Hochschule Reutlingen
Reutlingen University

Hochschule Reutlingen

Reutlingen University

Studiengang Mechatronik Master

– Image Understanding –

Trainieren von Body Gestures zur Steuerung des SCITOS powered by Kinetic Space [1]

Name: Jürgen Nahm und Philipp Stark

Testat:

Abkürzungsverzeichnis

Scitos Mobiler Service Roboter der Firma Metralabs [2]

Leonie Bezeichnung des Scitos der Hochschule Reutlingen

Kinetic Space Anwendungsprogramm von Prof. Dr. Matthias Wölfel [1] zur Erkennung von Körpergesten mit der Microsoft Kinect [3]

UDP User Datagram Protocol [4]

Mira Center Anwendungsprogramm zur Steuerung des Scitos

XML Extensibel Markup Language

PIPE Spezielle Umleitung innerhalb der Linux Konsole [5]

Kinect 3D Kamera des Herstellers Microsoft [3]

Processing Java ähnliche Programmiersprache

IDE Integrated Development Environment

OSI Open System Interconnection Model

OSC Open Sound Control

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Die vorliegende Ausarbeitung wurde im Rahmen eines Praktikums im Fach Image Understanding des Masterstudiengangs Mechatronik der Hochschule Reutlingen verfasst.

Als Ziel galt es das theoretisch erlernte Wissen aus diesem Themengebiet anhand eines Projekts praktisch anzuwenden und anhand einer zugrunde liegenden Problemstellung darüber hinaus methodisch zu einer Lösung zu kommen.

1.1 Aufgabenstellung

Als Aufgabenstellung dieses Praktikumsprojekts gilt es über eine handelsübliche Microsoft Kinect Kamera [3] Fahrbefehle für den SCITOS [2] Assistenzroboter zu generieren.

Aufgrund der Tatsache, dass diese Aufgabenstellung sehr umfangreich ist werden hieraus verschiedene Arbeitspakete definiert. Die Reihenfolge der Aufzählung dieser Pakete gibt die Gliederung der vorliegenden Ausarbeitung wieder

Arbeitspaket

a) Einarbeitung in das Vorgänger-Projekt [11]

b) Konzepterstellung und Netzwerkkommunikation über UDP

c) Realisierung der Client-Kommunikation

d) SCITOS Serveranbindung, Gestenauswertung und Generierung der Fahrbefehle.

2 Konzepterstellung und Netzwerkkommunikation über UDP

Dieses Kapitel stellt zu Beginn im Prinzipiellen das Konzept vor welche Komponenten auf welcher Hardwareeinheit benutzt werden und geht dann im zweiten Teil näher auf die Netzwerkkommunikation über UDP ein.

2.1 Hardwareaufbau und verwendete Komponenten

Die Abbildung 2.1 stellt den gesamt Überblick sowie die einzelnen Verwendeten Komponenten dar. Die Steuerung des SCITOS Assistenzroboters erfolgt im wesentlichen in den folgenden Schritten (Beschreibung zur Abbildung 2.1 von links nach rechts).
Dies Vorgehensweise ist Sinngemäß an Quelle [7] angelehnt.

Schritte

1. Gesten werden über eine handelsübliche Microsoft Kinect [3] Kamera aufgezeichnet
2. Kinect [3] Daten werden über den Computer in der KineticSpace [1] Software, bereitgestellt von Herrn Professor Wölfel von der Hochschule Furtwangen, verarbeitet und mit hinterlegten Gesten verglichen um letztendlich bei Übereinstimmung einen positiven Ausschlag im Tool unter *Current Gesture Match* sowie eine Konsolenausgabe im Terminal mit der Meldung „*found gesture #'XY'* “ zu erzeugen
3. Auf dem Desktop PC läuft parallel zu der KineticSpace [1] Software ein Clientprogramm welches als Eingabe die vom Tool erzeugten Ausgabestrings über den angeschlossenen Router versendet.
4. Auf dem SCITOS Assistenzroboter „*Leonie*“ läuft im Miracenter unter Anderem ein Serverprogramm welches die vom Client versendeten Daten über WLAN empfängt und den String nach bestimmten Gesten filtert, welche wiederum bestimmt Fahrbefehle für den SCITOS generieren.

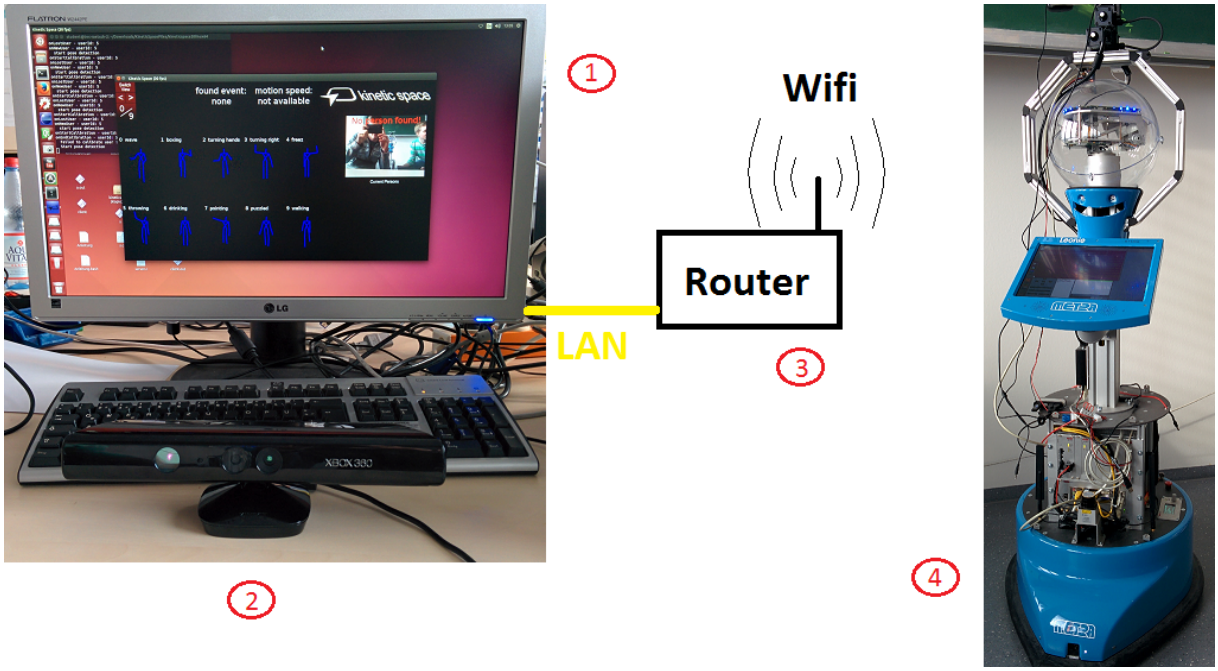

Abbildung 2.1: Hardwareaufbau der SCITOS Ansteuerung

Zusammengefasst kann die Kommunikation zwischen Bildverarbeitung mit der Kinect [3], der Datenübertragung hin zum SCITOS Assistenzroboter sowie die Generierung der Fahrbefehle in dem verwendeten Konzept als unidirektional bezeichnet werden. Eine vereinfachte Blockdarstellung ist der Abbildung 2.2 zu entnehmen.

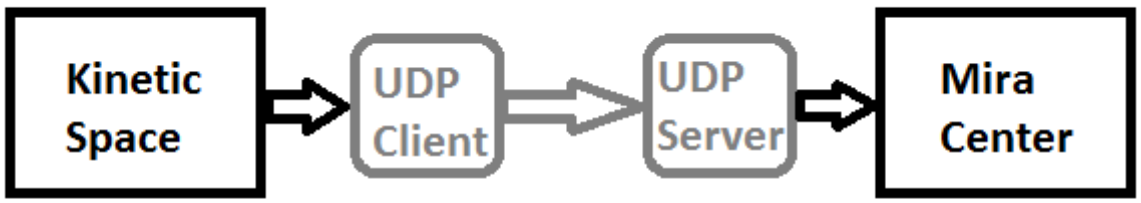

Abbildung 2.2: Blockdarstellung des verwendeten Konzepts

2.2 Netzwerkkommunikation über UDP

Im vorhergehenden Unterkapitel wurde die Blockdarstellung des kompletten Aufbaus dargestellt. Erkennbar wird hieraus, dass für die Kommunikation auf das Transportprotokoll User Datagram Protocol (UDP) aufgesetzt wurde. Die Abbildung 2.3 stellt auf der linken Seite das OSI-Schichtenmodell [9] vor und auf der rechten Seite ist die von uns verwendete Struktur dargestellt.

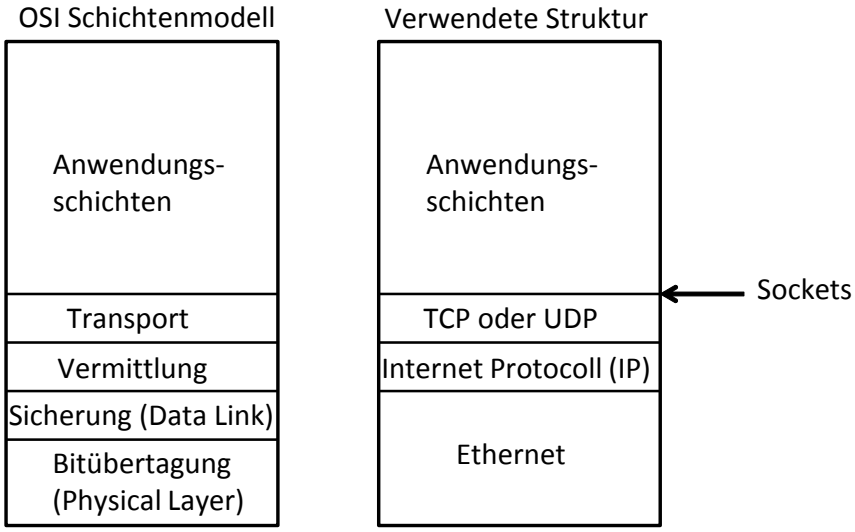

Abbildung 2.3: Open System Interconnection Model (OSI) [9]

In der dritten Schicht, der Transportschicht, wurde von uns das UDP Protokoll ausgewählt. Für unsere Verwendung hat UDP einige Vorteile gegenüber TCP. Die für uns relevanten **Vorteile** sind sinngemäß Quelle [4]:

- schnellerer Kommunikationsaufbau möglich (kein Drei-Wege-Handschlag wie bei TCP)
- kleinere Datenpakete können schneller übertragen werden (schlankeres Übertragungsprotokoll dadurch höherer Datendurchsatz)
- geringere Übertragungsverzögerungsschwankungen durch Auslassen von wiederholtem Senden.

Aus den genannten Vorteilen lässt sich auch ein großer Nachteil ableiten. UDP verzichtet im Gegensatz zu TCP auf nahezu alle Kontrollfunktionen. Somit kann es vorkommen, dass die Nachricht die zum Zeitpunkt $t(n+1)$ versendet wird die Nachricht $t(n)$ überholt. Auf dieses Projekt bezogen stellt das allerdings keinen Nachteil dar, da zwischen zwei

erkannten Gesten genug Zeit vergeht, sodass alle in der korrekten Reihenfolge übertragen werden können.

Der UDP Header beinhaltet den Quell- und Zielport, die Anzahl der gesamten Länge des Datagramms (maximal $2^{16} - 1$) sowie die Prüfsumme. Die insgesamte Länge des Headers, zu sehen in Abbildung 2.4, beträgt 8 Bytes [4].

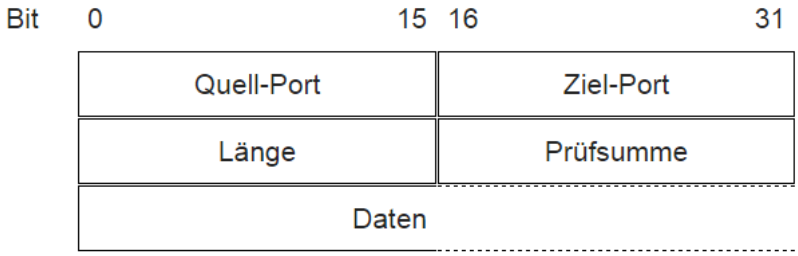

Abbildung 2.4: Header des UDP Datagramms [4]

Das verwendete IPv4 Internet Protokoll setzt seinerseits ebenfalls noch einen Header vor das gesamte UDP-Datagramm. Für die maximal verfügbare Länge des UDP Datenbereichs aus Abbildung 2.5 ergibt sich somit eine Länge von 65.507 Bytes (65.535 − 8 Bytes − 20 Bytes Ipv4 Header)[4].

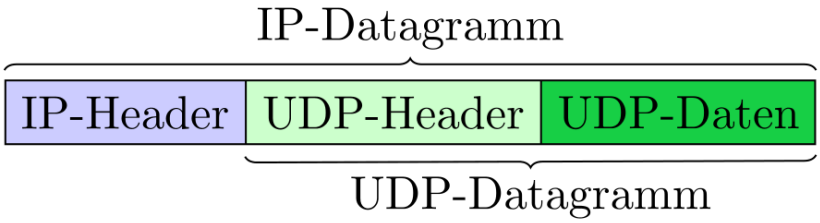

Abbildung 2.5: Gesamtes, zu übertragendes Datagramm[4]

3 Erkennung der Körper-Gesten

Zur Erkennung von Körper Gesten, dient die Pro Version des von Prof. Dr. Matthias Wölfel erstellten Kinetic Space[1]. Dies ist ein in Processing geschriebenes Tool, welches auf einer Microsoft Kinect[3] Basiert. In Abbildung 3.1 ist dargestellt,in welchem Verzeichnis sich das Anwendungsprogramm inklusive zugehöriger Bibliotheken befindet.

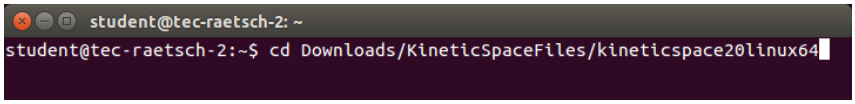

Abbildung 3.1: Aufruf des Kinetic Space [1] Arbeitsverzeichnis in der Linux Konsole

Ein Start der Kinetic Space [1] Anwendung kann nun durch die Eingabe **./kineticspace** erfolgen.
Abbildung 3.2 zeigt die GUI von Kinetic Space [1]. Oben links ist die Menü leiste zu sehen. Oben rechts befindet sich das Kamera Bild, welches bei einer oder mehrerer erkannter Personen zu einem Drahtmodell ändert. In der Mitte sind neun, vom Anwender frei programmierbare Körpergesten zu sehen.

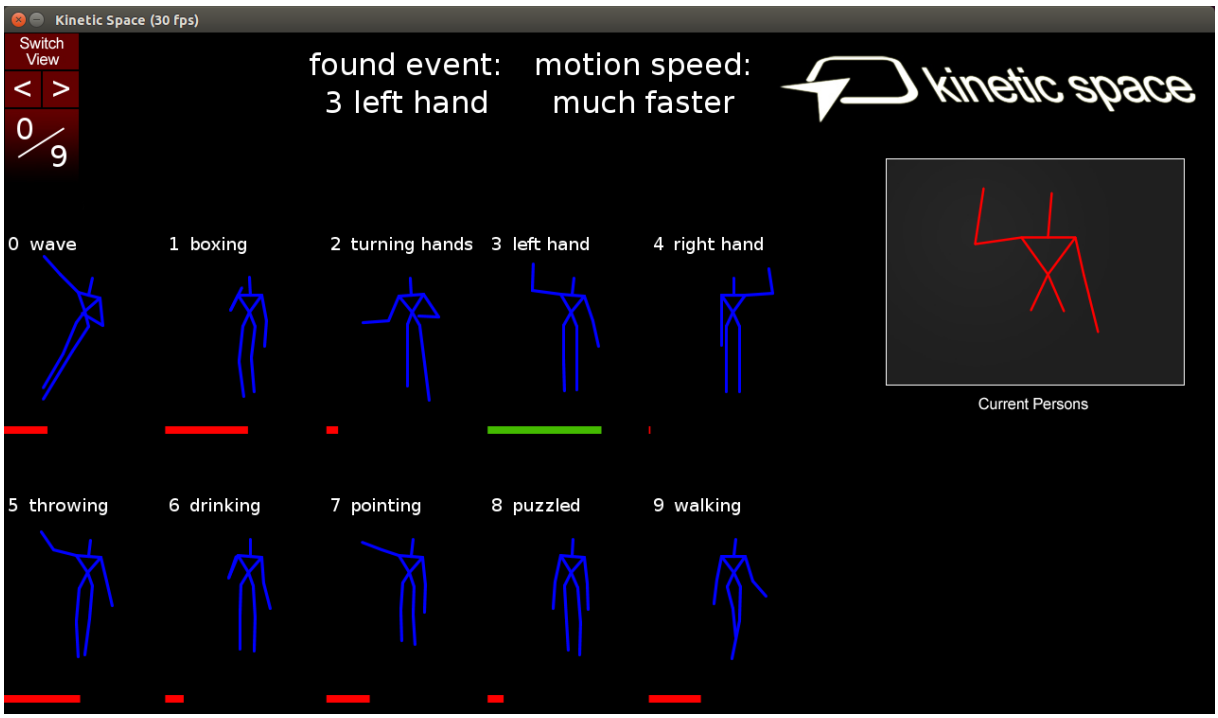

Abbildung 3.2: Benutzeroberfläche von Kinetic Space [1] nach dem Aufruf des Programmes

Abbildung 3.3 stellt das lernen/teachen von neuen Körpergesten dar. Diese Funktion erreicht man, durch betätigen der Pfeiltasten am oberen linken Rand der Anwendung. Für eine Ausführliche Beschreibung der Anwendung, möchten wir hier gerne an die Dokumentation von Herrn Prof. Dr. Matthias Wölfel verweisen[1].

Abbildung 3.3: Teach in Modus für neue Körpergesten [1]

Abbildung 3.4 zeigt, die Ausgabe auf der Linux Konsole, der in Abbildung 3.2 dargestellten, erkannten Körpergeste. Eine Beschreibung dieser Werte erfolgt in Abschnitt 4.

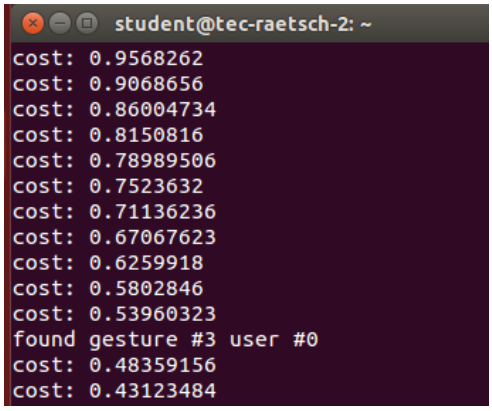

Abbildung 3.4: Konsolen Ausgabe von Kinetic Space [1]

4 Realisierung der Client-Kommunikation

In Kapitel 2.1 wurde bereits im Konzept kurz erwähnt, dass die Ausgabe die das Programm KineticSpace [1] erzeugt an ein parallel ablaufendes Programm als Eingabewerte übergeben wird um anschließend über das Netzwerk versendet zu werden. Im Detail erfolgt diese Parallelität über sogenannte Linux-Pipes.

Das Tool KineticSpace [1] nutzt als Ausgabetool das Terminal. Hierbei schreibt es den „*cost*“ Faktor, welche das Ergebnis der Überdeckung von hinterlegter Geste mit erkannter Geste wiedergibt sowie bei erfolgreicher Überdeckung den Text „*found gesture 'XY'* “ in das Terminal.

Diese Ausgabewerte werden nun mit Hilfe der erwähnten Pipes [5] von der Konsole ausgelesen und als Eingabewerte für den UDP-Client genutzt. Der Client wiederum leitet diese Werte weiter an den Server, welcher auf dem SCITOS Assistenzroboter in der MIRA Umgebung läuft. *Leonie* hat eine feste, statische IP-Adresse mit der Nummer „192.168.1.201“. Dies ermöglicht einen Verbindungsaufbau zu jeder Zeit.

Der Client sowie der Server wurden in der Programmiersprache C geschrieben. Der Quellcode des Clients wird im Folgenden dargestellt und stammt aus der Quelle [6]. Der Quelltext ist komplett durch kommentiert und gut zu verstehen. Die prinzipielle Programmstruktur gliedert sich im Groben wie folgt

Zeile

18 Da UDP nahezu keine Kontrollmechanismen besitzt übernimmt der Client sowie der Server teile dieser Aufgabe

35 Transportmedium „socket“ anlegen, ebenfalls dargestellt in der Abbildung 2.3. Dieser beinhaltet alle Informationen zu Adresse Server, verwendetes Internet Protokoll sowie ob UDP oder TCP als Transportmedium genutzt werden soll

ab 58 Sendedauerschleife. Der Buffer wird mit dem Text des Terminals befüllt (siehe Zeile 63), welcher wiederum von der Ausgabe der KineticSpace [1] Software stammt.

65 Senden der Message über die global angelegte SERVER-Adresse sowie den hinterlegten Port.

Listing 4.1: C-Code des UDP-Clients [6]

```
1  /*
2  UDP Client
3  */
4  #include<stdio.h>
5  //printf
6  #include<string.h>
7  //memset
8  #include<stdlib.h>
9  //exit(0);
10 #include<arpa/inet.h>
11 #include<sys/socket.h>
12 #define SERVER "192.168.1.201"
13 //Serveradresse
14 #define BUFLEN 1024
15 //Bufferlaenge
16 #define PORT 8888
17 //Sendeport
18 void die(char *s)
19 //Wenn ein Systemaufruf fehlschlaegt erscheint eine
20 //Fehlermeldung und das Programm wird abgebrochen
21 {
22 perror(s);
23 exit(1);
24 }
25
26 /*--------------------Hauptprogramm-----------------------------*/
27 int main(void)
28 {
29 struct sockaddr_in AdressStorage;
30 //Variable AdressStorage der Struktur sockaddr_in beinhaltet
31 //verschiedene Komponenten zum Speichern von Adressen
32 int s, i, slen=sizeof(AdressStorage);
33 char buf[BUFLEN];
34 char message[BUFLEN];
35 if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
36 {
37
38 die("socket");
39 }//UDP Socket erstellen. Wenn Rueckgabewert der Funktion socket -1
```

```
    ist, ist die Kommunikationsverbindung fehlgeschlagen
40
41  //SOCK_DGRAM gibt an, dass UDP Sockets verwendet werden fuer TCP
        Sockets wird SOCK_STREAM verwendet
42
43  memset((char *) &AdressStorage, 0, sizeof(AdressStorage));
44  //Beschreibt den kompletten Speicherbereich der Variablen
45  //AdressStorage mit Nullen
46  AdressStorage.sin_family = AF_INET;
47  //Adressfamilie des Netzwerk wird auf "Internet" gesetzt, auf IPv4.
        Fuer IPv6 muss AF_INET6 eingegeben werden
48  AdressStorage.sin_port = htons(PORT);
49  //Konvertierung und Deklaration der Portnummer
50
51  if (inet_aton(SERVER , &AdressStorage.sin_addr) == 0)
52
53  {
54  fprintf(stderr, "inet_aton()␣fehlgeschlagen\n");
55  exit(1);
56  }//inet_aton liefert eine Null falls die IP Adresse keine Valide
        Adresse ist
57
58  while(1)
59  //Sendedauerschleife
60  {
61  printf("Message␣eingeben␣:␣");
62  bzero(buf,BUFLEN);//Befuellt den Buffer mit Nullen
63  fgets(message,(BUFLEN-1),stdin);//Schreibt die Standarteingabe stdin
        in das Array message
64
65  if (sendto(s, message, strlen(message) , 0 , (struct sockaddr *) &
        AdressStorage, slen)==-1)
66  {
67  die("sendto()");
68  }//Message ueber Netzwerk versenden. Sendto liefert -1 zurueck wenn
        Uebertragung fehlgeschlagen ist
69
70  memset(buf,'\0', BUFLEN);
71  //Befuellt Speicherbereich von Buffer mit Nullen
72
```

```
73 /*-------------------------------------------------------------------------*/
   //Falls keine Daten vom Server zurueckgesendet werden kann dieser
   Block auskommentiert werden.
74
75 if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &AdressStorage, &
   slen) == -1)
76
77 {
78 die("recvfrom()");
79 }
80
81 puts(buf);
82 //Falls Daten zurueck kommen werden diese im Terminal Ausgegeben
83 /*-------------------------------------------------------------------------*/
84 }
85
86 close(s);
87 return 0;
88 }
```

Das anschließende Kompilieren des Quelltextes kann zum einen durch eine IDE erfolgen und zum anderen wie in Quelle XY beschrieben mithilfe der Kommando Zeile mit folgendem Befehl:

```
1 student@tec-raetsch-2:~$ gcc client.c -o client && ./client
```

Soll der kompilierte Code nicht gleich ausgeführt werden muss der hintere Teil „&& ./client“ entfernt werden.

Der Kompilierte Quellcode kann nun, durch die erwähnte Linux-Pipe [5], dargestellt mit dem senkrechten Strich, mit Kinetic Space [1] verbunden werden (siehe Abbildung 4.1)

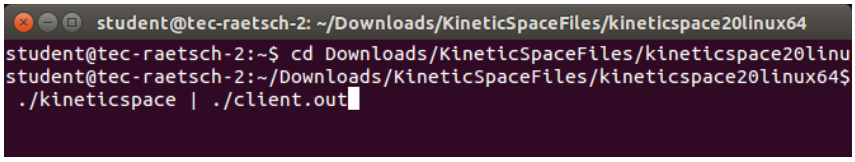

Abbildung 4.1: Pipen des Clients mit dem KineticSpace [1] Programm

ACHTUNG: Damit der Client eine Verbindung mit dem Server aufbauen kann, muss vor dessen Start natürlich der Server auf der Leonie gestartet werden!!!

5 Serveranbindung, Gestenauswertung und Generierung der Fahrbefehle

In diesem Abschnitt soll gezeigt werden, wie die Anbindung den Server (in diesem Leonie) realisiert wurde.

Zunächst wurde für das Miracenter ein neues Projekt angelegt. Dies erfolgte gemäß der Anleitung nach XY. Anschließend wurde wie in XY beschrieben ein XML File erstellt.

Listing 5.1: Konfiguration des XML Files für das Mira Center auf dem Scitos

```
1 <root>
2 <unit class="phildomainname::philunitname" id="philunitname" />
3 </root>
```

Anschließend wurde das Projekt, wie in XY beschrieben, im Terminal mit dem **make** kompiliert.

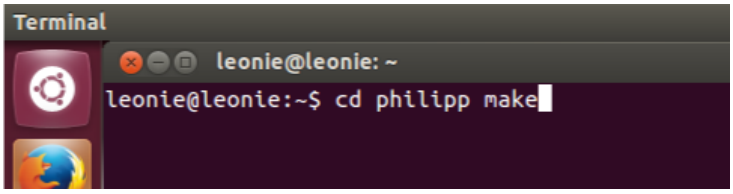

Abbildung 5.1: Konsolen Aufruf zum Kompilieren des Mira Projektes [7]

Es wird daraufhin ein C-File erzeugt, in welches der Quelltext für die udp Kommunikation, sowie für die Steuerung des Scitos Roboters eingefügt werden kann. Es wird eine so genannte Process Funktion erstellt, welche Zyklisch aufgerufen wird. Innerhalb dieser wurde der Quelltext eingefügt.

Listing 5.2: Zyklisch ausgeführter Code im Mira Center(Server-Anwendung) [7] [6]

```
1 void philunitname::process(const Timer& timer)
2 {
3 // TODO: this method is called periodically with the specified cycle
     time, so you can perform your computation here.
4
5 struct sockaddr_in si_me, si_other;
6
```

```
7  int s, i;
8  socklen_t slen = sizeof(si_other) , recv_len;
9  char buf[BUFLEN];
10 char needle[]="found gesture";
11 float headspin=60;
12
13 //create a UDP socket
14 if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
15 {
16 die("socket");
17 }
18
19 // zero out the structure
20 memset((char *) &si_me, 0, sizeof(si_me));
21
22 si_me.sin_family = AF_INET;
23 si_me.sin_port = htons(PORT);
24 si_me.sin_addr.s_addr = htonl(INADDR_ANY);
25
26 //bind socket to port
27 if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
28 {
29 die("bind");
30 }
31
32 //keep listening for data
33 while(1)
34 {
35 //printf("Waiting for data...");
36 fflush(stdout);
37
38 //try to receive some data, this is a blocking call
39 if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &
      si_other, &slen)) == -1)
40 {
41 die("recvfrom()");
42 }
43
44 //print details of the client/peer and the data received
45 if(strstr(buf,needle))
```

```
46 {
47 printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr),
       ntohs(si_other.sin_port));
48 printf("String enthaelt Found Gesture\n");
49 callService<void>("/robot/Robot", "moveHeadLeftRight",headspin);
50 }
51 // else
52 //   {
53 // printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr
       ), ntohs(si_other.sin_port));
54 // printf("Data: %s\n" , buf);
55 //  }
56 //now reply the client with the same data
57 if (sendto(s, buf, recv_len, 0, (struct sockaddr*) &si_other, slen)
       == -1)
58 {
59 die("sendto()");
60 }
61 }
62 close(s);
63 }
64
65 ////////////////////////////////////////////////////////////
66
67 }
```

Daraufhin kann der Quelltext mithilfe des Miracenter geladen werden. Abbildung 5.2 stellt den Aufruf des Miracenters inklusive XML File und benötigten Zusatz Libaries dar.

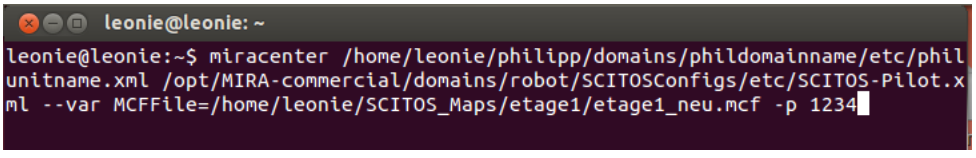

Abbildung 5.2: Konsolen-Aufruf der Server-Anwendung auf dem Scitos

Nach dem Aufruf ist Abbildung 5.3 zu sehen. In Abbildung 5.3 wurde vor Erstellung des Screenshots die Kinetic Space [1] Anwendung auf den Cleint gepipt, so dass eine Ausgabe im Server(Scitos) sichtbar ist. Wie bereits im vorherigen Abschnitt beschrieben filtert die Anwendung auf Strings, und gibt bei einem Match die Geste inkl. IP Adresse des Senders aus. Daraufhin führt der Scitos einen Fahrbefehl, in diesem Fall Kopfdrehen, aus.

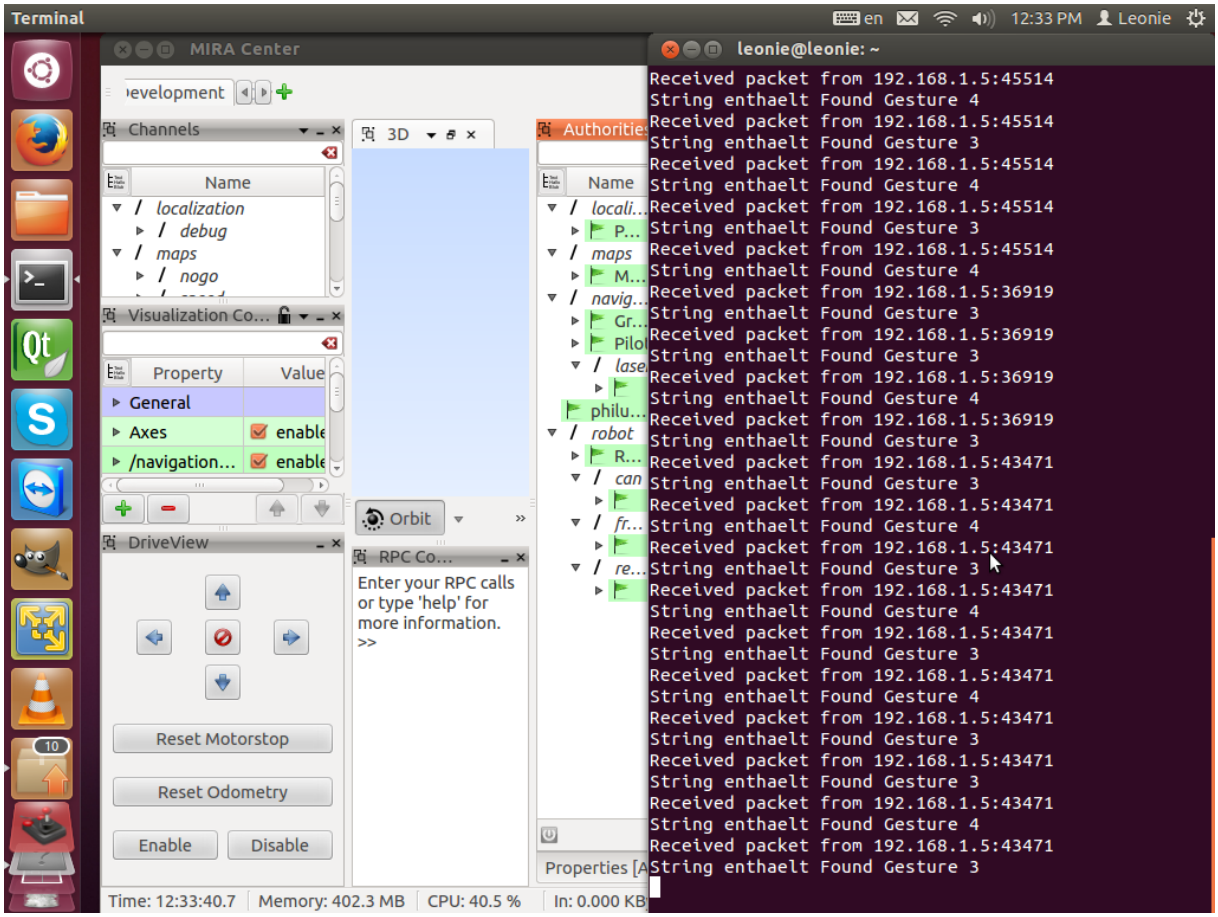

Abbildung 5.3: Konsolen-Ausgabe der Server-Anwendung auf dem Scitos

5.1 Alternativer Versand von UDP Nachrichten

Als Alternative zu dem Client Programm aus Abschnitt 4 soll hier die Anwendung Netcat erläutert werden. Die Quelle für diesen Abschnitt ist [8]

Netcat ist ein, in Ubuntu bereits enhaltenes Konsolen Tool zum Versand/Empfang von UDP oder gar TCP Nachrichten. Im folgenden soll ein kurzes Beispiel gegeben werden, wie Anwendung Kinetic Space [1] auf Netcat gepipt wurde, und die Daten per UDP an den Localhost (IP 127.0.0.1) gesendet und mit diesem auch wieder empfangen wurden.

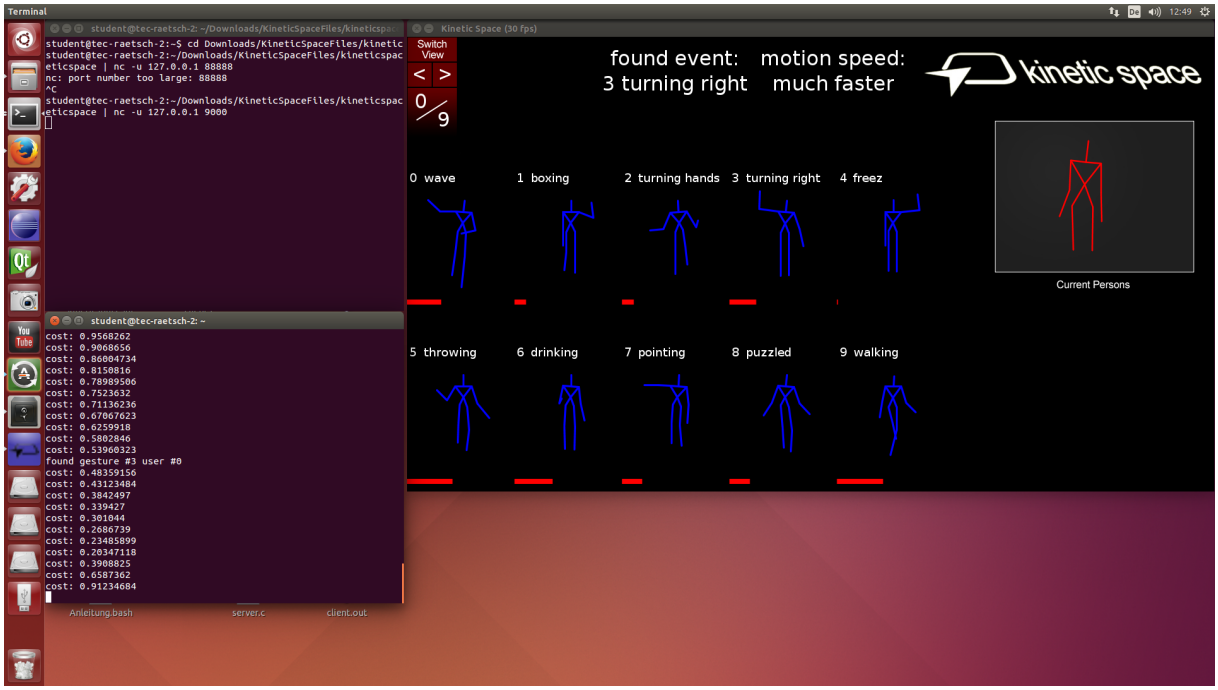

Abbildung 5.4: Beispielhafte Anwendung von Netcat auf dem Local Host [1]

In Abbildung 5.4 oben rechts, ist die Kinetic Space [1] Anwendung von Prof. Dr. Matthias Wölfel zu sehen. Diese wurde im Terminal oben links auf Netcat gepipt. Das **nc** ist der Aufruf von Netcat. Udp wird durch **-u** erreicht. es folgt die IP-Adresse, in diesem Fall der Local Host **127.0.0.1**. und Abschließend eine freier Port in diesem Fall **9000**. Im Terminal unten links wurde Netcat als Empfänger eingerichtet. Der Aufruf hierfür erfolgt durch.

Listing 5.3: Konsolen Aufruf von Netcat als UDP Empfänger sinngemäß nach [8]

```
1 nc -u -l 9000
```

Der Aufruf erfolgt analog zu dem Aufruf des Netcat Senders, mit dem Unterschied des **-l**. Dieses signalisiert das es sich um einen Empfänger handelt. Ebenso, **kann** beim Empfänger die IP-Adresse des Senders weggelassen werde

Für eine ausführliche Beschreibung von Netcat sei der **help** Befehl der Konsole empfohlen. Mit diesem lassen sich zu jedem Parameter hilfreiche Beschreibungen finden.

6 Fazit & Ausblick

Zusammenfassend kann diesem Praktikumsprojekt aus dem Fach *Image Understanding* entnommen werden, wie

a) prinzipiell die Körpergestenerkennung mit der Microsoft Kinect Kamera [3] realisiert wird

b) im Allgemeinen eine Client-Server Netzwerkkommunikation basierend auf dem UDP Protokoll realisiert wird

c) Strings ausgewertet werden und hieraus Fahrbefehle für den SCITOS Assistenzroboter generiert werden können.

Das Vorgängerprojekt spricht im Abschnitt *Fazit* die Option an, die Filterung der Strings über das Open Sound Control (OSC) Protokoll [10] zu realisieren. Diese Möglichkeit ist für die benötigte Verwendung unserer Ansicht nach zu umständlich und würde verhältnismäßig zu viel Zeitaufwand benötigen um dies erfolgreich umsetzen zu können. Eine Realisierung der Filterung mit Hilfe der Linux Pipes um die Strings an den SCITOS Assistenzroboter weiter zu leiten um die Filterung dort im C-Code zu realisieren führt schneller und zielführender zum Erfolg.

Wie bereits erwähnt wurde, wurde aus Sicherheitsgründen das Generieren von Fahrbefehlen aus erkannten Gesten in dieser Arbeit so durchgeführt, dass wir vorerst nur den Kopf haben drehen lassen. Als **weiterführende Tätigkeit** kann somit folgendes definiert werden

1. Kopfdrehen durch konkrete Fahrbefehle ersetzen
2. Autorisiertes erteilen von Fahrbefehlen. Autorisierung kann in Kooperation mit der Gruppe *NAO Gesichtserkennung* durchgeführt werden.
3. Auswertung Kinect Mikrofone, sodass Leonie über Gesten und über die Sprache gesteuert werden kann
4. Leonie imitiert Bewegungen ohne Lerndaten durch Auswertung der Joint Koordinaten.

Literatur

[1] URL: https://code.google.com/p/kineticspace/.

[2] URL: http://metralabs.com/index.php?option=com_content&view=article&id=67&Itemid=66.

[3] URL: http://www.microsoft.com/en-us/kinectforwindows/develop/downloads-docs.aspx.

[4] URL: http://de.wikipedia.org/wiki/User_Datagram_Protocol.

[5] URL: http://wiki.ubuntuusers.de/Shell/Umleitungen.

[6] URL: http://www.binarytides.com/programming-udp-sockets-c-linux/.

[7] URL: http://projekte.rt-lions.de/SCITOS/FollowHeadPose/FollowHeadPose_Ausarbeitung.zip.

[8] URL: http://wiki.ubuntuusers.de/netcat.

[9] URL: http://de.wikipedia.org/wiki/OSI-Modell.

[10] URL: http://de.wikipedia.org/wiki/Open_Sound_Control.

[11] Shasindran Poonudurai und Alexander Mielchen. *Gesture Recognition with MS Kinect Sensor - Hochschule Reutlingen*. 2014.