Real-time image processing on an iPAQ based robot (iBOT)

Datalogi 2B Bachelor project
University of Copenhagen (DIKU)

Steffen Nissen (lukesky@diku.dk), Steffen Larsen (zool@diku.dk)
& Sidsel Jensen (purps@diku.dk)

May 23, 2002

Abstract

This project concerns building a modular autonomous real-time image processing robot. The robot is based on LEGO Mindstorms, a COMPAQ iPAQ and a high-resolution camera connected to the iPAQ, which communicate together using infrared signals and PCMCIA. Our goal is to participate in the DTU RoboCup 2002 competition with the finished robot, where the use of a camera as primary input source is totally new. As a result we developed a working, fully autonomous robot with an image processing algorithm based on ridge detection and participated in RoboCup. The image processing algorithm was flawed under extreme lighting conditions, but the robot satisfied the original requirements for the system. We have also written a small API for other robotic developers to use.

Preface

This project is written by Sidsel Jensen, Steffen Larsen and Steffen Nissen. It is a bachelor project for Steffen Larsen and Sidsel Jensen, whereas it is a graduate project for Steffen Nissen.

This project is a part of our education at the Department of Computer Science (DIKU), Copenhagen University. The project took place in the spring of 2002, during a period from February first to May 23rd, with Associate Professor Klaus Hansen as our project supervisor.

Since the report is written both as a bachelor and a graduate project, we have indicated, who have written which sections. If more than one person have written a section, it is shown by both persons, and if all persons have written a section, it is shown by "All":

- Sidsel Jensen (1, 1.1, 1.2, 2, 3, 3.1, 4.1, 4.2, 5.1, 7, 7.1, 7.2, 7.3, 7.4, 8)
- Steffen Larsen (2.1, 3.2, 4, 4.2.1, 4.3, 4.3.1, 4.3.2, 4.4, 4.4.1, 4.4.2, 5.2, 9, 9.1, 9.2)
- Steffen Nissen (2.1, 4, 4.2.2, 5.2, 5.3, 5.3.1, 5.3.2, 5.3.3, 5.4, 5.5, 6.1, 6.2, 6.3, 6.4, 9, 9.1, 9.2)
- All (5, 10)

The source code to the project can be found at the following location at DIKU: `~zool/Projects/bachelor/src`, and images from our image processing during the qualification round, can be found at:
`http://www.hamster.dk/~purple/robot/iBOT/images/`

Acknowledgements

First of all, we would like to thank COMPAQ for the sponsorship of the iPAQ. Without this sponsorship, none of our ideas would have been carried out in life. Also a huge "thank you" goes to developer Jamey Hicks for hacking the device driver for the iPAQ PCMCIA slot (the sleeve). This bug fix would have taken us weeks, if we were to have done it ourselves.

We also had the pleasure of joining a newly created robotics study group at DIKU. This helped us get a lot of new ideas and really good feedback. On that occasion, we would like to thank ph.d student Kim Steenstrup Pedersen from the image group at DIKU and also team Overkill (the other RoboCup team from DIKU).

Last but not least, we would like to thank Klaus Hansen (our project supervisor) for being optimistic yet realistic about our project.

Contents

List of Figures

1 Introduction

This report focuses on the building and use of a real-time image processing system running on an iPAQ based autonomous robot (iBOT), built for the DTU RoboCup 2002 contest.

This paper is written in English so it might reach a broader audience. It is our hope that people (eg. undergraduate students) interested in the field of robotics will be able to use our report as an introduction and gain further interest in the field. The primary goal of this paper is to describe the theoretical and practical elements of the project. The project is split into two sections:

- The practical and mechanical parts of assembling a robot
- The theoretical part behind the programming of the robot

1.1 Motivation

Our motivation for this project comes from the fact that last semester we had a course called "Robotic Experimentation". Throughout the course we dealt with simple image processing and basic robot movement routines. Unfortunately the German produced EyeBots[EyeBot] did not have the computational capacity to do real-time image processing based on camera input. So we thought up a new type of robot based on LEGO Mindstorms and a handheld device with a camera attached.

Our goal for this project is *to build a small modular real-time image processing autonomous robot and to have fun while doing it.* To get the hardware units to work seamlessly together but also, to build a relatively simple API for the robot and to write a small manual describing the API functions, for other developers to use.

We have decided to participate in the annual DTU RoboCup Competition, which will give us a fairly well-bounded problem to solve within the time limit for this project. This will also give us a challenging, but yet well defined image processing problem to solve.

1.2 Structure of the report

The report is structured to follow a natural path all the way from low level hardware to high level decision making. We start out by describing the hardware specific system design (see section 3), and then turn to detailed RCX and iPAQ system engineering (see section 4). This is followed by an thorough description of the image processing algorithms (see section 5). Next we turn to describing the strategy of the robot (see section 6) and evaluate the participation in RoboCup 2002 (see section 7). We end the report with a discussion of general purpose usage of autonomous robots and possible future work (see section 8 and 9).

Each main section is related to a set of demands, that our robot need to fulfill in order to complete the RoboCup course. These demands are listed in the section 2.1, and we refer back to the requirement specification throughout the report, in order to elucidate which demands, the specific section is trying to solve.

Furthermore a full account of the project schedule and the specific installation notes, can be found in our project weblog [WebLog2002].

2 The RoboCup Competition

The annual DTU RoboCup Competition [DTU], held this year on the 24th of April 2002, is a contest for self made robots. Only fully autonomous robots are allowed in the competition. This is the sixth year of the competition. The purpose of the competition is for the robot to follow a course marked by 38 mm white tape on the floor. The course consists of different obstacles of varying difficulty, and the course is laid out so as to make it possible to choose an easy or a hard route between the different obstacles. All the most difficult obstacles are optional, so it is possible to complete the course with a minimum of choices. Distributed everywhere on the course are a number of gates. Crossing each gate gives one point. The first gate is special, though, it closes like a guillotine after a certain amount of time, to ensure that the robots are not too slow. The robot with the largest amount of points (and the shortest completing time) wins the contest.

As you can see on figure 1, there are several different choices, the robot can take during the course. If the robot turns right at every Y-fork (when the tape splits in two), it can however complete the course without having to overcome any extra obstacles like following a wall, follow black or no tape and driving down a staircase. The main obstacles of the course are:

- to follow a wall
- to decide whether to turn left or right at an Y-fork
- to follow black tape
- to drive over a section with no tape at all
- to drive on different surfaces
- to handle both indoor and outdoor lighting
- to go down a staircase

The other participants are mostly engineering students, which implies that their foci are on hardware. No participants before us have been using a camera to detect the white tape, instead they have used an array of light sensors. The use of a camera as "the eye of the robot" in this competition is brand new.

2.1 Requirements specification

While using a sensor-based robot is fairly simple, it is far more challenging to use a camera as guidance for the white tape, because of the more complex output it generates. We chose to use the camera, because we find the challenges far more interesting and fun.

First of all we have to demand that our image processing is processed at real time and is fairly robust. This is because we use a camera as our only input source and that the robot has to process the image, before it takes a specific action.

Another demand that is raised by using a camera, is the varying environments that the course includes; like indoor light, shadows and sunlight from the outdoors. We have to develop a method which eliminates these kind of situations in order to get better image processing and more robustness.

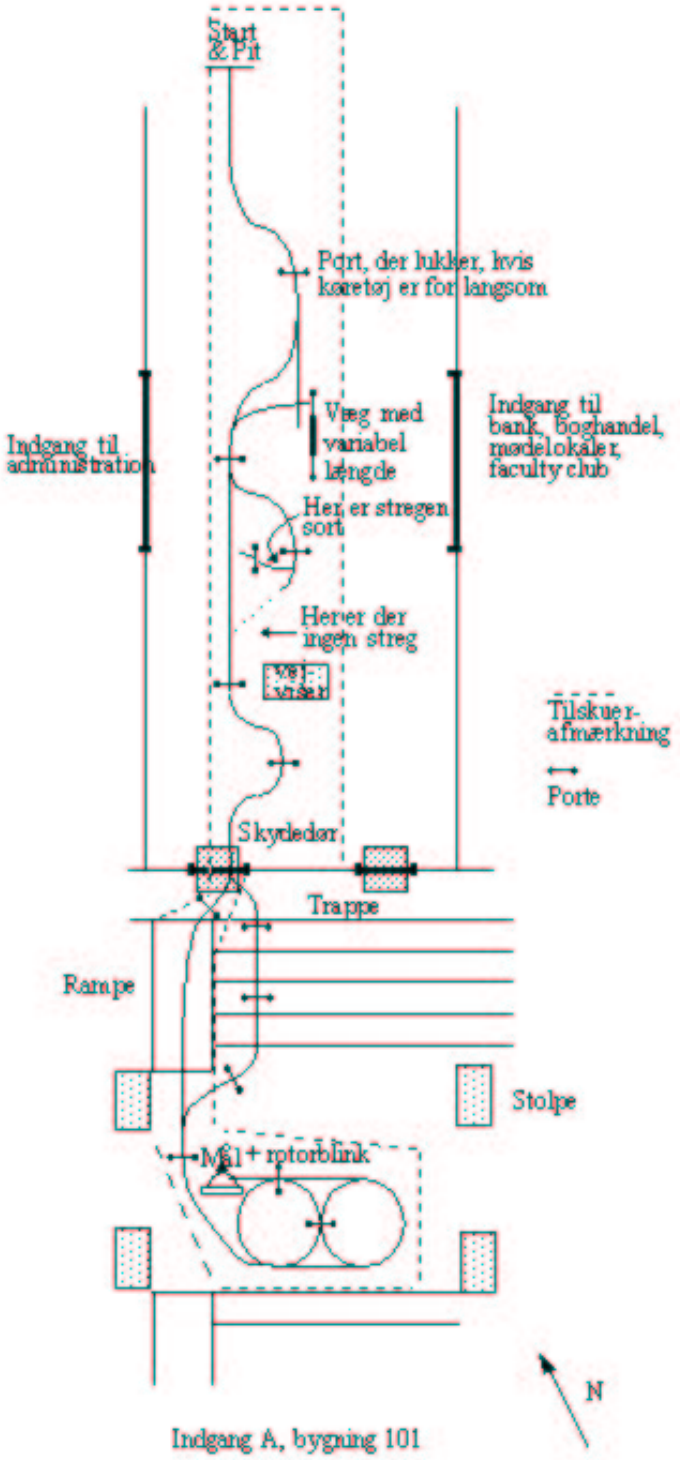


Figure 1: The DTU RoboCup 2002 Course (in Danish).

Because the iBOT is autonomous, there is no means of knowing why the robot is behaving the way it does. To solve this problem, we need extensive debugging facilities on the iBOT, that records the reasons for its behavior. This is also a very useful feature when we need to tune the parameters and algorithms for the given environment.

In the competition our general strategy is to keep it as simple as possible and stay to the right at all the Y-forks (see section 2). We chose this strategy because we have a very tight time limit and the fact that this is the first time we participate in the competition.

Our demands for the iBOT are:

Camera A high quality camera

Detect the tape Image processing algorithms to detect and follow the tape

Real time processing Enough processing power, to detect the tape in real time

Robustness Handle different kinds of environments

Evade obstacles Chose the right path at the Y-forks

Debugging Record the behavior of the iBOT

Speed Make it through the first gate

Time limit The robot must be finished for the RoboCup competition

3 Hardware Design

Since we are not engineers but computer scientists, we wanted to build a robot that did not require any kind of soldering. So we decided to let LEGO Mindstorms[LegoMind] handle all driving routines and motor gearing. Also the LEGO concept itself gave us the freedom of being able to rebuild our robot, as the project progressed and as we saw fit.

3.1 The individual components

During our research for this project, we came across an American research site[VidiPAQ] which had developed video drivers for the Winnov Videum Traveler camera[Winnow] for the COMPAQ iPAQ[Compaq]. This gave us the idea of how to build our robot so it could be small, modular and light weight.

Our robot ended up consisting of the following 4 pieces of hardware:

- LEGO Mindstorms Kit with the LEGO RCX unit
- A handheld COMPAQ iPAQ 3660
- A PC Card Expansion Pack for the iPAQ (a PCMCIA sleeve)
- A Winnov PC Card high-resolution camera

And the following pieces of software:

- LEGO Mindstorms visual programming language
- Familiar Linux v.0.5.1
- LIRC - Linux Infrared Remote Control program
- Video 4 Linux 2 - video devices handling
- Winnov video drivers

The main idea is to let the LEGO RCX unit handle all driving (eg. wheels, motors and the physical form of the robot itself), whereas the COMPAQ iPAQ is the brain of the robot. The COMPAQ iPAQ communicates with the LEGO RCX UNIT through an infrared protocol based on image input fed by the camera connected to the iPAQ (see figure 2). We have chosen the COMPAQ iPAQ, because it is small, fast, contains an infrared port and is extendable with a camera, which satisfies our demands for real time image processing and a high performance camera (see section 2.1).

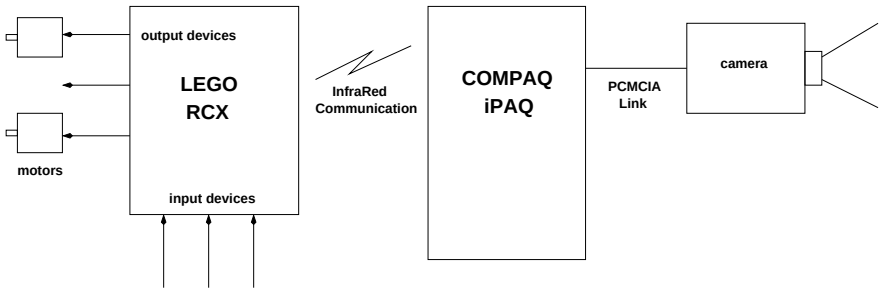


Figure 2: Communication between the iPAQ and the RCX.

The autonomous LEGO RCX unit is based on an 16MHz Hitachi H8 microcontroller. The 8-bit CPU provides most of the control logic for the RCX. The RCX contains 16KB of internal ROM, which is preprogrammed with the system ROM. It also contains 32KB of static RAM. This part of memory holds the firmware, the user programs and any data required by those programs. The RCX uses IR light to communicate with a desktop computer or another RCX. IR communication for the desktop computer is provided by an IR interface that is attached to a standard 9-pin serial port [Baum2000], [RCXmanual].

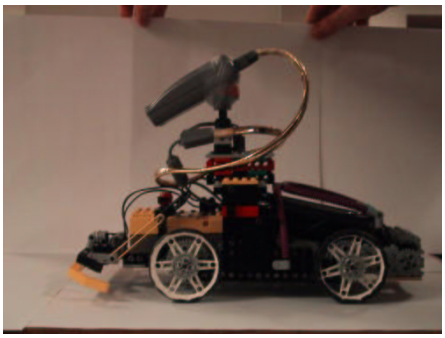
The COMPAQ iPAQ is based on an 206MHz Intel StrongARM SA-1110 32-bit RISC processor. It contains 64MB of RAM and 16MB of flash ROM. It has a TFT screen which can show 4.096 different colors (12 bit). The screen area is size 320x240 pixels. The iPAQ can be attached to a desktop computer by USB (standard) or an optional serial cable. The PC Card Expansion Pack makes it possible to expand the iPAQ with a Type II PC-Card. We chose to expand it with the Winnov Videum Traveler camera.

The camera is a high resolution PCMCIA camera with a maximum resolution of 640×480 pixels, and a maximum frame rate of 30fps at 320×240 pixels. The camera supports many different video formats and has a built-in auto-brightness function.

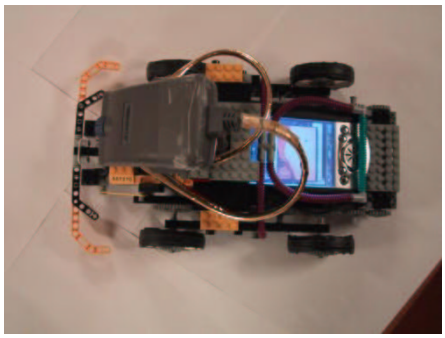
3.2 The iBOT

In this section we shortly describe the resulting physical form of the iBOT, and the modifications we made to some of the hardware to make it work.

First of all the physical appearance of the robot is a direct result of the shape of the iPAQ and the RCX (see figure 3). We had to place the units rather close to each other, to get the IR communication to function properly.



From the side.



From the top.

Figure 3: The iBOT.

The iBOT uses differential steering, because it makes it far easier to navigate. The differential steering uses two synchronous geared wheels for each side. The original LEGO tires, that we used at first, had too munch friction, so we made a new set of tires, which instead of rubber consisted of black tape. This reduced the friction tremendously, so the robot could move more freely on the floor.

speed	~35cm/s
length	32cm
height (with camera)	23cm
width	19cm
ground clearance	1cm
weight	~1Kg
motors	4
wheels	4
bumper(s)	1
other LEGO pieces	A lot!

Table 1: iBOT Data Sheet.

At the beginning of the project, we had some difficulties with the speed of the iBOT. We had to speed up the robot (see table 1), because the first gate closes after a short period of time (see section 2). As a result we changed the gearing and installed two extra motors, so it now uses four, instead of the original two. This was enough to get us through the first gate and fulfill our demand from section 2.1. We also attached a bumper as a last resort for avoiding obstacles.

4 System Engineering

With the physical parts of the robot in place, we can start building a development environment for the iPAQ and the RCX unit. This section focus on the development environments on the two units, and the communication between them.

4.1 Linux on the iPAQ

We decided to install the Linux distribution "Familiar Linux v.0.5.1" on the iPAQ [familiar]. Primarily because this gave us the video and the IR drivers for free, but also because this gave us a scaled down Operating System, with all the development tools and environments that comes along using UNIX. Included in this is the possibility to log onto the iPAQ using a serial cable. These two advantages combined gave us a really strong base, that we could never have gotten by using for example Windows CE (Pocket PC).

We installed Linux by following the instructions at the handhelds website [LiniPAQ]. First we installed the bootloader [iPAQBoot]. Then we installed the root image of Familiar Linux. All this was done by using the serial interface, and the X modem and Z modem capabilities in the console terminal program. The Linux distribution has a small package system (ipkg), that can be used to install system packages. We used this system to install the meta-package *"task-x"*, which includes all the necessary pieces to get a working X server and clients up and running. We also used the package system to install the proper Winnov video drivers, and other packages needed to get a fully functional Linux system. An overview of available ipkg packages can be found at [ipkg].

The design choice of using Linux also gave us the possibility of using standard Linux tools, like Linux Infrared Remote Control (LIRC) [Lirc] and Video 4 Linux 2 [V4L2]. The camera drivers operate by the Video 4 Linux 2 API, which is a new and improved API for video devices. It is also more flexible and extendable, and it supports more kinds of devices. The driver we are using was originally developed at the University of Michigan [VidiPAQ].

4.2 Development Environment

Having the software and hardware in place, we needed a functioning development environment for the iPAQ. This chapter goes into detail about the development tools we used for the iPAQ and the LEGO RCX unit.

4.2.1 Developing for the iPAQ

We developed all our software using a StrongARM C++ cross compiler (GNU g++) for the iPAQ [LiniPAQ]. Hence all our code was developed on a desktop machine, cross compiled for the StrongARM and transfered to the iPAQ. Some hacking was needed in the Makefile in order to get it to work properly though (see appendix C.8), because we had some problems with the linker and the X-11 libraries.

Developing for the StrongARM platform introduced some limitations. Due to the lack of a floating point unit in the iPAQ architecture, we were forced to use fixed point algebra instead of floats in all central loops. This was done because the floating point

emulation was far too slow for our needs. The square root function was also re-written to fixed point algebra [ARMcode] implemented in `int_math.h` (see appendix C.5).

4.2.2 Debugging the Camera

Since the camera is a central part of our robot, and image processing is a central part of our software, it is essential that we can easily debug the camera and the image processing algorithms.

Our most important debugging facility for the camera, is the `xcaptest` program. This program displays what the camera sees in real time, and is the best way to determine whether the camera is working properly.

In order to debug the image processing algorithms, we save images during the different steps of the algorithms. These images can then be transferred to our desktop computer using the Z-modem protocol, in order to understand the robots behavior. We are also able to view the images directly on the iPAQ using the Quick Image Viewer (`qiv`) program. This actually fulfilled our requirements of extensive debugging facilities (see section 2.1).

Furthermore we have made some modifications to our main program `robot` (see appendix C.2), which gives us the opportunity to run the image processing algorithms directly on our desktop computer. Since we do not have a camera attached to our desktop computer, the image processing on the desktop computer is done using a pre-taken picture. This feature makes debugging specific elements of our image processing much faster, since we can tune constants and constraints, and see the results immediately.

4.2.3 Developing for the RCX unit

Our original plan was to replace the standard LEGO firmware with legOS[legOS]. LegOS is an Open Source project which provides a C and C++ programming environment for the RCX. This would enable us to write the software for both the iPAQ and the RCX completely in C++, which we would prefer. We also looked shortly at the "Not Quite C" [NQC] environment for the RCX. (figure 4 shows how the RCX unit works with different kinds of firmware).

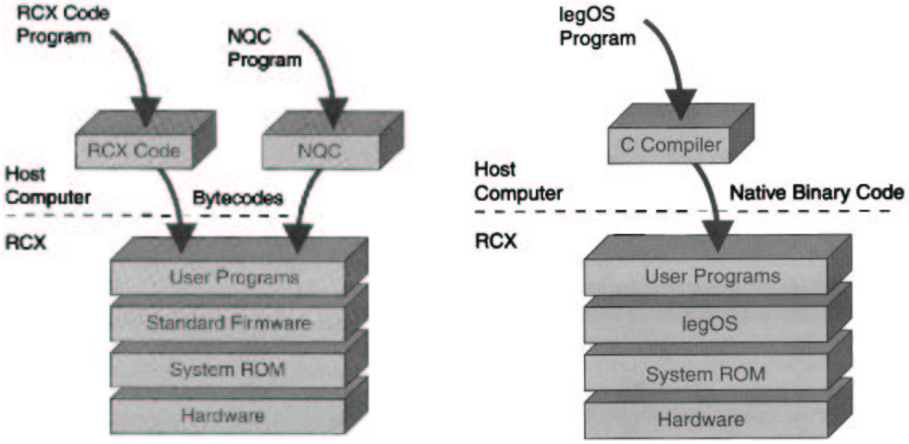
This original choice however, resulted in some difficulties and we ended up using the standard LEGO programming language. This was also well suited because of the simplicity of our need.

4.3 Communication with the RCX

This section deals with the specific infrared communication between the RCX unit and the iPAQ. We tried several options before we reached our final solution.

At first glance the IR communication seemed to be no problem at all, because we assumed that both units (the iPAQ and the RCX) communicated via some kind of standard irDA protocol. This was, however, not the case. LEGO have intentionally used some kind of proprietary protocol for the RCX instead of the standard irDA.

Our first idea was to reverse engineer the whole thing ourselves, but due to the tight schedule, we were forced to take a shortcut. We got the idea that we could use a LEGO remote control to record the IR commands it would send to the iPAQ, when we pressed



RCX with standard firmware. RCX with legOS.

Figure 4: The RCX with different firmware

the command buttons on the remote. We could then use the iPAQ as a remote control for the RCX.

We used the feature in the RCX unit which allows it to send and receive IR signals to another RCX unit. These commands are called "Send to RCX" and "RCX sensor watch" in RIS, LEGOs own programming language (see appendix B). They give the RCX the ability to send a message (a number between 0 and 255) to another RCX. This number can then act as a direct trigger, to activate another set of commands on the receiving RCX.

What we wanted, was to develop a program on the iPAQ that emulated another RCX unit (and remote control). This way we would be able to monitor and send IR signals from the RCX to the iPAQ. Of course the other way around is, also a possibility. This would enable us to receive and perform certain tasks on the RCX; like "stop", "drive forward", "drive backward", "turn left" and "turn right".

In order to make these things possible, we found a package called LIRC for the iPAQ that allowed us to decode and send infrared signals of many common remote controls. The most important part of LIRC is the lircd daemon that can decode IR signals received by the device drivers and provide that information on a socket. By using this socket the clients can get the infrared codes received by the lircd daemon and they can send commands to the lircd daemon. We can then decode the signal from the LEGO remote control and record them on the iPAQ.

We first tried to use a small program called `irrecord`, which is included in the LIRC package. This is a program which interactively tries to record remote control signals and create a config file for the lircd daemon. However, we could not get the iPAQ to receive the correct IR signals from the RCX. The biggest problem was that `irrecord` thought the RCX unit worked as a standard remote control like one for TV, VCR or DVD. Instead, we had to figure out how the IR communication for the RCX unit actually worked.

4.3.1 RCX Encoding and Transmission

The RCX uses a 38kHz carrier line, which is quite typical for TV remotes and other standard remotes. As for the sampling rate, the RCX runs at 2400 bps, which makes each bit approximately 417μs long (1/2400 bps) [IR-Communication]. We quickly discovered that the IR transmitted bytes using the following scheme:

- Byte encoding is 1 start bit, 8 data bits, odd parity bit, 1 stop bit (11 bits in all).
- Start bit is "0" and coded as a 417μs pulse of 38kHz IR.
- Data bits "0" is coded as a 417μs pulse of 38kHz IR.
- Data bits "1" is coded as 417μs of nothing.
- Stop bit is "1" and coded as 417μs of nothing.

The scheme used to transmit RCX packages (see section 4.3.2) results in an equal number of zero bits and one bits, allowing a receiver to compensate for a constant signal bias (eg. caused by ambient light) simply by subtracting the average signal value. Note, that the header also has an equal number of ones and zeros; this "warms up" the receiver before the real data arrives. This may be due to the lack of any other means of synchronism in the protocol.

4.3.2 RCX Package Level

Next we were to find out how the IR transmitter actually sent and received the data (protocol). We discovered that the IR protocol is fairly simple by using some really good information from Kekoa Proudfoot[RCXinternals]. An IR package consists of the following sequence:

```
(0x55 0xff 0x00) D1 ~D1 D2 ~D2 ... Dn ~Dn C ~C
```

The initial section (inside the parenthesis) is the header for any packet transmission. 0x55 is 1010101 in binary, 0xff is 11111111 and 0x00 is obviously 00000000 meaning that there is the same number of ones and zeros. The middle portion of the data (D1 to Dn) are the opcodes you want to send to the RCX unit, followed by the complement of the opcode. The two final bytes are an one byte checksum and its complement. You can not send the same message twice to the RCX, since the RCX never executes the same opcode two times in a row, so every opcode comes with an alternate opcode.

The checksum is very simple. It is the sum of the packet data eg.

```
C = D1 + D2 + D3 + ... + Dn
```

If you want to send the message of say 0x12, the packet looks like this:

```
(0x55 0xff 0x00) 0xf7 0x08 0x12 0xed 0x09 0xf6
```

First comes the packet header followed by 0xf7 which indicates that the next part is a message. 0x08 indicates that it is a 8 bit message. Next comes the message number itself eg. 0x12 and the complementary of 0x12. 0x09 and 0xf6 are the tail of the packet (the checksum). Actually each byte is not 8 bits, but rather 11 bits because of the start, stop and the parity bit (see section 4.3.1).

4.4 Sending RCX Commands

After doing a lot of intensive research on the IR communication, we realized that our main problem was that `irrecord` expected a gap between the IR commands. All "normal" remote controls has a pause in its signal, so you can identify where IR commands start and stop. The RCX does not pause between messages. This gave us a real headache – until we realized the problem. The LIRC tools thought the pause was the 0xFF of the header, since it was a long silence (see section 4.3.2). The result of this, was that when `irrecord` automatically generated the config file, it only generated data for the header up until 0xFF, but not for the rest. It never got to the data part of the package. Once the problem had been identified, we changed the config file to use raw codes, and used the program `mode2` (included in the LIRC package), to record full raw messages sent from the RCX. The full message was converted by using our perl converting program (see appendix D), into the configuration file in appendix F.

With this configuration file in place, we were now able to send commands to the RCX with the program `rc` (also a part of LIRC) and receive command from the RCX with the program `irexec`. We hacked the `rc` program so it fitted our needs and included it in our `sendRCX.cc` file (see appendix C.4). For further information about the API defined in `sendRCX.cc` see appendix A.

4.4.1 Sending from the iPAQ

With the working implementation of `sendRCX.cc` we were now able to define an API. What we basically needed was only motor commands. We defined and developed a set of messages on the RCX which corresponded to the messages it could receive and react on, from the iPAQ (see appendix B). The code for the RCX is "implemented" in LEGOs own programming language. We used "RCX sensor watchers" to receive commands from the iPAQ and "send message" to send messages back to the iPAQ. We also sent back the message "10". This hack was done because `irexec` only executed the command, when it received the next command.

The RCX API commands that can be sent via IR from the IPAQ to the RCX (see `sendRCX.cc` in appendix C) are:

- Stop
- Backward
- Forward
- Turn right
- Turn left

4.4.2 Sending from the RCX

Due to the lack of stability and robustness in the IR communication, we were forced to send the packages containing the messages every time we processed an image. This took a lot of CPU time from our iPAQ, because of the time it uses to send out IR pulses (38000 times per second), which again affected our frame rate. We had to make a little protocol to make the messages less time (CPU) consuming.

When the RCX unit receives a motor command, it sends back the same command to the iPAQ. This is then received by `irexec` (see appendix E) on the iPAQ and echoed into a file, called `/tmp/recieveBot`. By looking at the last command we have received in the file, we can reduce the number of commands we send via IR from the iPAQ. This actually enhances the performance on the iPAQ, because we now only send new commands instead of one for every processed image frame.

When the bumper on the RCX unit is pressed, it also sends back a message. This message is also received by `irexec` on the iPAQ which creates a file, called `/tmp/bumper`. We then check whether the file exists or not in order to determine if the bumper was pressed. This is used in our strategy (see section 6.1), to determine whether the robot drove into an unforeseen obstacle.

5 Image Processing

One of the major tasks in this project is to manipulate the immense information stored in the pictures taken with the camera into something usable. This means reducing the picture into relevant information or rather *image processing*.

The main purpose of the image processing is to detect the 38mm white tape on the floor. This is actually more difficult than it seems. We have identified a number of problems that we are required to solve. The main problem is connected to the fact, that we have absolutely no control of the environment. Certain problems could be:

Over exposure The image can be overexposed, due to sunlight.

Highlight Spots Spots or blitz can interfere in the image.

Environment We have some features in the environment which looks like the tape.

We also need to be able to rely on our image processing because the camera is our only input source, therefore we will try to make sure that our image processing is as robust as possible.

5.1 Image Processing Idea

Several strategies were available for detecting the tape, eg. edge detection, but we chose a different algorithm, called ridge detection [Lindeberg1996].

An overview of the basic image processing idea, can be seen in figure 5. The first thing that happens is that the camera takes a picture. This picture is then normalized to give better contrast. We then apply a Gaussian filter over the entire image, which smoothen the image a great deal. The Gaussian smoothened image is then sent through our ridge detection algorithm, which identifies possible ridge points on the top of the ridge. The ridge points gives an indication of where we can find the white tape. Each ridge point is then added to a box, and each box is then added to a component. Several boxes are added to the same component, if they are adjacent to each other. We then find the largest connected component and calculate the center of mass and angle for the specific component and drive towards that.

We could have chosen to implement an edge detection algorithm instead, but this would have introduced us to a new set of problems. For example too few edges in the resulting picture leads to the fact that the edges break up into smaller ones, which are

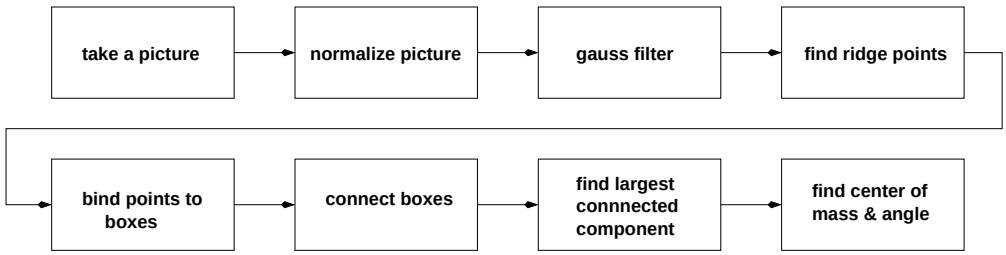


Figure 5: Overview of the image processing idea.

not connected. Too many edges on the other hand, leaves us with too much information that need new filtering to be usable. So we ended up choosing the ridge detection algorithm over edge detection.

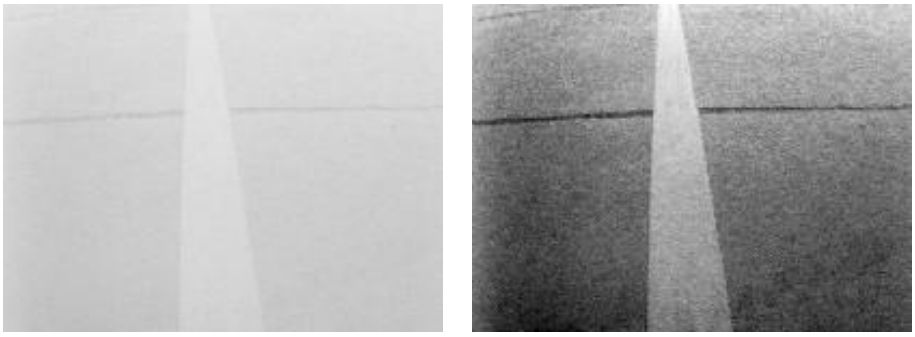
In the following sections we will go into detail about the different steps in the image processing algorithm.

5.2 Preprocessing

First we take a picture, but this picture is not always perfect for our purpose, so we normalize the picture to get better contrast. The problem with many of the outdoor images is that they are overexposed, which implies that there is a very small difference between the white tape and the surroundings. A simple and very effective solution is to normalize the image.

At first we actually tried to use the built-in brightness function (`setBright`) in the camera, but it turned out that normalization was more efficient and easier to control, so we chose to do normalization.

We normalize the image, by first calculating the upper and lower bound for the intensity of the pixels. We then use these boundaries to re-calculate new pixel values in the complete range from 0 to 255, and we now have a much higher contrast in the image. A good example of this can be seen in figure 6.



Original image. Normalized image.

Figure 6: Normalizing an overexposed image.

Then we apply a Gaussian filter to the image by means of convolution [Convolution].

We do this in order to reduce noise in the image and make the white tape appear as a smooth ridge that we detect with our ridge detection algorithm (see section 5.3).

The Gaussian filter is a bell shaped filter, which means that it gives a more gentle smoothing than eg. a mean filter. This more gentle smoothing reduces the noise in the image, without loosing important information about the ridge. To generate the 2-dimensional Gaussian filter we use equation (1) [Gauss].

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

Where σ is the standard deviation of the distribution. An example of the Gauss distribution can be seen in figure 7.

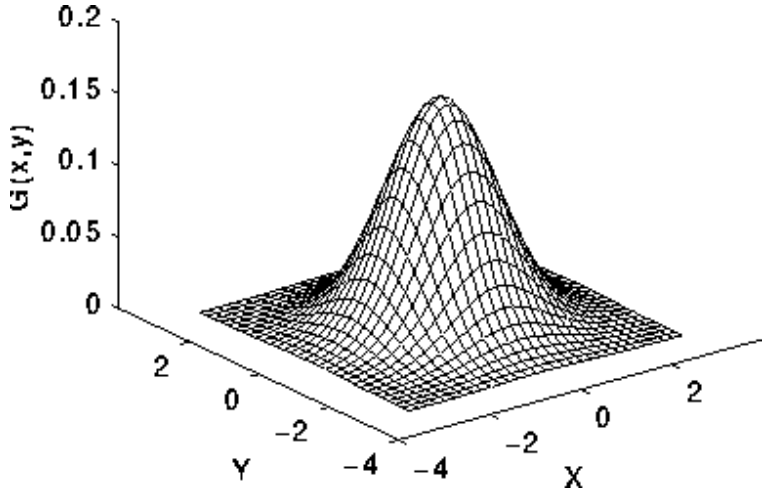


Figure 7: Gauss distribution where σ is 1.

Since the image is stored as a collection of discrete pixels, we need to produce a discrete approximation of the Gaussian function before we can perform the convolution. This will effectively create a symmetric matrix that we can apply to the image.

The Gaussian matrix gives us two variables to tune; the σ (the spreading) and the size of the Gaussian matrix. The size of the matrix is of course dependent on the value of σ.

The σ-value determines how large features that are turned into a smooth ridge, eg. if we are looking for a wide piece of tape, we will use a large σ-value etc.

We use the σ value to sharpen the ridge for use in our ridge point detection (see section 5.3). This is done, because ridge point detection is designed, to look for points that lie completely on the top of the ridge. This means, that the sharper the top of the ridge is, the easier it is to detect.

Further information about our Gaussian smoothing implementation, can be found in section 5.5 and in the files `gauss.cc` and `gauss.h` (see appendix C.6 and appendix C.7).

5.3 Ridge Point Detection

At this point the image has been smoothed so much by the Gaussian smoothing (see section 5.2), that the white tape now looks like a smooth ridge. It is possible to find

the top of this ridge and the direction of it. In this section we will explain how this is done.

5.3.1 Vertical ridges

Let's first look at how we would detect a bright ridge if it was completely vertically oriented. By a vertical ridge, we mean a ridge that is parallel to the y-axis. A cross section of this would look like figure 8.

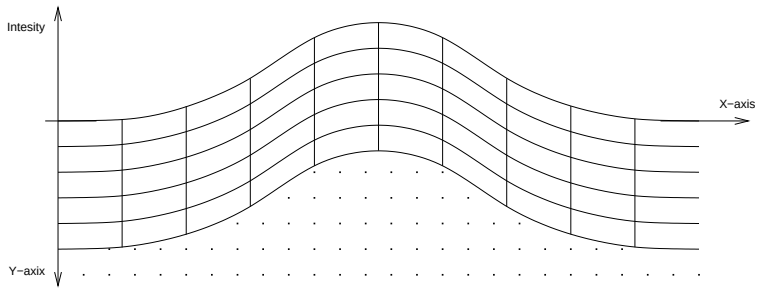


Figure 8: Cross section of a completely vertical ridge.

If we look at the derived values (the x and y gradients), we see that the first derived in the y-direction (L_y) is always zero. But the first derived in the x-direction (L_x) is a bit more interesting (see figure 9). L_x makes a zero crossing exactly at the top of the ridge. If we were to look only for vertical ridges, we could look for zero crossings in L_x. But since we are not only looking for vertical ridges, this is not an option. More information about derived values and differential geometry can be found at [Morse2000-1] and [ParDiff].

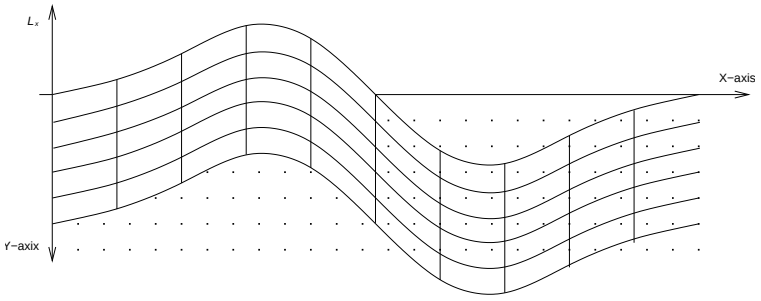


Figure 9: Cross section of L_x in a completely vertical ridge.

If we look a bit closer at figure 9 we can see that the L_x value drops rapidly from a value above zero, before the ridge top, to a value below zero, after the ridge top. This steep downwards hill in the L_x value, can easily be seen in the second derived x value (L_{xx}). In figure 10 it can be seen that the L_{xx} value is very small at the top of the ridge.

Now we have all the the components needed to find a ridge, we can define some criteria that will determine if we have a ridge, or if we do not have a ridge.

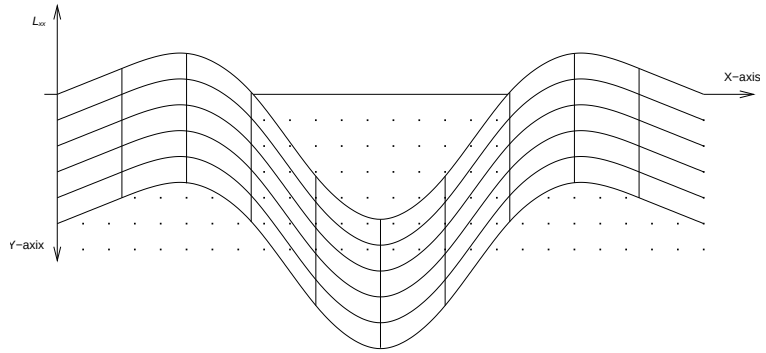


Figure 10: Cross section of L_{xx} in a completely vertical ridge.

There are many different ways to determine if there is a ridge or not, but we have chosen the one defined in [Lindeberg1996]. This method requires the following constraints to be met.

$$\begin{aligned} L_x &= 0, \\ L_{xx} &< 0, \\ |L_{xx}| &\geq |L_{yy}| \end{aligned} \tag{2}$$

Since L_x is zero precisely at the top of the ridge, L_{xx} is much lower than zero at the top of the ridge and L_{yy} is zero across the whole ridge, all of the constraints in (2) will be met precisely at the top of a perfect ridge.

Since we only need to find large ridges, and we do not require that the ridges are perfect, we have altered the constraints a bit. We do not require that L_x is completely zero, but we do require that $|L_x|$ is below a threshold k. In this way we will find ridges that are not completely perfect, which is also very rare in images because of their discrete nature. In order to find only large ridges, we have tightened the constraint for L_{xx}, and we now require, that it is below a threshold $-g$. The last constraint have also been tightened, by requiring that $|L_{xx}|$ is a factor f greater than $|L_{yy}|$. All these changes give the following constraints.

$$\begin{aligned} |L_x| &< k, \\ L_{xx} &< -g, \\ |L_{xx}| &\geq |L_{yy}| * f \end{aligned} \tag{3}$$

5.3.2 All ridges

We now have the full set of constraints to detect bright vertical ridges, but since we need to be able to detect ridges that point in any direction, we need to introduce a

local (p, q) coordinate system, where p is parallel to the ridge and q is orthogonal to the ridge.

In [Lindeberg1996] it is clearly described how we can introduce this new coordinate system. The angle β, in which the new coordinate system and thereby also the ridge is pointing, is given by (4) and (5). These two equations use the mixed second-order derivative L_{xy}.

$$\cos\beta\,|_{(x_0,y_0)} = \sqrt{\frac{1}{2}\left(1+\frac{L_{xx}-L_{yy}}{\sqrt{(L_{xx}-L_{yy})^2+4L_{xy}^2}}\right)}\Bigg|_{(x_0,y_0)} \tag{4}$$

$$\sin\beta\,|_{(x_0,y_0)} = (signL_{xy})\times\sqrt{\frac{1}{2}\left(1-\frac{L_{xx}-L_{yy}}{\sqrt{(L_{xx}-L_{yy})^2+4L_{xy}^2}}\right)}\Bigg|_{(x_0,y_0)} \tag{5}$$

With the angle β calculated, we can calculate the first derived L_p, L_q and the second derived L_{pp} and L_{qq}. The sketch for how this can be calculated is given in [Lindeberg1996]. The equations for these are given in equation (6), (7), (8) and (9).

$$L_p = \sin\beta * L_x - \cos\beta * L_y \tag{6}$$

$$L_q = \cos\beta * L_x - \sin\beta * L_y \tag{7}$$

$$L_{pp} = (\sin\beta)^2 L_{xx} - 2\cos\beta\sin\beta L_{xy} + (\cos\beta)^2 L_{yy} \tag{8}$$

$$L_{qq} = (\cos\beta)^2 L_{xx} - 2\cos\beta\sin\beta L_{xy} + (\sin\beta)^2 L_{yy} \tag{9}$$

With the first and second derived values calculated for the local coordinate system, we can do the same calculations for the (p, q) coordinate system as we did for the (x, y) coordinate system (see equation (3)), and find all bright ridge points in this way.

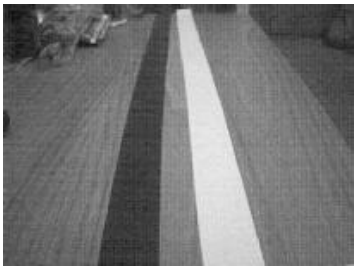
The calculations, that we have used to find bright ridge points, can also be used to find dark ridge points, but for this application we only look for bright ones. In figure 11 the result of running the ridge point detection algorithm on both dark and bright ridges can be seen.

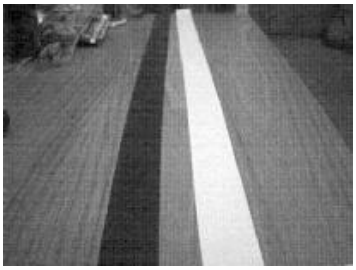
5.3.3 Differential Mathematics in Images

In section 5.3.1 and 5.3.2 we assume that we can calculate the derived values L_x, L_y and the second derived values L_{xx}, L_{yy} and L_{xy}. But since we do not have the exact equations for the x and y curves, we need to find an approximation.

There are several ways to approximate L_x and L_y. They all use convolution of a kernel on an image (see [Convolution]). You could use the Prewitt kernel or the Sobel kernel as suggested in [Morse2000-2], but we use more simple kernels (see figure 12). Since the image has been smoothened by the Gauss, it should be just as good to use the simple kernel, as it would be to use the more complex kernels.

These kernels give equation (10) for L_x and equation (11) for L_y where $i(x, y)$ is the intensity function.


Original image.


Normalized image.


Gaussian smoothed image.

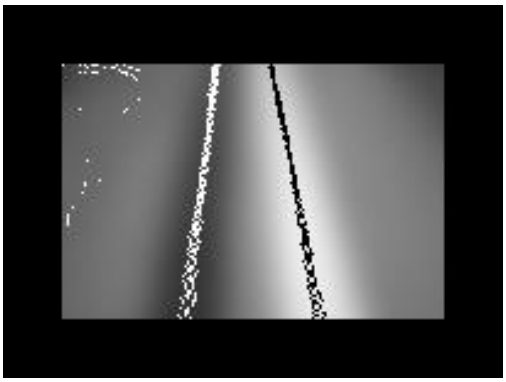


Figure 11: Illustration of the ridge point detection algorithm. Dark ridge points are identified by white dots and bright ridge points are identified by black dots.

-1	0	1

-1
0
1

Figure 12: Kernels for calculating L_x and L_y.

$$L_x(x,y) = i(x+1,y) - i(x-1,y) \tag{10}$$
$$L_y(x,y) = i(x,y+1) - i(x,y-1) \tag{11}$$

With the equations for L_x and L_y in place, it is straight forward to calculate L_{xx}, L_{yy} and L_{xy}, as seen in equation (12), (13) and (14).

$$\begin{aligned} L_{xx}(x,y) &= L_x(x+1,y) - L_x(x-1,y) \\ &= (i(x+2,y) - i(x,y)) - (i(x,y) - i(x-2,y)) \\ &= i(x+2,y) + i(x-2,y) - 2*i(x,y) \end{aligned} \tag{12}$$
$$L_{yy}(x,y) = i(x,y+2) + i(x,y-2) - 2*i(x,y) \tag{13}$$
$$L_{xy}(x,y) = i(x-1,y-1) + i(x+1,y+1) - i(x+1,y-1) - i(x-1,y+1) \tag{14}$$

With these calculations in place, it is very simple and fast to calculate the first and second derived values.

5.4 Post-processing

When all the ridge points have been identified, we need to connect them into one big ridge. We could do this by using a conventional connected component algorithm [Connected], but because of the somewhat limited computational power of the iPAQ, compared to a normal PC, and the fact that ridge points does not always lie adjacent, we have chosen another method.

We have also chosen to limit the information that we need for a ridge, to its center of mass and its angle.

Instead of having a complete representation of all ridge points, we have chosen to use a discrete representation. We have separated the image into 12×9 boxes. Each time we detect a ridge point, we add it to one of these boxes. Each box have a counter that keeps track of the number of ridge points that have been added, furthermore it has three sum variables, one for the cosines values calculated in equation (4), one for the x positions and one for the y positions. In this way, we still have all the information necessary to calculate the center of mass and the direction of the ridge, but we throw away information about the individual ridge points.

We run a connected components algorithm like the one described in [Connected] on all the boxes that have more ridge points, than a selected threshold. We do however have some additional constraints, which make sure that two boxes with angles that point in very different directions can not be connected. We do not write the information back to the boxes, when we combine two connected components. The reason for not writing this information back to the boxes is that we do not need to know which boxes are part of the connected components, but only their combined angle and center of mass.

Now we have a set of connected components, each with a center of mass and an angle. Then all we need to do, is to find out which of these connected components that defines our ridge and drive in that direction. An example of how the connected component for a ridge looks like can be seen in figure 13.

5.5 Image Processing Implementation

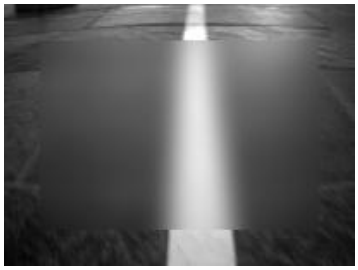
The iPAQ has a 206 MHz Intel StrongARM SA-1110 32-bit RISC processor, which is fast for most robotics computation, but since our image processing is very demanding, we were not certain how it would perform.

We decided to use an image resolution of 204×152 pixels (8 bit of gray-scale), and we quickly realized that a naive implementation of the image processing algorithms, would be far too slow for the iPAQ, so we were forced to make several different optimizations. In section 4.2.1 we describe that we use fixed point algebra for all central computations, this optimization was completely necessary if we wanted to get a decent frame rate.

The two most time-consuming image processing operations in our program is the Gaussian smoothing (see section 5.2) and the ridge point detection algorithm (see section 5.3).

Original image.

Normalized image.



Gaussian smoothed image.

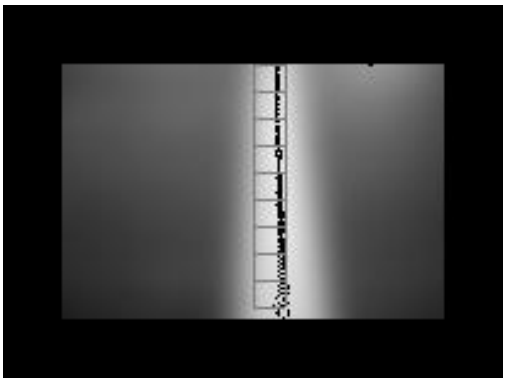


Figure 13: Illustration of the connected component algorithm.

The Gaussian smoothing operation has been optimized in two different ways. First of all we use two one-dimensional Gaussian kernels, instead of one two-dimensional kernel, which changes the running time from $O(n^2)$ to $O(n)$ where n is the size of the Gaussian kernel. Secondly we did some constant propagation and loop unrolling, by writing a program (see appendix C.6), that generated an include file `gauss.h` for C++ (see appendix C.7). In this way, the Gaussian smoothing was made very fast, and we could change the size of the Gaussian kernel without major changes in the frame rate.

The ridge point detection algorithm was primarily optimized by the use of fixed point algebra. But we also made sure that all operations were done in one big loop, instead of first calling one function that generated the L_x values, then one that generated the L_y values etc.

All of these optimizations gave us a frame rate of 5-6 frames per second, which was fast enough for our need.

6 Behavior of the iBOT

6.1 Robot Strategy

For the RoboCup competition (see section 2), there can be many different strategies. We choose the simplest strategy, which is to turn right in all Y-forks. In this way, we would not be able to drive through more than 7 gates, but given our limited time period, completing the course with 7 gates, would be considered a great success. This

is in accordance with our requirement of evading obstacles (see section 2.1).

If we had not chosen this strategy, we would have needed information about the complete topology of the course, in order to complete the race successfully. This would have required far more complex programming, which we did not have the time to implement.

If we lose sight of the tape, our strategy is to drive backwards. This strategy is based on the consideration that when we drive back, we get a broader view of the course, and we will probably be able to detect the tape that we lost.

We have changed this behavior a bit, to make the robot more robust. Instead of just driving backwards whenever we lose the tape, we allow the robot to drive forward in special situations. We do not want the robot to get stuck in an infinite loop, where it first drives forward following the tape, then drives backward because it looses the tape, and so on. We have solved this problem, by incrementing a counter by two each time we drive back, and decrementing the same counter by one each time we drive forward. In this way the counter keeps growing as long as the robot is in the infinite loop, and we can simply refuse the robot to drive backwards for a while after we have detected an infinite loop.

If we are to hit something with the bumper, we drive back for a couple of frames, and then we hope that we will see something, that will make us drive around the obstacle.

6.2 Choosing the Right Component

When we have processed an image, we have a set of connected components. Most of the time we only have one connected component like in figure 13. But if we have more than one component, we will have to find out which component identifies the ridge. This is not as simple as one might think, you can not always just naively select the largest component.

The other components in the picture will usually be minor elements on the ground or highlight spots. The largest problem is the highlight spots, because they can often be more intense than the white tape. This can cause the highlight spot to be the largest component, which in turn can cause the robot to miss the white tape.

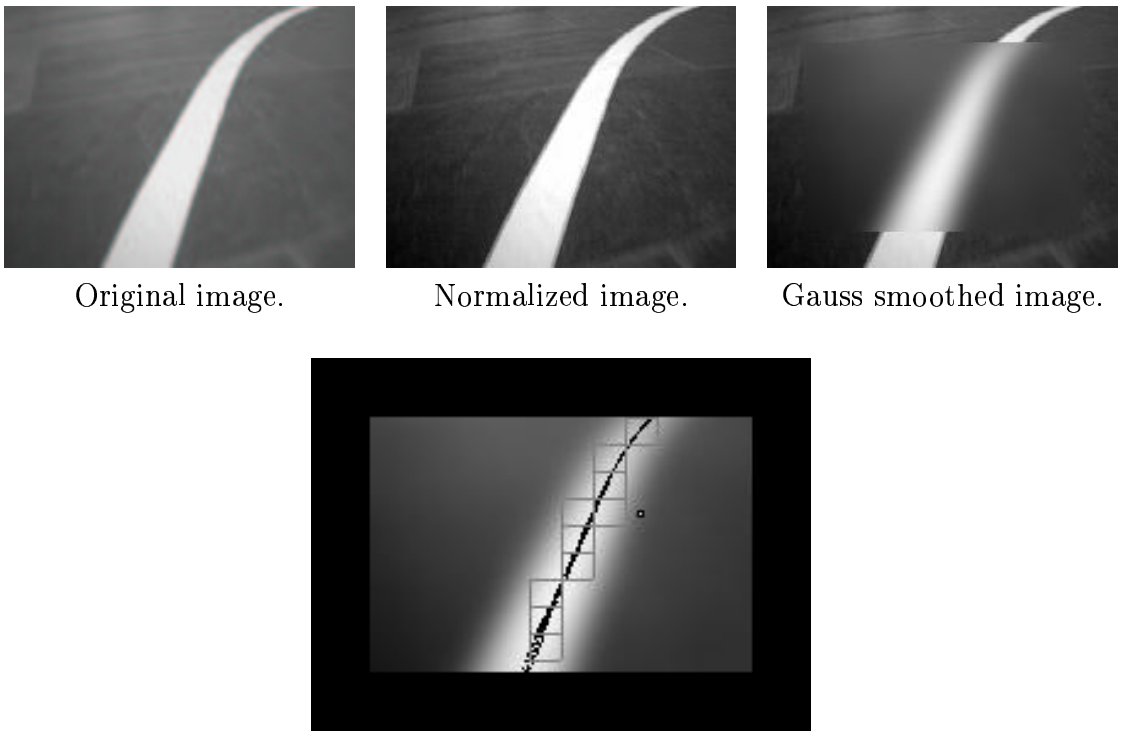
Many different things can be done to make sure the right component is chosen. One solution that we found to be somewhat effective was to introduce a weight for a component. This weight is a combination of the number of boxes the component connects and the number of ridge points it connects. Furthermore we noticed that the highlights were mostly located in the top of the image, while the white line was most active at the bottom of the image. This information was added to the weight, by giving a bonus for boxes located near the bottom of the image.

6.3 Following the Tape

When we have selected the correct component, we need to find out in which direction the robot should move. The information that we have available to us is the center of mass for the component, and the angle of the component. Let's first take a look at how these are calculated. Since the ridge is larger in the bottom of the image, there are also more ridge points at the bottom of the image. Therefore we have chosen to give a bonus to the boxes located near the top of the image. This bonus is not the same

bonus that was given to the weight in section 6.2, but a bonus that only has an effect on the placement of the center of mass and the angle. With respect to this bonus, we now have a center of mass and an angle. This center of mass and angle is more sensitive to variations at the top of the image, than at the bottom. This give us the opportunity to react faster to curves in the tape.

We use the angle to shift the center of mass to either sides. In this way, if the tape turns toward the right, the center of mass is shifted to the right, and if the tape turns toward the left, the center of mass is also shifted to the left.



Original image. Normalized image. Gauss smoothed image.

Figure 14: Illustration of shifting the center of mass. The center of mass is illustrated by a white dot with a black ring around it.

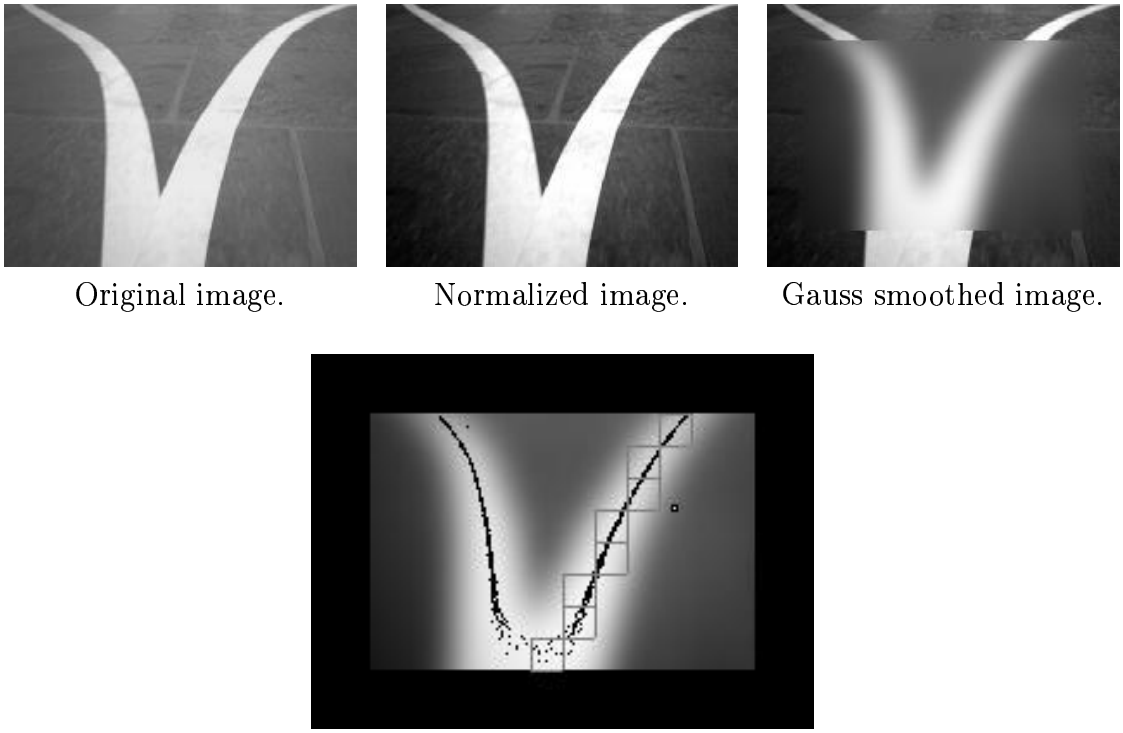
In figure 14 it is easy to see that the center of mass has been shifted to the right because of the angle of the tape.

We use this new point to drive toward, and since the robot has only three different forward modes (forward, turn-left and turn-right) (see section 4.4.1), we have divided the image into three equally large vertical regions. If the new point is in the left region, we drive to the left, and so on.

6.4 Y-Forks in the Tape

Y-forks is when the tape splits, and leaves a left and a right path. When we encounter a Y-fork we should always choose the path to the right (see section 6.1). The problem is, that if we were to use the connected component algorithm described in section 5.4, we would get one big component, with no information about forks etc.

In order to make the robot always choose the right fork, we have chosen a very simple solution. Whenever we encounter a fork during the connected component algorithm, we throw away everything to the left. In this way we will never choose the left path. This strategy works for our specific purpose, but as a general purpose strategy it is not recommended. An example of how we process y-forks can be seen in figure 15.



Original image. Normalized image. Gauss smoothed image.

Figure 15: Illustration of how our connected component algorithm processes Y-forks.

7 Evaluation of RoboCup

Looking back at the RoboCup competition it was a success, even though a couple of things went wrong. Unfortunately, one of the problems was that our robot was not as robust to changes in the environment, as some of the other robots in the competition.

A lot of things were different on the competition day itself compared to the pre-qualification:

- The tape was worn, it was not as white as it could have been
- The course was illuminated by spotlights (we turned them off)
- The course was scattered with sponsor items
- The journalists flashed their cameras right into the camera of the robot
- The power was turned off, where we charged the robots batteries

7.1 Round 1

Round 1 went relatively well. We passed three gates, before the robot started having problems with the direct sunlight, coming from the outside. The robot drove rather fast, but still steadily along the white tape. It did not "wiggle" along the tape like some of the sensor based robots. At the end, right after the third gate, the robot started having trouble finding the tape. It lost sight of the tape several times, which resulted in a lot of backwards driving (see section 6.1). After a while it started to drive forwards toward a spot with no tape. We assume that it was driving towards a highlight spot. Immediately before this happened, it was driving towards a very sharp left turn towards the fourth gate. Unfortunately it rampaged into some sponsor books instead and at this point, we asked the judges to stop the time.

7.2 Round 2

Round 2 did not go well at all. Without our knowledge, the power source we were connected to at our table had been turned off. This resulted in power problems on the robot, since we did not have the time to charge the robot properly before round 2. The camera had a very large power consumption, so if we did not charge it, the robot started behaving oddly, and the camera turned off, leaving the robot blind. This happened right before Round 2, and we could not get the robot operating in time, so we had to withdraw it from the second round.

7.3 The Result

Our final position in the competition can be seen in table 2. We got 5 points in all. 3 points for passing through 3 gates, and 2 points for finishing in less than 2 minutes. We were hoping to pass 4 gates, so our expectations were almost met. We could possibly have ended up one position better, if we had stopped our robot a little bit sooner. "Overkill", the other DIKU participant, passed one more gate than us, giving them a total of 6 points. The winning team Aalbot, got 18 points in all. 14 points for passing through gates, 2 points for hitting the goal and 2 points for finishing in less than 2 minutes. They only missed one gate. The defending champion Crazy Ivan, had optimized their robot too much, so it cut corners and drove straight into one of the gates. At the pre-qualification, 2 days before, they had successfully passed all gates, gaining 17 points (out of a total of 19), but unfortunately the robot was too slow to get any time points. Several of the robots were participating for the second or even third time. "Den Sovende Planet" won the fighter price, whereas "Sigma Contra Ventum" won the price for best DTU vehicle and "Klods Hans" won the price for best design.

7.4 Discussion

The two primary problems we had during the competition was, that we did not do enough to filter out highlight spots and that the power to the iPAQ was cut, so the batteries was not charged properly before Round 2. The power consumption problem, we had no way of handling, except to move the charge point to another power outlet. The problem with filtering out highlight spots, we did however have the possibility to solve.

Name	Points 1.Round	Time 1.Round	Points 2. Round	Time 2.Round	Position
Troublemaker	12	2:17	10	2:00	3
Thrane&Thrane Robot Klub	7	1:12	6	1:03	9
The Snail	8	3:54	12	5:28	5
The Punisher	6	1:59	7	2:22	11
tEAM iBOT(us)	**5**	**1:35**	**-**	**-**	**14**
Speedipus Rex	8	1:29	12	3:00	4
Sigma Contra Ventum	12	1:56	6	1:03	2
Overkill	5	1:08	6	1:19	12
Madam Skrald II	3	0:26	4	0:29	16
Lovewagon	7	1:43	6	1:27	10
Klods Hans	7	1:03	7	1:01	8
Den Sovende Planet	2	0:17	5	0:50	13
Den Nemme Udvej	7	1:45	9	3:02	6
Cyclops	5	2:43	2	0:23	15
Crazy Ivan	8	1:25	4	0:35	7
Aalbot	18	1:24	18	1:26	1

Table 2: Scoreboard from the RoboCup 2002 Competition

The problem typically arose in three situations. One of them was when a highlight spot was identified as the largest connected component in the picture. This could happen if the highlight "extended" the tape straight ahead, where in fact the tape was turning, making it look like a Y-fork in the tape, and making us miss the turn. It was especially bad, in those cases where we had to take a left turn, because of our strategy to always take a right turn in forks.

The problem also occurred if we had lost the white tape, but there was a large highlight spot in the picture, thus making the robot believe that it had found the tape again, when in fact it was only light. This tended to send the robot on some wild goose chases. It was even worse on some surfaces than others, depending on how much the light was reflected off the floor.

A third instance of this problem, was when the robot saw both tape and highlight, but the highlight spot was the largest connected component in the image. If the highlight spot was identified as a valid component, it generally shifted the center of mass in the wrong direction. We then turned away from the tape, and had trouble finding it later on.

An account of the three problematic scenarios:

- The highlight extends the tape, making it look like a Y-fork
- No tape, but a highlight in the picture
- Both highlight and tape in picture, but the highlight is the biggest component

The best way to make the robot more robust against these problem instances would be to do better filtering and identification of highlights. This means improving our image processing algorithm substantially. A really good way to improve the algorithm, would be to tighten the constraints on possible ridge points. Thus filtering out more "wrong" ridge points. An alternate solution could also be, to change the box model entirely, into an algorithm that looks solely on individual pixels. This would make the model a lot more sophisticated and take things such as the shape of the highlight into account. The sole reason why all this was not implemented, was of course, lack of time.

8 General Discussion

Now we have described all the individual details in the project, but we still lack putting the use of our robot into a larger perspective. Why is it even interesting to bother about robotics? What purpose do they fulfill?

Autonomous robots as a whole can be used for many different things. One of the most popular uses is in space research. We all know the Mars Explorer robot named Pathfinder. But what if it did not need to be guided by man? What if it could avoid obstacles on its own and gather materials from Mars without human interference? It would have been a lot more efficient, that way. Now, and in the future, robots will be used as forced labor, as the Czech name "robot" itself implies. Our own robot can also be used for a more general purpose. It could do object foraging, landmark recognition, mission planning or simply play games like tag or robotic soccer.

The result of this project is a small RCX API that can be used by others, who want an autonomous RCX based robot. With the API, it is possible to do basic driving routines, based on image input. The image processing algorithm itself can easily be changed into something else, thus changing the entire purpose of the robot. In fact our original plan for this project was to program the robot to play robotic soccer.

It is neither harder nor easier to play robotic soccer, than it was to participate in RoboCup. It really depends on how sophisticated the image processing algorithm is. We could have implemented a crude image processing solution, which would match the information gathered by the sensor based robots, but we chose not to. One of the interesting properties of driving with a camera, was how far ahead we could see. We also had a much better chance of finding the white tape again, had we lost it. The sensor based robots, had trouble finding the tape, once they had lost it.

An even more elegant solution, would have been a re-reinforced learning algorithm. Letting the robot drive through the RoboCup course several times, and thus letting it build an internal map of the course, and which way to select to complete the course. A simpler solution, would have been to just store the entire map of the course in the robot, and enable the robot to check the map at any given time, so it would always know its own position on the course. This, however would have taken an extra amount of time to develop, but unfortunately we did not have this time.

9 Future Work

Based on this project, many new projects can be started. It would be natural, to develop other applications for the iBOT like robotic soccer etc. (see section 8). But

because of all the fun we had in the RoboCup competition, we would like to participate again next year with a new and improved robot.

The areas that we would like to improve are first of all, the image processing, which needs to be far more robust to the changing light conditions during the competition. Secondly we would like to be able to complete more of the advanced challenges; like following the wall and going down the stairs (see section 2).

9.1 Image processing

Many different solutions are possible, in order to make the image precessing more robust. First of all we could decrease the resolution of the image, which will decrease the CPU usage and allow more advanced image processing to be applied. Secondly the ridge point detection algorithm could need some more tuning, and a better debugging environment has to be developed in order to identify the parts of the algorithms that can be tuned. Last but not least, the post-processing needs to be more detailed to better be able to determine where the ridge is, and what direction the robot should drive.

With a more detailed post-processing, we would have much more information available, that can determine which part of the image represents the tape. This can be done in several different ways like on the basis of shape, size, location of the component etc. Another idea could be to memorize the last location of the ridge in the picture. This way we can predict where the ridge might be in next picture, and use this information to eliminate features or highlight spots, that look similar to the tape (see section 5).

With a more robust image processing, we can turn to developing a more advanced strategy. A more advanced strategy could include information about the topology of the course, so the robot is aware of its whereabouts. This would help us in the effort of completing more RoboCup challenges.

9.2 Hardware

With the camera as our only input source, we were of course very limited in our strategy. We are really encouraged to add more hardware to the robot. If we would like to complete challenges like the "the wall", next year, we have to add some sort of a distance sensor like a sonar or a IR device. To complete "the stairs" (mentioned in section 2) we have to build a more robust robot with more ground clearance (see section 3.2). For this use, LEGO is probably not the best choice of chassis for the vehicle, because of its fragile nature.

We would also like to get a more precise control of the robot. This is not currently possible with the LEGO Mindstorms motors, because they do not have enough torque. Two obvious new designs for an enhanced robot, would be:

- To use more powerful motors, to gain more control of the differential steering.
- To use servo-steering on the front wheels, to gain more precise control of the robot and to get a much more direct effect out of the engines.

The camera could also be enhanced in several different ways. The camera could be positioned on a higher turret on the vehicle or perhaps a turnable turret, and thus giving it a broader view. We also have the choice of adding more cameras to the robot

or fit the camera with a zoom lens. The downside of adding more cameras is of course that the image processing requires even more CPU processing.

With more cameras and more advanced image processing, we are forced to either decrease the resolution of the images, or simply replace the iPAQ with a more powerful unit. This would however also require that we use a more powerful and stable battery unit.

These are some of the ways that we can enhance our robot, for use in the RoboCup 2003 competition. Other enhancements like installing light sensors, that look directly down on the tape might also be a possibility, but because we do not find that interesting, we still wish to rely on the camera as our primary input source.

10 Conclusion

We originally intended *to build a small modular real-time image processing autonomous robot and to have fun while doing it.* The process of building the robot, was a lot of hard work, but we did have a lot of fun while doing it. The participation of the DTU RoboCup 2002 was the culmination of the project. At first it was unsure whether we had the time needed to complete the robot before the competition, but we pulled through.

Technically we achieved our own goal and we have a fully functional autonomous robot, that can do real-time image processing. The robot can follow a white line quite fast and smooth, although it still has problems with various light conditions. We also completed the simple RCX API for other developers to use.

When we started out, our basic skills in image processing were relatively poor, since none of us, had taken any image processing classes. This meant we had to research on image processing [HIPR2] and the mathematics behind [Mathworld], before we could develop the image processing.

One month and twenty days before the RoboCup competition, we received the last parts for the iBOT. Twenty days later when we signed up for the competition, we did not yet have any means of communication between the RCX and the iPAQ, which put us under a lot of pressure.

So on the day of the competition we behaved like proud parents when our robot drove through the first gate. This was the first year, that somebody entered the competition with a camera-based robot, so even though we only passed three gates, we were quite satisfied. A lot of the other participants were interested in our use of the camera as our primary input source and gave us a lot of credit for this solution.

11 References

References

System design

[Baum2000] Dave Baum et al.
"Extreme Mindstorms: An Advanced Guide To LEGO MINDSTORMS". Apress, 2000.

[familiar] the Familiar Project
"http://familiar.handhelds.org"

[iPAQBoot] iPAQ Bootloader Installation Instructions
"ftp://ftp.handhelds.org/pub/linux/compaq/ipaq/stable/install.html"

[ipkg] ipkg Packages
"http://oxy.lcs.mit.edu/familiar/releases/"

[IR-Communication] Mindstorms IR-Communication
"http://baserv.uci.kun.nl/~smientki/Lego_Knex/Lego_electronica/IR_tower/IR_tower.htm"

[legOS] legOS API
"http://legos.sourceforge.net/"

[LegoMind] LEGO Mindstorms
"http://www.mindstorms.com"

[LiniPAQ] Linux on iPAQ
"http://www.handhelds.org"

[Lirc] Linux Infrared Remote Control
"http://www.lirc.org"

[NQC] Not Quite C
"http://www.enteract.com/~dbaum/nqc/index.html"

[RCXinternals] RCX Internals
"http://graphics.stanford.EDU/~kekoa/rcx/"

[RCXmanual] RCX Manual (hardware guide)
"http://www.legolab.daimi.au.dk/DigitalControl.dir/RCX/Manual.dir/RCXManual.html"

[V4L2] Video for Linux Two
"http://www.thedirks.org/v4l2/"

[VidiPAQ] Video on iPAQ (winnov drivers)
"http://pads.east.isi.edu/"

[Winnow] VideumCam Traveler (the camera)
"http://www.winnov.com/products/discontinued/vidcamtraveler.htm"

Image processing

[Connected] Connected components
"http://www.dai.ed.ac.uk/HIPR2/label.htm"

[Convolution] Convolution
"http://www.dai.ed.ac.uk/HIPR2/convolve.htm"

[Gauss] Gaussian Smoothing
"http://www.dai.ed.ac.uk/HIPR2/gsmooth.htm"

[HIPR2] Image Processing Learning Resources
"http://www.dai.ed.ac.uk/HIPR2/index.htm"

[Lindeberg1996] Tony Lindeberg
"Edge Detection and Ridge Detection with Automatic Scale Selection" International Journal of Computer Vision 30(2), 117-154, 1998.

[Morse2000-1] Brian S. Morse
"Lecture 11: Differential Geometry" Bringham Young University 2000.
"http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MORSE/diffgeom.pdf"

[Morse2000-2] Brian S. Morse
"Lecture 13: Edge Detection" Bringham Young University 2000.
"http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MORSE/edges.pdf"

[ParDiff] Partial Differentiation
"http://web.mit.edu/wwmath/vectorc/scalar/partial.html"

Other

[ARMcode] ARM code square root routines
"http://www.finesse.demon.co.uk/steven/sqrt.html"

[Compaq] COMPAQ homepage
"http://www.compaq.com/"

[DTU] DTU RoboCup 2002
"http://www.robocup.dk".

[EyeBot] EyeBot homepage
"http://www.ee.uwa.edu.au/ braunl/eyebot/"

[Mathworld] Eric Weisstein's World of Mathematics
"http://mathworld.wolfram.com/"

[WebLog2002] Sidsel Jensen & Steffen Nissen & Steffen Larsen
"Weblog on robot bachelor paper - spring 2002" Copenhagen University, DIKU.
"http://www.hamster.dk/~purple/robot/iBOT/weblog/"

A RCX API for the iBOT

This appendix describes what developers need to do, in order to communicate through IR with the RCX from the iPAQ, using our API. We presume that the lircd daemon is configured like described in section 4.4.

We have created a class called `ComRCX`, which can be found in `sendRCX.h` (see appendix C.4). The constructor of this class takes no parameters. The only method that lies in this class is `void sendRCX(ComRCX::command)`. `ComRCX::command` is an enum which can be the following:

stop Sends a stop signal to the RCX

forward Sends a drive forward signal to the RCX

backward Sends a drive backward signal to the RCX

turnleft Sends a turn left signal to the RCX

turnright Sends a turn right signal to the RCX

The `ComRCX` class uses two-way communication with the RCX, so that commands already received by the RCX is not sent again. This reduces the CPU time used for sending commands to the RCX.

B RCX Source Code

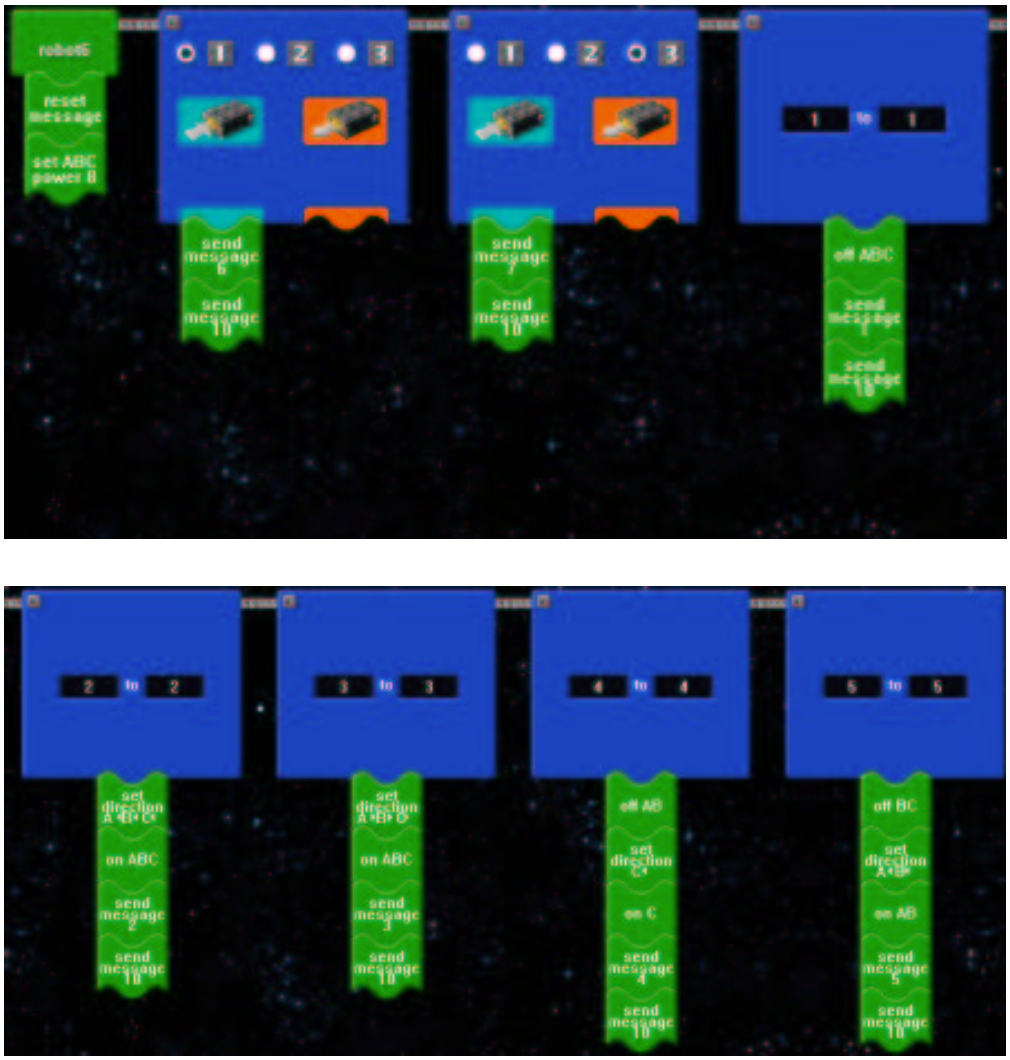


Figure 16: The RCX code

C Source Code

C.1 robot.h

```
#ifndef _ROBOT_H_
#define _ROBOT_H_

// height and width off our image
const unsigned int width = 204;
const unsigned int height = 152;
const size_t size = width*height;

typedef unsigned char image[height][width];

#include "gauss.h"

const unsigned int filtersize = 5 + GAUSS_SIZE;
const unsigned int edgesize = (filtersize/2);
const unsigned int innerwidth = width-(2*edgesize);
const unsigned int innerheight = height-(2*edgesize);

const unsigned int boxnumberwidth = 12;
const unsigned int boxnumberheight = 8;
const unsigned int maxcomponents = boxnumberwidth*boxnumberheight;

const unsigned int boxwidth = innerwidth/boxnumberwidth;
const unsigned int boxheight = innerheight/boxnumberheight;


const unsigned int frames_before_backward = 0;                                  // 120;

// how much weight a componet should have to be a component
const unsigned int weight_min_component = 500;

const unsigned int backthress = 14;
const unsigned int noback_for = 20;

//the number of positive pixels in a box
const unsigned int boxthress = 4;

//reward for each box a component spans
const unsigned int reward = 10;

//reward for each box a component spans
const unsigned int reward2 = 100;

// 90 degree (PI/2)
const float radian_angle = 1.5714285714;

/*
  Box class
  Used to add points to a box
*/
class Box
{
public:
    Box():sum_cos(0),sum_h(0),sum_w(0),count(0),compnumber(0){}
    int sum_cos;
    unsigned int sum_h;
    unsigned int sum_w;
    unsigned int count;
    unsigned int compnumber;
    bool canConnect(Box &b)
    {
        float boxrad = acos(b.sum_cos/(b.count*256.0));
        float rad = acos(sum_cos/(count*256.0));
```

```
        //check to see if angle less than radian_angle
        if(((boxrad < rad) ? (rad - boxrad) : (boxrad - rad)) < radian_angle){
            return true;
        }
        return false;
    }
    void addPoint(unsigned int h, unsigned int w, int cos, int sin)
    {
        sum_h += h;
        sum_w += w;
        if(sin < 0){
            sum_cos -= cos;
        }else{
            sum_cos += cos;
        }
        count++;
    }


};

/*
  Component class
  Used for adding boxes into a component
 */
class Component
{
public:
    Component() :
        sum_cos(0),
        sum_h(0),
        sum_w(0),
        count(0),
        weight(0),
        center_h(0),
        center_w(0),
        compnumber(0),
        leftbranch(false)
    {}
    void addBox(Box &b, unsigned int bheight)
    {
        unsigned int factor = (boxnumberheight - bheight);
        sum_h += b.sum_h * factor;
        sum_w += b.sum_w * factor;
        sum_cos += b.sum_cos * (int)factor;
        count += b.count * factor;
        weight += (b.count + reward) * (bheight+1) + reward2;
    }

    void addComponent(Component &c)
    {
        sum_h += c.sum_h;
        sum_w += c.sum_w;
        sum_cos += c.sum_cos;
        count += c.count;
        weight += c.weight;
    }

    bool canConnect(Box &b)
    {
        float boxrad = acos(b.sum_cos/(b.count*256.0));
        float rad = acos(sum_cos/(count*256.0));
        //check to see if angle less than radian_angle
        if(((boxrad < rad) ? (rad - boxrad) : (boxrad - rad)) < radian_angle){
            return true;
        }
        return false;
```

```
    }

    //Call this function before trying to access the finished component.
    void finishComponent()
    {
        if(count){
            //we shift the center of mass according to the angle
            float rad = acos(((sum_cos/(float)count))/256.0);

            // ((130/pi)*(rad-pi/2)) ... shift a maximum of 60 pixels
            // we only want an angle of 180 degrees = pi
            int shift = int(41.380*(rad-1.5708));

            center_h = (sum_h/count) - edgesize;
            center_w = (int)((sum_w/count) - edgesize) + shift;

        }
    }
    int sum_cos;
    unsigned int sum_h;
    unsigned int sum_w;
    unsigned int count;
    unsigned int weight;

    //center_h and w has zero in the inner image
    int center_h;
    int center_w;

    unsigned int compnumber;
    bool leftbranch;
};

/*
  Boxes class
  A container object for boxes
*/
class Boxes
{
public:
    void addPoint(int h, int w, int cos, int sin)
    {
        boxes[(h-edgesize)/boxheight][(w-edgesize)/boxwidth].addPoint(h, w, cos, sin);
    }

    Component findRidge(image im);

    //Clears the boxes for use in next image
    void clear(){
        memset(this, 0, sizeof(*this));
    }
private:
    Box boxes[boxheight][boxwidth];
    Component connected[maxcomponents];
};

/*
  Robot class
  Our general class for take and write an image image. Also our main
  image processing algorithm (findRidgepoint) is located here
*/
class Robot
{
public:
    Robot() :
        framecounter(0),
        timecounter(0),
        lasttime(0),
```

```
        backcounter(0),
        noback(0)
    {
        starttime.tv_sec = 0;
        starttime.tv_usec = 0;
    }
    void writePPM(char *data);
    void takePicture();
    Component findRidgePoints (image im, image outim);
    ComRCX::command processImage(bool write);
private:
    timeval starttime;
    unsigned int framecounter;
    int timecounter;
    int lasttime;
    unsigned int backcounter;
    unsigned int noback;
    image image1;
    image image2;
    Boxes boxes;
};

#endif
```

C.2 robot.cc

```
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/mman.h>
#include <errno.h>
#include <iostream.h>
#include <fstream.h>
#include <string.h>

#define WRITEPIC
#define FPS

/* These are needed to use the Videum driver */
#ifndef NOROBOT
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/videodev.h>                                    /* Video for Linux Two */
#endif

#include "sendRCX.h"

#include "robot.h"
#include "int_math.h"

#ifdef NOROBOT
char filename[255];
#endif

#ifndef NOROBOT
//The V4L2 device.
int vid;

//The V4L2 format.
struct v4l2_format fmt;
#endif

//Writes an image to a file in the tmp directory
void Robot::writePPM(char *data)
{
    timeval tv;
    gettimeofday(&tv, 0);
    char buf[100];
    if(lasttime == tv.tv_sec){
        timecounter++;
    }else{
        timecounter = 0;
    }
    lasttime = tv.tv_sec;
    sprintf(buf, "/tmp/pic_%ld_%d.ppm", tv.tv_sec, timecounter);
    FILE *fd = fopen(buf, "w");
    fprintf(fd, "P5\n%d %d\n255\n", width, height);
    int bytesleft = size;
    while(bytesleft){
        int byteswritten = fwrite(data + (size - bytesleft), 1, bytesleft, fd);
        if(!byteswritten){
            break;
        }
        bytesleft -= byteswritten;
    }
```

```
    fclose(fd);
}

//Takes a picture
void Robot::takePicture()
{
#ifndef NOROBOT
    size_t n = read(vid, (char *)image1, sizeof(image));
    if (n < 0) {
        fprintf(stderr, "read() returned error %d\n", errno);
        exit(1);
    }
#else
    ifstream imagefile(filename);
    imagefile.getline((char *)image1, sizeof(image));
    imagefile.getline((char *)image1, sizeof(image));
    imagefile.getline((char *)image1, sizeof(image));
    imagefile.read((char *)image1, sizeof(image));

#endif

#ifdef FPS
    if(framecounter%10 == 0){
        if(framecounter ≠ 0){
            timeval endtime;
            timeval difftime;
            gettimeofday(&endtime, 0);
            timersub(&endtime, &starttime, &difftime);
            double diff = difftime.tv_sec + (difftime.tv_usec/1000000.0);
            cerr ≪ "fps = " ≪ (10.0/diff) ≪ endl;
        }
        gettimeofday(&starttime, 0);
    }
#endif
    framecounter++;
}

/* This function does connected component analysis on the boxes,
   in order to find the ridge.
*/
Component Boxes::findRidge(image im)
{
    unsigned int compcnt=0;
    for(int h = boxnumberheight-1; h ≥ 0; h--){
        unsigned int last_found_component = 0;
        int last_found_position = 0;
        for(int w = boxnumberwidth-1; w ≥ 0; w--){
            if(boxes[h][w].count > boxthress+h){
                unsigned int foundcomponent = 0;
                bool leftbranch = false;
                bool added = false;

                if ((unsigned int)h+1 < boxnumberheight &&
                    (unsigned int)w+1 < boxnumberwidth &&
                    boxes[h+1][w+1].compnumber &&
                    (boxes[h+1][w+1].canConnect(boxes[h][w]) ||
                        connected[boxes[h+1][w+1].compnumber].canConnect(boxes[h][w]))) {
                    if((last_found_component ≠ boxes[h+1][w+1].compnumber &&
boxes[h+1][w+1].compnumber ≠ connected[last_found_component].compnumber)
                        || last_found_position ≤ w+1){
                        foundcomponent = boxes[h+1][w+1].compnumber;
                        boxes[h][w].compnumber = foundcomponent;
                        connected[foundcomponent].addBox(boxes[h][w], h);
                        last_found_component = foundcomponent;
                        last_found_position = w;
                        added = true;
                    }else{
```

```
                    leftbranch = true;
                }
            }

            if(!leftbranch){
                if((unsigned int)h+1 < boxnumberheight &&
                    boxes[h+1][w].compnumber &&
                    (boxes[h+1][w].canConnect(boxes[h][w]) ||
                        connected[boxes[h+1][w].compnumber].canConnect(boxes[h][w]))) {
                    if((last_found_component ≠ boxes[h+1][w].compnumber &&
boxes[h+1][w].compnumber ≠ connected[last_found_component].compnumber)
                            || last_found_position ≤ w+1){
                        if(foundcomponent && foundcomponent ≠ boxes[h+1][w].compnumber){
                            if(foundcomponent < boxes[h+1][w].compnumber){
                                connected[boxes[h+1][w].compnumber].compnumber =
foundcomponent;
                            }else{
                                connected[foundcomponent].compnumber =
boxes[h+1][w].compnumber;
                                foundcomponent = boxes[h+1][w].compnumber;
                            }
                        }else{
                            foundcomponent = boxes[h+1][w].compnumber;
                        }

                        if(!added){
                            boxes[h][w].compnumber = foundcomponent;
                            connected[foundcomponent].addBox(boxes[h][w], h);

                            last_found_component = foundcomponent;
                            last_found_position = w;
                            added = true;
                        }
                    }else{
                        leftbranch = true;
                    }
                }
            }

            if(!leftbranch){
                if ((unsigned int)h+1 < boxnumberheight &&
                    (unsigned int)w-1 < boxnumberwidth &&
                    boxes[h+1][w-1].compnumber &&
                    (boxes[h+1][w-1].canConnect(boxes[h][w]) ||
                        connected[boxes[h+1][w-1].compnumber].canConnect(boxes[h][w]))) {

                    if(last_found_component ≠ boxes[h+1][w-1].compnumber &&
boxes[h+1][w-1].compnumber ≠ connected[last_found_component].compnumber
                            || last_found_position ≤ w+1){
                        if(foundcomponent && foundcomponent ≠ boxes[h+1][w-1].compnumber){
                            if(foundcomponent < boxes[h+1][w-1].compnumber){
                                connected[boxes[h+1][w-1].compnumber].compnumber =
foundcomponent;
                            }else{
                                connected[foundcomponent].compnumber =
boxes[h+1][w-1].compnumber;
                                foundcomponent = boxes[h+1][w-1].compnumber;
                            }
                        }else{
                            foundcomponent = boxes[h+1][w-1].compnumber;
                        }

                        if(!added){
                            boxes[h][w].compnumber = foundcomponent;
                            connected[foundcomponent].addBox(boxes[h][w], h);

                            last_found_component = foundcomponent;
```

```
                        last_found_position = w;
                        added = true;
                    }
                }else{
                    leftbranch = true;
                }
            }
        }

        if(!leftbranch){
            if ((unsigned int)w+1 < boxnumberwidth &&
                boxes[h][w+1].compnumber &&
                (boxes[h][w+1].canConnect(boxes[h][w]) ||
                    connected[boxes[h][w+1].compnumber].canConnect(boxes[h][w]))) {

                if(foundcomponent && foundcomponent ≠ boxes[h][w+1].compnumber){
                    if(foundcomponent < boxes[h][w+1].compnumber){
                        connected[boxes[h][w+1].compnumber].compnumber = foundcomponent;
                    }else{
                        connected[foundcomponent].compnumber = boxes[h][w+1].compnumber;
                        foundcomponent = boxes[h][w+1].compnumber;
                    }
                }else{
                    foundcomponent = boxes[h][w+1].compnumber;
                }

                if(!added){
                    boxes[h][w].compnumber = foundcomponent;
                    connected[foundcomponent].addBox(boxes[h][w], h);

                    last_found_component = foundcomponent;
                    last_found_position = w;
                    added = true;
                }
            }
        }

        if(!foundcomponent){
            boxes[h][w].compnumber = ++compcnt;
            connected[boxes[h][w].compnumber].compnumber = compcnt;
            connected[boxes[h][w].compnumber].leftbranch = leftbranch;
            connected[boxes[h][w].compnumber].addBox(boxes[h][w], h);
        }
    }
  }
}

unsigned int curmaxweight=0;
unsigned int curmaxcomp=0;

for(unsigned int i = 1; i ≤ compcnt; i++) {
    if(!connected[i].leftbranch){
        if(connected[i].compnumber ≠ i && !connected[connected[i].compnumber].leftbranch){
            connected[i].addComponent(connected[connected[i].compnumber]);
        }

        if(connected[i].weight > curmaxweight) {
            curmaxweight = connected[i].weight;
            curmaxcomp = i;
        }
    }
}

if(curmaxweight < weight_min_component){
    cerr ≪ "component of weight " ≪ curmaxweight ≪ " not selected." ≪ endl;
    curmaxcomp = 0;
    curmaxweight = 0;
```

```
    }

    if(curmaxcomp){
        connected[curmaxcomp].finishComponent();
    }

#ifdef WRITEPIC
    if(curmaxcomp){
        for(unsigned int h = 0; h < boxnumberheight; h++){
            for(unsigned int w = 0; w < boxnumberwidth; w++){
                if(boxes[h][w].compnumber == curmaxcomp){
                    for(unsigned int i = w*boxwidth+edgesize; i < (w+1)*boxwidth+edgesize; i++){
                        im[h*boxheight+edgesize][i] = 128;
                        im[(h+1)*boxheight+edgesize][i] = 128;
                    }
                    for(unsigned int i = h*boxheight+edgesize; i < (h+1)*boxheight+edgesize; i++){
                        im[i][w*boxwidth+edgesize] = 128;
                        im[i][(w+1)*boxwidth+edgesize] = 128;
                    }
                }
            }
        }
    }
#endif


    /*
      cout << "maxw=" << curmaxweight
      << ", maxc=" << curmaxcomp
      << ", COM=[" << connected[curmaxcomp].center_h
      << "][" << connected[curmaxcomp].center_w << "]" << endl;

    for(unsigned int h = 0; h < boxnumberheight; h++){
        for(unsigned int w = 0; w < boxnumberwidth; w++){
            cout << "[" << boxes[h][w].compnumber << ","
            << boxes[h][w].count << "] ";
        }
        cout << endl;
    }
    */

    return connected[curmaxcomp];
}

/* This function finds bright ridge points in the Gauss smoothed image.
 */
Component Robot::findRidgePoints(image im, image outim)
{
    boxes.clear();
    for(unsigned int h = edgesize; h < height-edgesize; h++){
        for(unsigned int w = edgesize; w < width-edgesize; w++){
            int double_val = 2*im[h][w];
            int Lxx = im[h][w-2] + im[h][w+2] - double_val;
            int Lyy = im[h-2][w] + im[h+2][w] - double_val;
            int Lxy = im[h-1][w-1] + im[h+1][w+1] - im[h-1][w+1] - im[h+1][w-1];

            int Lxx_Lyy = Lxx-Lyy;

#ifdef WRITEPIC
            outim[h][w] = im[h][w];
#endif
            int bar = sqrt_int(((Lxx_Lyy*Lxx_Lyy)+4*Lxy*Lxy)*256);
            if(!bar){
                // we can't devide by zero, so just continue with the next pixel
                continue;
            }
```

```
                int int_foo = (Lxx_Lyy*256*16 / bar);
                // intcosB and intsinB is 256 times larger than their floating point counterparts
                int intcosB = sqrt_int((256 + int_foo)*128);
                int intsinB = sqrt_int((256 - int_foo)*128);
                if(Lxy < 0){
                    intsinB = -intsinB;
                }

                int Lx = im[h][w+1] - im[h][w-1];
                int Ly = im[h+1][w] - im[h-1][w];

                // intLp and intLq is 256 times larger than Lp and Lq
                int intLp = intsinB * Lx - intcosB * Ly;
                //int intLq = intcosB * Lx + intsinB * Ly;

                // intLpp and intLqq is 256*256 times larger than Lpp and Lqq
                int intLpp = intsinB*intsinB * Lxx - 2*intcosB*intsinB * Lxy + intcosB*intcosB * Lyy;
                int intLqq = intcosB*intcosB * Lxx + 2*intcosB*intsinB * Lxy + intsinB*intsinB * Lyy;

                /*
                  cout << "[" << h << "][" << w << "] Lp=" << intLp
                  << ", Lpp=" << intLpp/256 << ", Lq=" << intLq
                  << ", Lqq=" << intLqq/256 << ",Lpp-Lqq=" << (abs(intLpp)-abs(intLqq))/256
                  << "(c1)(" << c1 << "," << c2 << ")" << endl;
                */

                //To see if it is zero, we find a treshhold*256
                if(abs(intLpp) ≥ abs(intLqq*2)){
                    if(intLp < 512 && intLp > -512 &&
                        intLpp ≤ -2*256*256 &&
                        im[h][w] > 150) {
                        boxes.addPoint(h, w, intcosB, intsinB);
#ifdef WRITEPIC
                        outim[h][w] = 0;
#endif
                    }
                }
                /* Black ridges are not needed to begin with
                  else if(abs(intLqq) >= abs(intLpp*2)){
                    if(intLq < 512 && intLq > -512 &&
                        intLqq >= 2*256*256 &&
                        im[h][w] < 120) {
                        outim[h][w] = 255;
                    }
                }
                */

            }
        }
        Component com = boxes.findRidge(outim);

#ifdef WRITEPIC
        if(com.center_h || com.center_w){
            if(com.center_h-1 > 0 && com.center_h+1 < (int)(height-edgesize) &&
                com.center_w-1 > 0 && com.center_w+1 < (int)(width-edgesize)){
                outim[edgesize+com.center_h][edgesize+com.center_w] = 255;
                outim[edgesize+com.center_h+1][edgesize+com.center_w+1] = 0;
                outim[edgesize+com.center_h+1][edgesize+com.center_w] = 0;
                outim[edgesize+com.center_h+1][edgesize+com.center_w-1] = 0;
                outim[edgesize+com.center_h][edgesize+com.center_w+1] = 0;
                outim[edgesize+com.center_h][edgesize+com.center_w-1] = 0;
                outim[edgesize+com.center_h-1][edgesize+com.center_w+1] = 0;
                outim[edgesize+com.center_h-1][edgesize+com.center_w] = 0;
                outim[edgesize+com.center_h-1][edgesize+com.center_w-1] = 0;
            }
        }
#endif
```

```
    return com;
}

/* Normalizes the image, before doing the Gauss smoothing.
 */
void normalize(image im)
{
    int min = 255;
    int max = 0;

    for(unsigned int h = 0; h < height; h++){
        for(unsigned int w = 0; w < width; w++){
            if(im[h][w] < min) min = im[h][w];
            else if(im[h][w] > max) max = im[h][w];
        }
    }

    unsigned int factor = 255*256 / (max - min);

    for(unsigned int h = 0; h < height; h++){
        for(unsigned int w = 0; w < width; w++){
            im[h][w] = ((im[h][w] - min) * factor)/256;
        }
    }
}

/*Processes the newly take picture, and determines how
  the robot should react to it.
 */
ComRCX::command Robot::processImage(bool write)
{
#ifdef WRITEPIC
    if(write){
        writePPM((char *)image1);
    }
#endif
    normalize(image1);

#ifdef WRITEPIC
    if(write){
        writePPM((char *)image1);
    }
#endif

    gaussProcess(image1);

#ifdef WRITEPIC
    if(write){
        writePPM((char *)image1);
    }
#endif

    Component c = findRidgePoints(image1, image2);

#ifdef WRITEPIC
    if(write){
        writePPM((char *)image2);
    }
#endif
    if(backcounter > backthress){
        noback = noback_for;
    }

    if(c.center_w == 0 && c.center_h == 0 &&
        framecounter > frames_before_backward &&
        noback == 0){
```

```
        backcounter += 2;
#ifdef NOROBOT
        cout << "backward" << endl;
#endif
        return ComRCX::backward;
    }

    if(backcounter > 0){
        backcounter--;
    }

    if(noback > 0){
        noback--;
    }

    int borderwidth = (innerwidth)/3;
    if(c.center_w > (int)innerwidth-borderwidth){
#ifdef NOROBOT
        cout << "right" << endl;
#endif
        return ComRCX::turnright;
    }else if(c.center_w && c.center_w < borderwidth){
#ifdef NOROBOT
        cout << "left" << endl;
#endif
        return ComRCX::turnleft;
    }

#ifdef NOROBOT
    cout << "forward" << endl;
#endif
    return ComRCX::forward;
}


int main(int argc, char *argv[])
{
#ifndef NOROBOT
    char *device = "/dev/video";

    if(system("/root/vctrl 204x152x8") != 0){
        cerr << "Unable to run vctrl." << endl;
    }

    unlink("/tmp/recieveBot");
    unlink("/tmp/bumper");

    vid = open(device, O_RDONLY);
    if (vid < 0) {
        fprintf(stderr, "Can't open %s\n", device);
        return 1;
    }

    struct v4l2_capability cap;
    int err = ioctl(vid, VIDIOC_QUERYCAP, &cap);
    if (err) {
        fprintf(stderr, "QUERYCAP returned error %d\n", errno);
        return 1;
    }
    if (cap.type != V4L2_TYPE_CAPTURE) {
        fprintf(stderr, "Device %s is not a video capture device.\n", device);
        return 1;
    }
    if (!(cap.flags & V4L2_FLAG_READ)) {
        fprintf(stderr, "Device %s doesn't support read().\n", device);
        return 1;
    }
```

```
    fmt.type = V4L2_BUF_TYPE_CAPTURE;
    err = ioctl(vid, VIDIOC_G_FMT, &fmt);
    if (err) {
        fprintf(stderr, "G_FMT returned error %d\n", errno);
        return 1;
    }

    if(width ≠ fmt.fmt.pix.width ||
        height ≠ fmt.fmt.pix.height){
        fprintf(stderr, "height or width failed\n");
        return 1;
    }

    ComRCX rcx;
#else
    strcpy(filename, argv[1]);
#endif
    Robot robot;
    unsigned int i = 0;
#ifndef NOROBOT
    for(i = 0; true; i++){
#endif
        robot.takePicture();

        ComRCX::command c = robot.processImage((i%6)==0);
#ifndef NOROBOT
        rcx.sendRCX(c);
    }
#endif

    return 0;
}
```

C.3 sendRCX.h

```
/*
  ComRCX class

  This class is used for communication from the iPAQ to the RCX unit.
  The method sendRCX can be called with a command parameter which is an
  command there can be sent to the RCX.
*/

class ComRCX{
public:
    enum command
        { stop,
          forward,
          backward,
          turnleft,
          turnright
              };
    ComRCX();
    ~ComRCX();
    void sendRCX(command c);
private:
    enum packet_state
        { P_BEGIN,
          P_MESSAGE,
          P_STATUS,
          P_DATA,
          P_N,
          P_DATA_N,
          P_END
        };

    const char *read_string(int fd);
    int send_packet(int fd,const char *packet);

    int fd;
    int lastcode;
    int bumper;
};
```

C.4 sendRCX.cc

```
//This file is based on the source for the rc program in LIRC.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <getopt.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/un.h>
#include <errno.h>
#include <signal.h>
#include <limits.h>
#include <iostream>
#include <fstream>
#include "sendRCX.h"

#define PACKET_SIZE 256
/* three seconds */
#define TIMEOUT 3

char *progname = "robot";

int timeout = 0;

void sigalrm(int sig)
{
    timeout=1;
}

const char* ComRCX::read_string(int fd)
{
    static char buffer[PACKET_SIZE+1]="";
    char *end;
    static int ptr=0;                                          //,end_len=0;
    ssize_t ret;

    if(ptr>0)
    {
        memmove(buffer,buffer+ptr,strlen(buffer+ptr)+1);
        ptr=strlen(buffer);
        end=strchr(buffer,'\n');
    }
    else
    {
        end=NULL;
    }
    alarm(TIMEOUT);
    while(end==NULL)
    {
        if(PACKET_SIZE<=ptr)
        {
            fprintf(stderr,"%s:  bad packet\n",progname);
            ptr=0;
            return(NULL);
        }
        ret=read(fd,buffer+ptr,PACKET_SIZE-ptr);

        if(ret<=0 || timeout)
        {
            if(timeout)
            {
                fprintf(stderr,"%s:  timeout\n",progname);
```

```
            }
            else
            {
                alarm(0);
            }
            ptr=0;
            return(NULL);
        }
        buffer[ptr+ret]=0;
        ptr=strlen(buffer);
        end=strchr(buffer,'\n');
    }
    alarm(0);timeout=0;

    end[0]=0;
    ptr=strlen(buffer)+1;
#ifdef DEBUG
    printf("buffer:  -%s-\n",buffer);
#endif
    return(buffer);
}

int ComRCX::send_packet(int fd,const char *packet)
{
    int done,todo;
    const char *string,*data;
    char *endptr;
    enum packet_state state;
    int status,n;
    unsigned long data_n = 0;

    todo=strlen(packet);
    data=packet;
    while(todo>0)
    {
        done=write(fd,(void *) data,todo);
        if(done<0)
        {
            fprintf(stderr,"%s:  could not send packet\n",
                progname);
            perror(progname);
            return(-1);
        }
        data+=done;
        todo-=done;
    }

    /* get response */
    status=0;
    state=P_BEGIN;
    n=0;
    while(1)
    {
        string=read_string(fd);
        if(string==NULL) return(-1);
        switch(state)
        {
        case P_BEGIN:
            if(strcasecmp(string,"BEGIN")≠0)
            {
                continue;
            }
            state=P_MESSAGE;
            break;
        case P_MESSAGE:
            if(strncasecmp(string,packet,strlen(string))≠0 ||
               strlen(string)+1≠strlen(packet))
```

```
        {
            state=P_BEGIN;
            continue;
        }
        state=P_STATUS;
        break;
    case P_STATUS:
        if(strcasecmp(string,"SUCCESS")==0)
        {
            status=0;
        }
        else if(strcasecmp(string,"END")==0)
        {
            status=0;
            return(status);
        }
        else if(strcasecmp(string,"ERROR")==0)
        {
            fprintf(stderr,"%s:  command failed:  %s",
                progname,packet);
            status=-1;
        }
        else
        {
            goto bad_packet;
        }
        state=P_DATA;
        break;
    case P_DATA:
        if(strcasecmp(string,"END")==0)
        {
            return(status);
        }
        else if(strcasecmp(string,"DATA")==0)
        {
            state=P_N;
            break;
        }
        goto bad_packet;
    case P_N:
        errno=0;
        data_n=strtoul(string,&endptr,0);
        if(!*string || *endptr)
        {
            goto bad_packet;
        }
        if(data_n==0)
        {
            state=P_END;
        }
        else
        {
            state=P_DATA_N;
        }
        break;
    case P_DATA_N:
        fprintf(stderr,"%s:  %s\n",progname,string);
        n++;
        if((unsigned int)n==data_n) state=P_END;
        break;
    case P_END:
        if(strcasecmp(string,"END")==0)
        {
            return(status);
        }
        goto bad_packet;
        break;
```

```
            }
        }
    bad_packet:
        fprintf(stderr,"%s:  bad return packet\n",progname);
        return(-1);
}

void ComRCX::sendRCX(command c)
{
        char *directive = "send_once";
        char *remote = "rcx";
        char code[10];
        char buffer[PACKET_SIZE+1];

        switch(c){
            case stop:
                strcpy(code, "1");
                break;
            case forward:
                strcpy(code, "2");
                break;
            case backward:
                strcpy(code, "3");
                break;
            case turnright:
                strcpy(code, "4");
                break;
            case turnleft:
                strcpy(code, "5");
                break;
        }

        int recievecode = 0;
        ifstream inputstr("/tmp/recieveBot");
        if(inputstr){
            inputstr >> recievecode;
        }

        int intcode = atoi(code);

        if(recievecode == intcode && intcode==lastcode){
            // we already have this command.. Do nothing!
            //cout << "not sending " << code << endl;
            return;
        }

        if(fopen("/tmp/bumper", "r")){
            cout << "robot:  Bumper touched" << endl;
            if(bumper == 0){
                bumper = 5;
            }
            unlink("/tmp/bumper");
        } else {
            // no bumper!
        }

        if(bumper){
            bumper--;
            strcpy(code, "3");
        }

        cout << "sending " << code << endl;

        sprintf(buffer,"%s %s %s\n",directive,remote,code);

        if(send_packet(fd,buffer)==-1) {
            exit(EXIT_FAILURE);
```

```
    }
    lastcode = intcode;
}

ComRCX::ComRCX()
    : lastcode(0),
      bumper(0)
{
    struct sockaddr_un addr;
    struct sigaction act;

    act.sa_handler=sigalrm;
    sigemptyset(&act.sa_mask);
    act.sa_flags=0;                                          /* we need EINTR */
    sigaction(SIGALRM,&act,NULL);

    addr.sun_family=AF_UNIX;
    strcpy(addr.sun_path,"/dev/lircd");
    fd=socket(AF_UNIX,SOCK_STREAM,0);
    if(fd==-1)
    {
        fprintf(stderr,"%s:  could not open socket\n",progname);
        perror(progname);
        exit(EXIT_FAILURE);
    };
    if(connect(fd,(struct sockaddr *)&addr,sizeof(addr))==-1)
    {
        fprintf(stderr,"%s:  could not connect to socket\n",progname);
        perror(progname);
        exit(EXIT_FAILURE);
    };
}

ComRCX::~ComRCX()
{
    close(fd);
}
```

C.5 int_math.h

```
/*
  This function have the ability to caltulate square root in int.
  found at: http://www.finesse.demon.co.uk/steven/sqrt.html
*/

#define iter1(N) \
roottry = root + (1 << (N)); \
if (n >= roottry << (N)) \
{ n -= roottry << (N); \
root |= 2 << (N); \
}

inline unsigned int sqrt_int (unsigned int n)
{
    unsigned int root = 0;
    unsigned int roottry;
    iter1 (15); iter1 (14); iter1 (13); iter1 (12);
    iter1 (11); iter1 (10); iter1 ( 9); iter1 ( 8);
    iter1 ( 7); iter1 ( 6); iter1 ( 5); iter1 ( 4);
    iter1 ( 3); iter1 ( 2); iter1 ( 1); iter1 ( 0);
    return root >> 1;
}
```

C.6 gauss.cc

```
#include <math.h>
#include <iostream>

/* This program is used to generated the gauss.h file, in order to get faster code.
 */
int main()
{
    int size = 43;
    int spread = 7;

    int halfgsize = size/2;
    int scale = 10000000;
    cout << "// File for use in Gaussian filtering, automaticly generated by gauss.cc" << endl;
    cout << "#define GAUSS_SIZE " << size << endl;
    cout << "#define HALFGAUSS_SIZE " << halfgsize << endl << endl;
    cout << "void gaussProcess (image im)" << endl;
    cout << "{" << endl;
    cout << "\tstatic image tmpim;" << endl;

    for(int j = 0; j < 2; j++){
        int sum = 0;
        if(j){
            cout << "\tfor(unsigned int h = HALFGAUSS_SIZE; h < height-HALFGAUSS_SIZE; h++){"
<< endl;
            cout << "\t\tfor(unsigned int w = HALFGAUSS_SIZE; w < width-HALFGAUSS_SIZE; w++){"
<< endl;
            cout << "\t\t\tim[h][w] = (";
        }else{
            cout << "\tfor(unsigned int h = 0; h < height; h++){" << endl;
            cout << "\t\tfor(unsigned int w = HALFGAUSS_SIZE; w < width-HALFGAUSS_SIZE; w++){"
<< endl;
            cout << "\t\t\ttmpim[h][w] = (";
        }
        bool firsttime = true;
        for(int i = -size/2; i ≤ size/2; i++){
            int value =
(int)rint(((1/(2*3.14159265359*pow(spread,2)))*(pow(2.7182818,(-(pow(i,2))/(2*pow(spread,2)))))*scale));
            sum += value;

            if(value){
                if(!firsttime){
                    cout << " + " << endl << "\t\t\t\t";
                }
                firsttime = false;
                if(value == 1){
                    if(j){
                        cout << "tmpim[h";
                    }else{
                        cout << "im[h";
                    }
                }else{
                    if(j){
                        cout << value << "*tmpim[h";
                    }else{
                        cout << value << "*im[h";
                    }
                }
                if(j){
                    if(i < 0){
                        cout << i;
                    }else{
                        cout << "+" << i;
                    }
                }
                cout << "][w";
```

```
                    if(!j){
                        if(i < 0){
                            cout << i;
                        }else{
                            cout << "+" << i;
                        }
                    }
                    cout << "]";
                }
            }
            cout << ")/" << sum << ";" << endl;
            cout << "\t\t}" << endl;
            cout << "\t}" << endl;
        }
        cout << "}" << endl;
    }
```

C.7 gauss.h

This is the output from `gauss.cc`.

```
// File for use in Gaussian filtering, automaticly generated by gauss.cc
#define GAUSS_SIZE 43
#define HALFGAUSS_SIZE 21

void gaussProcess (image im)
{
    static image tmpim;
    for(unsigned int h = 0; h < height; h++){
        for(unsigned int w = HALFGAUSS_SIZE; w < width-HALFGAUSS_SIZE; w++){
            tmpim[h][w] = (361*im[h][w-21] +
                548*im[h][w-20] +
                816*im[h][w-19] +
                1191*im[h][w-18] +
                1702*im[h][w-17] +
                2383*im[h][w-16] +
                3270*im[h][w-15] +
                4396*im[h][w-14] +
                5790*im[h][w-13] +
                7473*im[h][w-12] +
                9449*im[h][w-11] +
                11708*im[h][w-10] +
                14212*im[h][w-9] +
                16905*im[h][w-8] +
                19700*im[h][w-7] +
                22495*im[h][w-6] +
                25167*im[h][w-5] +
                27588*im[h][w-4] +
                29631*im[h][w-3] +
                31182*im[h][w-2] +
                32151*im[h][w-1] +
                32481*im[h][w+0] +
                32151*im[h][w+1] +
                31182*im[h][w+2] +
                29631*im[h][w+3] +
                27588*im[h][w+4] +
                25167*im[h][w+5] +
                22495*im[h][w+6] +
                19700*im[h][w+7] +
                16905*im[h][w+8] +
                14212*im[h][w+9] +
                11708*im[h][w+10] +
                9449*im[h][w+11] +
                7473*im[h][w+12] +
                5790*im[h][w+13] +
                4396*im[h][w+14] +
                3270*im[h][w+15] +
                2383*im[h][w+16] +
                1702*im[h][w+17] +
                1191*im[h][w+18] +
                816*im[h][w+19] +
                548*im[h][w+20] +
                361*im[h][w+21])/568717;
        }
    }
    for(unsigned int h = HALFGAUSS_SIZE; h < height-HALFGAUSS_SIZE; h++){
        for(unsigned int w = HALFGAUSS_SIZE; w < width-HALFGAUSS_SIZE; w++){
            im[h][w] = (361*tmpim[h-21][w] +
                548*tmpim[h-20][w] +
                816*tmpim[h-19][w] +
                1191*tmpim[h-18][w] +
                1702*tmpim[h-17][w] +
                2383*tmpim[h-16][w] +
                3270*tmpim[h-15][w] +
```

```
                    4396*tmpim[h-14][w] +
                    5790*tmpim[h-13][w] +
                    7473*tmpim[h-12][w] +
                    9449*tmpim[h-11][w] +
                    11708*tmpim[h-10][w] +
                    14212*tmpim[h-9][w] +
                    16905*tmpim[h-8][w] +
                    19700*tmpim[h-7][w] +
                    22495*tmpim[h-6][w] +
                    25167*tmpim[h-5][w] +
                    27588*tmpim[h-4][w] +
                    29631*tmpim[h-3][w] +
                    31182*tmpim[h-2][w] +
                    32151*tmpim[h-1][w] +
                    32481*tmpim[h+0][w] +
                    32151*tmpim[h+1][w] +
                    31182*tmpim[h+2][w] +
                    29631*tmpim[h+3][w] +
                    27588*tmpim[h+4][w] +
                    25167*tmpim[h+5][w] +
                    22495*tmpim[h+6][w] +
                    19700*tmpim[h+7][w] +
                    16905*tmpim[h+8][w] +
                    14212*tmpim[h+9][w] +
                    11708*tmpim[h+10][w] +
                    9449*tmpim[h+11][w] +
                    7473*tmpim[h+12][w] +
                    5790*tmpim[h+13][w] +
                    4396*tmpim[h+14][w] +
                    3270*tmpim[h+15][w] +
                    2383*tmpim[h+16][w] +
                    1702*tmpim[h+17][w] +
                    1191*tmpim[h+18][w] +
                    816*tmpim[h+19][w] +
                    548*tmpim[h+20][w] +
                    361*tmpim[h+21][w])/568717;
            }
        }
    }
```

C.8 Makefile

```
CC=arm-linux-gcc
CXX=arm-linux-g++
CFLAGS=-Wall -Xlinker -dy -Xlinker -R/skiff/local/arm-linux/lib/X11
CXXFLAGS=-Wall -O3 -funroll-loops -finline-functions \
-Xlinker --strip-all -Xlinker -dy -Xlinker \
-R/skiff/local/arm-linux/lib/X11 -ffast-math

#CXXFLAGS=-Wall -O3 -funroll-loops -finline-functions \
# -ggdb -Xlinker -dy -Xlinker -R/skiff/local/arm-linux/lib/X11
# -ffast-math
#-I/usr/include
#LDLIBS= -L/skiff/local/arm-linux/lib/X11 -lXaw -lXt -lX11
#LDLIBS=-L/usr/X11R6/lib -lXt -lX11
#LDLIBS=
all: robot

robot: sendRCX.h sendRCX.cc robot.cc gauss.h gauss.cc robot.h int_math.h
	$(CXX) $(CXXFLAGS) sendRCX.cc robot.cc -o robot

gauss.h: gauss.cc
	g++ gauss.cc -o gauss
	./gauss > gauss.h

vr: robot.cc sendRCX.h gauss.h robot.h int_math.h gauss.cc
	g++ -O3 -DNOROBOT robot.cc -o vr
```

D convertnumbers.pl

```
#!/usr/bin/perl
use POSIX;

$i = 0;
$button = 2;
while(<>){
    $num = strtod($_);
    $numm = floor(($num/417)+0.5)*417;
    if($numm > 10000){
        print "\n\n\tname $button\n\t\t";
        $button++;
        $i = 0;
    }else{
        print "$numm\t";
    }

    if($i == 6){
        print "\n\t\t";
        $i = 0;
    }
    $i++;
}

print "\n";
```

E IRexec

```
begin
        prog    = irexec
        remote  = rcx
        button  = 1
        repeat  = 0
        config  = echo "1" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 2
        repeat  = 0
        config  = echo "2" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 3
        repeat  = 0
        config  = echo "3" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 4
        repeat  = 0
        config  = echo "4" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 5
        repeat  = 0
        config  = echo "5" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 6
        repeat  = 0
        config  = touch /tmp/bumper && echo "bumper" > /dev/tty0
end

begin
        prog    = irexec
        remote  = rcx
        button  = 7
        repeat  = 0
        config  = echo "7" > /tmp/recieveBot && echo "Left bumber" > /dev/tty0
end

begin
        prog    = irexec
        remote  = rcx
        button  = 8
        repeat  = 0
#        config  = echo "8" > /tmp/recieveBot
end

begin
```

```
        prog    = irexec
        remote  = rcx
        button  = 9
        repeat  = 0
#        config  = echo "9" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 10
        repeat  = 0
#        config  = echo "10" > /tmp/recieveBot
end

begin
        prog    = irexec
        remote  = rcx
        button  = 11
        repeat  = 0
        config  = echo "modtog 11"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 12
        repeat  = 0
        config  = echo "modtog 12"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 13
        repeat  = 0
        config  = echo "modtog 13"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 14
        repeat  = 0
        config  = echo "modtog 14"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 15
        repeat  = 0
        config  = echo "modtog 15"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 16
        repeat  = 0
        config  = echo "modtog 16"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 17
```

```
        repeat  = 0
        config  = echo "modtog 17"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 18
        repeat  = 0
        config  = echo "modtog 18"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 19
        repeat  = 0
        config  = echo "modtog 19"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 20
        repeat  = 0
        config  = echo "modtog 20"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 21
        repeat  = 0
        config  = echo "modtog 21"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 22
        repeat  = 0
        config  = echo "modtog 22"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 23
        repeat  = 0
        config  = echo "modtog 23"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 24
        repeat  = 0
        config  = echo "modtog 24"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 25
        repeat  = 0
        config  = echo "modtog 25"
end
```

```
begin
        prog    = irexec
        remote  = rcx
        button  = 26
        repeat  = 0
        config  = echo "modtog 26"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 27
        repeat  = 0
        config  = echo "modtog 27"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 28
        repeat  = 0
        config  = echo "modtog 28"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 29
        repeat  = 0
        config  = echo "modtog 29"
end

begin
        prog    = irexec
        remote  = rcx
        button  = 30
        repeat  = 0
        config  = echo "modtog 30"
end
```

F LIRC

```
# Copyright (C) 1999 Christoph Bartelmus
#
# You may only use this file if you make it available to others,
# i.e. if you send it to <lirc@bartelmus.de>
#
# this config file was automatically generated
# using lirc-0.6.5(ipaq) on Sun Apr  7 17:59:44 2002
#
# contributed by
#
# brand:                       newtest
# model no. of remote control:
# devices being controlled by this remote:
#

begin remote

  name  rcx
  flags RAW_CODES
  eps            30
  aeps          100

  ptrail          0
  repeat      0     0

      begin raw_codes

        name 1
      417   417   417   417   417   417
      417   417   417   834   417   4170
      3753  834   417   1251  417   1668
      417   417   1668  417   2085  417
      417   417   3336  417   834   2919
      417   417   1668  2085  417   417
      417   1251  2502

   name 2
      417   417   417   417   417   417
      417   417   417   834   417   4170
      3753  834   417   1251  417   1668
      417   417   1668  417   2085  417
      834   417   2919  417   417   417
      417   2502  417   417   417   417
      834   2919  834   834   2085

   name 3
      417   417   417   417   417   417
      417   417   417   834   417   4170
      3753  834   417   1251  417   1668
      417   417   1668  417   2085  417
      417   834   2502  834   1251  3336
      834   417   417   2919  417   417
      417   417   2085

   name 4
      417   417   417   417   417   417
      417   417   417   834   417   4170
      3753  834   417   1251  417   1668
      417   417   1668  417   2085  417
      1251  417   2502  417   417   834
      417   2085  417   417   417   834
      417   2085  417   417   1251  417
      2502

   name 5
```

```
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    417   417   417   417   2085  834
    834   417   417   2919  1251  3336
    417   834   2502

name 6
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    834   834   2085  834   417   417
    834   2919  417   417   417   2502
    417   417   834   417   2919

name 7
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    417   1251  2502  417   1668  2085
    417   417   834   2919  417   417
    417   417   3336

name 8
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    1668  417   2085  417   417   1251
    417   1668  417   417   417   4170
    3753

name 9
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    417   417   834   417   1668  834
    834   834   417   2502  3753  834
    417

name 10
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    834   417   417   417   1668  834
    417   417   417   417   417   2502
    417   417   3336  417   834   2919
    417

name 11
    417   417   417   417   417   417
    417   417   417   834   417   4170
    3753  834   417   1251  417   1668
    417   417   1668  417   2085  417
    417   834   417   417   2085  417
    1251  417   417   1668  417   417
    834   417   2919  417   417   417
    417   2502  417

name 12
    417   417   417   417   417   417
    417   417   417   834   417   4170
```

```
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    1251   834    1668   834    417    834
    834    2502   417    834    2502   834
    1251

name 13
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    417    417    834    2085   417
    834    417    834    1668   417    417
    1251   417    2502   417    417    834
    417    2085   417

name 14
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    834    1251   2085   417    417    417
    1251   1668   417    417    417    417
    417    417    2085   834    834    417
    417

name 15
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    1668   1668   834    2085   2502
    834    834    2085   834    417    417
    834

name 16
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    2085   417    1668   417    417    1668
    417    1251   417    417    417    1251
    2502   417    1668   2085   417

name 17
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    417    1251   417    1251   834
    834    1251   417    2085   1668   417
    2085   417    417    1251   417    1668
    417

name 18
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    834    417    834    417    1251   834
    417    417    417    834    417    2085
    417    417    834    417    1668   834
    834    834    417

name 19
    417    417    417    417    417    417
    417    417    417    834    417    4170
```

```
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    834    834    417    1668   417
    1251   834    417    1251   417    417
    834    417    417    417    1668   834
    417    417    417    417    417

name 20
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    1251   417    417    417    1251   834
    417    834    417    417    417    2085
    417    834    417    417    2085   417
    1251   417    417    1668   417

name 21
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    417    417    417    417    417
    1668   417    834    417    417    417
    417    1251   417    417    1251   834
    1668   834    417    834    834

name 22
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    834    834    417    417    1668   417
    417    417    834    417    417    1251
    417    417    417    417    417    834
    2085   417    834    417    834    1668
    417

name 23
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    1251   417    417    1251   834
    1668   417    417    2085   834    1251
    2085   417    417    417    1251   1668
    417

name 24
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    1668   834    1251   834    417    1251
    834    2085   417    1668   1668   834
    2085

name 25
    417    417    417    417    417    417
    417    417    417    834    417    4170
    3753   834    417    1251   417    1668
    417    417    1668   417    2085   417
    417    417    834    834    1668   417
    834    834    834    1251   417    417
    2085   417    1668   417    417    1668
    417    1251   417
```

```
    name 26
        417   417   417   417   417   417
        417   417   417   834   417   4170
        3753  834   417   1251  417   1668
        417   417   1668  417   2085  417
        834   417   417   834   1668  417
        417   417   417   417   834   1251
        417   417   417   417   1251  417
        1251  834   834   1251  417

    name 27
        417   417   417   417   417   417
        417   417   417   834   417   4170
        3753  834   417   1251  417   1668
        417   417   1668  417   2085  417
        417   834   417   834   1251  834
        1251  417   834   2085  834   417
        834   417   1251  834   417   417
        417   834   417

    name 28
        417   417   417   417   417   417
        417   417   417   834   417   4170
        3753  834   417   1251  417   1668
        417   417   1668  417   2085  417
        1251  1251  1668  417   417   834
        1251  1251  417   417   417   834
        834   417   1668  417   1251  834
        417   1251  417

    name 29
        417   417   417   417   417   417
        417   417   417   834   417   4170
        3753  834   417   1251  417   1668
        417   417   1668  417   2085  417
        417   417   417   1251  1251  834
        834   417   1251  2085  1251  417
        417   417   1251  834   417   834
        417   417   417

    name 30
        417   417   417   417   417   417
        417   417   417   834   417   4170
        3753  834   417   1251  417   1668
        417   417   1668  417   2085  417
        834   1668  1251  834   417   417
        1668  2085  417   417   417   417
        417   417   1668  417   834   417
        417   417   417   1251  417
    end raw_codes
end remote
```