# Equipping a robot with omnidirectional depth perception

eingereichter
INGENIEURPRAXIS-BERICHT
von

Christian Lück

geb. am 27.04.1989
wohnhaft in: Adlkofen
Setzensackstraße 1
84166 Adlkofen
Tel.: 08707/8356

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer: Cristian Axenie, David Weikersdorfer
Beginn: 04.03.2013
Abgabe: 06.04.2013

**Abstract**

Normalen Farb-Kameras sowie dem "event-based dynamic vision sensor" (eDVS) fehlt die inherente Möglichkeit die Entfernung eines Objektes von der Kamera zu messen. Während zahlreiche berechnende Methoden zur Entfernungsbestimmung vorgeschlagen wurden, fand man bis heute keine zufriedenstellende Lösung.
In diesem Projekt sollen mehrere aktive Tiefen-Kameras verwendet werden um einen kleinen Omnirobot mit 360°-Sicht auszustatten. Die Tiefen-Bilder der Kameras werden lokal mit Hilfe einiger Mini-Computer vorverarbeitet und anschließend über eine WLAN Verbindung zu einem Desktop-Computer gesendet.

**Abstract**

Normal color cameras as well as the event-based dynamic vision sensor (eDVS) lack the inherent possibility to estimate the distance of objects to the camera. While many computational methods for estimating depth have been proposed no satisfactory solution has been found to date.
In this project multiple active depth-sensing cameras will be used to equip a small Omnirob with 360 degree depth vision. The depth data will be captured and preprocessed locally on the robot using several low-profile mini-computers and streamed over a wireless LAN connection to a desktop computer.

# Contents

# Chapter 1

# Hardware

## 1.1 Raspberry Pi

The first Minicomputer i tried was the Raspberry Pi with Raspbian as operating system. It turned out that it is not powerful enough to handle more than one of the cameras. The device barely (95% cpu load) could read 320*240 depth images from two cameras simultaneously and transmit a downsampled 160*120 version of them. The downsampling approach had to be chosen because the device was too slow for proper image compression and also too slow to transmit the full resolution.

## 1.2 Android-Minicomputers

For the project two RK3066 powered Android-Minicomputers were available. One of them is called "Rikomagic MK802IIIs" and the other one is called "MK808". The only difference between both are the wireless connectivity peripherals. They have different Wifi-Chipsets and the MK808 additionally supports Bluetooth.
Because OpenNI and libfreenect are build for linux, Picuntu had to be installed on both of them.
Both devices built in Wifi-Modules have closed-source drivers only and it was not possible to use them with linux. An externel USB-Wifi-Adapter (rt2770) was used instead which intruduced big trouble in the project: Large USB- and Wifi-traffic caused the Network connection to drop erratically. Using a different USB-Wifi-Adapter (rtl8188cus) solved this problem.
Over all the RK3066 powered devices proved to be powerful enough to handle three Asus Xtion cameras each, where the bandwith of the wifi-connection is the bottleneck. One of those devices is able to read 320*240 sized depth-images from three cameras, compress them with snappy and transmit them over wifi at 30fps per camera with a cpu load smaller than 60%.

# Chapter 2

# Software

## 2.1 Used Libraries

### 2.1.1 libfreenect

According to my research on the internet, libfreenect seemed to be the way to go. The library is open-source, well documented, easy to use and very compact in the sense of code lines and code organisation. Unfortunately the Xtion-branch of the library is no longer under active development and because of that only the older Asus Xtion version (Device-Number 0x0600) worked out of the box. Getting the newer version to run (Device-Number 0x0601) required some minor modifications in the library's source code. The second drawback of libfreenect is that i could not get it to work on any ARM-architecture.

### 2.1.2 OpenNI

While beeing open-source too, this library is in contrast to libfreenect a rather confusingly organised piece of software. I rarely found the code sequences i was looking for because the code is distributed over many different files, folders, classes, functions and wrappers while the documentation of every single function only repeats the function name and links to some other functions. Apart from this the library worked without errors for both versions of the Asus Xtion on my x86 desktop-computer and all used ARM powered devices.

### 2.1.3 Snappy

Snappy is a lossless compression/decompression library that aims at beeing fast and therefore beeing applicable on low-profile hardware. In my case i used it to reduce the traffic on the wifi-connection. Experiments showed that the compression ratio snappy achieves with the Asus Xtion camera data is typically between 30% and 50%, depending on the particular image.

### 2.1.4 freeglut

Freeglut is a library on top of OpenGL that allows to easily create and manage windows containing OpenGL content. In my project i used a freeglut sample application to get the camera images displayed on the screen in a comfortable and efficient way.

## 2.2 The Server Application

A small server application was written, which is supposed to run on one of the mini computers to grab the depth-image of the camera(s) and send it via wifi. To interface the cameras, OpenNI is used because libfreenect seems to be incompatible with ARM-architectures. The gathered image-data gets split in smaller pieces and a small header is prepended on every fragment. The number of fragments per dept-image is hard coded but can be changed easily. The fragment header consists of two values: The first one is "uint32_t frameid" and the second one is "uint16_t fragid", whereas frameid is a per camera counter for the gathered images and fragid is a per image counter for the fragments. The fragments (including the header) are then sent to the client ip-address in one UDP-packet each.

**Syntax:**

**XtionServer PORT CLIENT-IP [NUMCAMERAS]**

**Arguments:**

**port:**

The application is using all ports in the intervall [port,port+numcameras] to send the image data. One port for each camera.

**client-ip:**

The ip-address of the client that is going to receive the data.

**numcameras:**

The number of cameras the application is going to handle.

**Examples:**

**XtionServer 30000 192.168.0.10 3**

Sending data of three cameras to a client with the ip address 192.168.0.10 on the ports 30000, 30001 and 30002.

## 2.3 The Client Application

As counterpart of the server application, a small client application was written to receive and display the camera data. The application is based on a GLUT sample application that creates a window and displays an image in it using OpenGL. The part i wrote receives the UDP-packets from the server, uncompresses them with snappy and reconstructs the complete image from the small fragments. When one fragment is lost in transmission, the corresponding fragment of an old image is

displayed. The number of fragments per image is hard coded but can be changed easily. Of course the number of fragments per image has to match on client and server side.

**Syntax:**

**XtionClient SERVER-IP PORT**

**Arguments:**

**server-ip:**

The ip-address of the server (with cameras).

**port:**

The port of the server where the server is sending the data.

**Examples:**

**XtionClient 192.168.0.10 30000**

Receiving data of a server that has the ip-address 192.168.0.10 and is sending camera-data on its port number 30000.

## 2.4 Testing

Transmitting the camera streams over wifi causes noticeable lag, stuttering and packet loss at the client side. To find out if this is a problem of my software or a general problem of wifi, i used a network benchmarking tool called "iperf". It consists of a server and a client and i configured the server to send a constant stream of UDP packets of approxmately the same bitrate my server sends camera data. On the client side i could see the packet loss i experienced with my software. So i assume that the problem is a general wifi related one.
Furthermore several tests were done to find out how the number of transmitted camera-streams is related to packet loss. The test was done with one, two and three cameras. The distance of the wifi connection was 2.7m with no obstacles in the line of sight.
Every test lasted 10 minutes and was repeated 3 times for each camera setup. For one camera it was measured that 6.2% of the frames were lost. With two cameras 8.7% and with three cameras 14.6%.

# Chapter 3

# Installation on OmniRob

The OmniRob is a small robot with a diameter of 30 cm and a heigth of 15cm. It has three omniwheels and can be controlled via wifi. The whole robot was constructed at the NST and build with the help of a laser cutter from 5mm POM material.
To install the six xtion-cameras on the robot a mounting-platform had to be build. The first draft of a camera mounting construction had a hexagonal shape with one camera on each edge. Because of the size of the cameras this hexagonal shape had to be a lot bigger than the robot, so this draft was discarded. The second and final approach consisted of two layers with a triagonal placement of the cameras on each layer.
The two layers are independed from each other and can be used separately. To gain 360 degree vision on the robot they can be stacked in a way that they are rotated 60 degrees to each other.
The electronics of each layer consists of two LiPo 3s batteries, a DC/DC-converter, a powered USB-hub, a USB-wifi-dongle and one of the android minicomputers. The construction has an on/off-switch and the batteries can be charged without opening.

# Chapter 4

# Timeplan

**Week 1:**
The desktop computer was set up with ubuntu and the required development tools. OpenNI and libfreenect were installed and tested.

**Week 2:**
First experiments with OpenNI on the Raspberry Pi. Software was written to transmit the camera data over network and display it on the desktop computer.

**Week 3:**
The android mini-computers were set up with Picuntu and OpenNI. Testing of my software was done on them and it turned out that a faulty wireless driver caused instability in the wifi connection.

**Week 4:**
Different protocols for network transmission were tested and snappy was used as compression library

**Week 5:**
The mounting platforms for the cameras on the OmiRob were constructed and built. The electronics for the platforms were soldered and everything was installed on the OmniRob.

# References

Conradt, J., Simon, P., Pescatore, M., and Verschure, PFMJ. (2002). Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, Int. Conference on Artificial Neural Networks (ICANN2002), p. 795-800, Madrid, Spain.

Conradt, J., Tevatia, G., Vijayakumar, S., & Schaal, S. (2000). On-line Learning for Humanoid Robot Systems, International Conference on Machine Learning (ICML2000), Stanford, USA.