(position)..). All possible functions and inputs are to be seen in fig. 3.5.
After chosen an action, several outputs (e.g. motor's positions, selected actions…) would be displayed on the PC screen.

As we created the user interface, we noticed that My KeepOn has sometimes some problems and struggles while executing some particular commands like bending forwards, turning to the right or using some particular positions as an input. Those errors are not constantly there but they just happen occasionally. To investigate those mistakes we used an I^2C monitor already programmed in a previous work.
The Monitor mode allows observing all I2C-bus traffic, by choosing slave address that you want to have further information about its communication with the master over the I2C bus. With the Monitor mode the I2c traffic and communication between master and slaves is not disturbed. The programmed monitor shows the addresses that are sent from the LPC to the MykeepOn.
It also shows the possible states already mentioned above in the previous section. We could see using the monitor that after sending the register number responsible for those functions the acknowledge bit is not received in these cases. So we concluded that those errors don't happen because of the communication interface code that we added to the existing setup code.

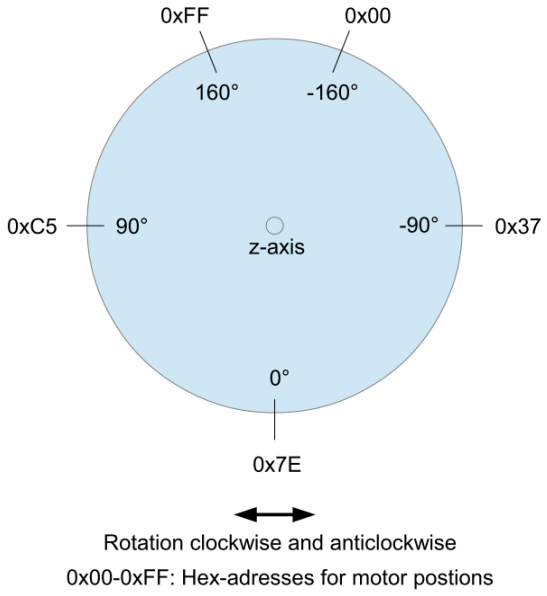

Figure 3.6: Possible positions with their addresses by the function turnTo

We tried to search for possible mistakes in the existing code and we conducted some experiments with the oscilloscope while searching, but we couldn't find yet anything that could be responsible for these problems.
These errors could be later a real challenge for testing the demo application.

3.3 Demo Application

During this practical course, we developed a demo application allowing the My KeepOn's head to localize and track a blinking LED. To simplify the task, we assumed that the LED and the MyKeepOn's head are on the same level regarding the y-Axis. That's why we considered only the x-coordinates while implementing the algorithm. As we explained above, the eDVS receives the coordinates of pixels having a change of brightness. As a consequence, we obtain a visible cloud on the vision map of the eDVS exactly where the LED is blinking. In addition, the eDVS is sending continuously two bytes for one event. First the eDVS has to start with the command "E+". The upper bit of the first byte is always zero to recognize a new event. The next 7 bits of the first byte are the y-coordinate of the event with values between 0 and 127. The second byte contains the x-coordinate of the event also with values between 0 and 127 since the eDVS has 128x128 Pixel. To avoid errors the program controls if the first bit of the first byte of the y-position is zero.
For implementing the localization algorithm, we used a simple approach based on calculating the average a_v of the x-coordinates in order to approximate the position of the LED. For this calculation we used this formula:

$$a_v = (1 - \alpha).a_v + \alpha.x \tag{3.1}$$

with α=0.01

In order to track the LED, we calculate in every step its average position. Based on the average position of the LED and on the current position of the head, we give the MyKeepOn certain commands in order to move its face towards the right direction. We also considered the limited range of angles allowed for My KeepOn's head: 160 degrees on each side (see fig. 3.6). When the desired position is reached, My KeepOn emits a 'Squating' sound.
Furthermore, we modified the setup code of My KeepOn in order to access information in its registers from the new microcontroller LPC4337 which operates the entire system.
Due to time issues and errors existing in My KeepOn setup code, we couldn't test yet properly the demo application that we developed.

4 Summary and Future Work

4.1 Summary

In this practical course, we studied the existing system's components e.g. the My KeepOn, the Omnirob as well as the eDVS. Furthermore we investigated two possible designs for interconnecting and interfacing the whole system, and we decided at the end that the autonomous design solution is more suitable for us in order to have an independent robot. After that, we assembled all of those devices together. We were also able to create a communication interface which allows the user to send executable commands to My KeepOn. While testing this interface, we found that My KeepOn couldn't occasionally execute all possible actions available in the stand-alone mode. These problems could be caused by errors existing in the setup hacking code of the toy robot. Because of these errors, we couldn't test yet the tracking demo application that we've already developed.

4.2 Future Work

As a future work, we propose to ameliorate the existing hacking setup code of My KeepOn in order to eliminate the errors, to have a good performance of all actions offered by My KeepOn and to test properly the demo application that we developed. Furthermore, we propose to create a more complex evaluation application, which can show better the potential of the devices that we used e.g. localizing and tracking a moving human while reacting over different hand motions by using for example My KeepOn's head or by emitting particular sounds.

List of Figures

Bibliography

[1] http://siliconretina.ini.uzh.ch/wiki/index.php
[2] I2C Tutorial, http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html
[3] UM10360, LPC17xx User Manual, NXP Semiconductors