HOCHSCHULE REUTLINGEN
Reutlingen University

- Studiengang Mechatronik Master -

Laborprojekt Image Understanding

FaceShift, SCITOS „Look at me!“

Thiemo Frank, Jens Wagner

Inhaltsverzeichnis

1 Aufgabenstellung

Ziel dieser Projektarbeit ist es eine Interaktion zwischen dem Service-Roboter SCITOS und menschlichen Personen zu ermöglichen. Hierfür soll der Gaze (Blick) und die Headpose (Kopfausrichtung) einer Person mit der FaceShift SDK ermittelt werden. An Hand der gewonnen Daten soll der SCITOS auf die Kopfbewegung der gegenüberstehenden Person reagieren und seinen eigenen Kopf so drehen, dass er in die gleiche Richtung ausgerichtet ist. Schaut die Person dem SCITOS direkt ins Gesicht, soll per Gaze-estimation erkannt werden ob die Aufmerksamkeit der Person auf dem Roboter liegt. Als 3D-Kamera wird eine Asus Xtion pro verwendet.

Die Projektarbeit gliedert sich in folgende Teilaufgaben:

- Einarbeitung FaceShift SDK,
- Ermitteln der 6 Freiheitsgrad des Kopfes einer Person,
- Ausrichten des SCITOS-Roboterkopfes auf diese Person,
- Awareness recognition,
- Dokumentation.

2 FaceShift SDK

2.1 Was ist FaceShift?

FaceShift wurde zur Analyse der Bewegung und des Gesichtsausdrucks eines Schauspielers entwickelt. Dies geschieht über einen Mix aus Gesichtsausdruck, Position des Kopfes und Blickrichtung. Die gewonnen Daten werden zum Animieren von virtuellen Charakteren in Spielen oder Filmen verwendet.

Für die Projektarbeit sind hierbei die einzelnen Tracking-Features der Software interessant. Folgende Tracking-Features können in Echtzeit als Stream ausgegeben werden:

- Expressions,
- Gaze,
- Head Pose.

In der Projektarbeit wird die 30 Tage Trial-Version der FaceShift Studio-Version verwendet. Diese kann auf `http://www.faceshift.com/get-trial/` heruntergeladen werden.

2.2 Anlegen eines Profils in FaceShift

Um eine Person erfolgreich zu Tracken muss zunächst ein personalisiertes Profil dieser Person angelegt werden. Dies geschieht in FaceShift unter „Training“. Hier müssen verschiedene Gesichtsausdrücke gescannt werden. Nachdem alle Gesichtsausdrücke gescannt wurden, kann mit dem Button „Build Profile“ das 3D-Modell erstellt werden.

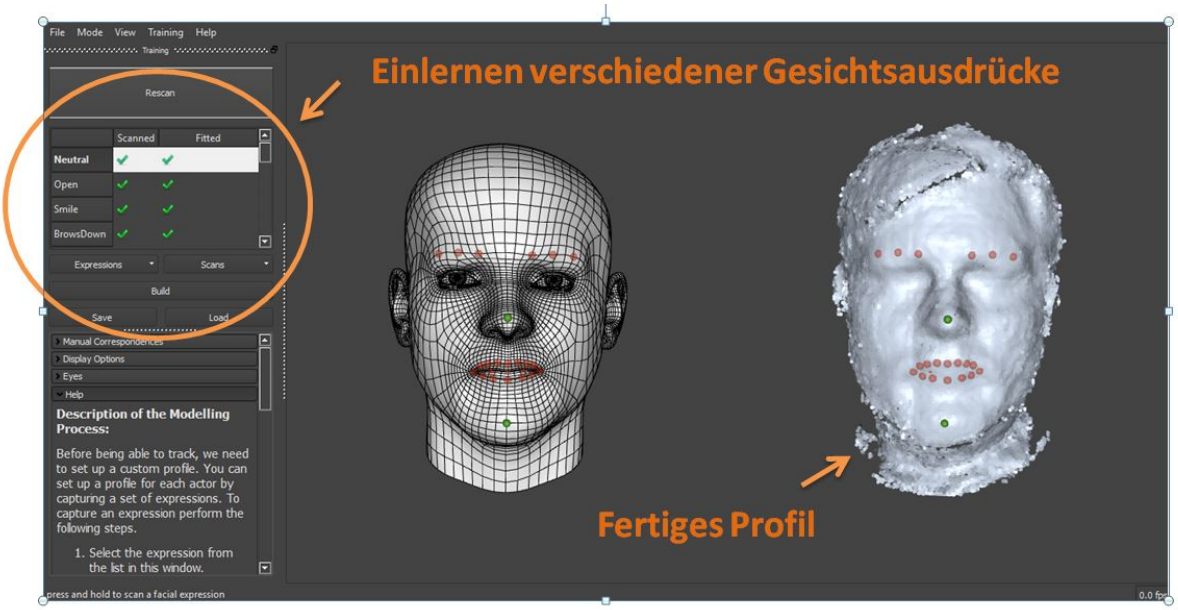

Abbildung 2.1: Personalisiertes Profil

2.3 Tracking einer Person

FaceShift bietet die Möglichkeit eine Person live in der Software zu tracken. Abbildung 2.2 zeigt die Funktion. Auf dem Bild ist das Live-Bild und das Modell mit der erkannten Ausrichtung zu sehen. Des Weiteren werden die aktuellen Yaw/Pitch/Roll Werte angezeigt. In Balkenform wird prozentual angezeigt welcher Gesichtsausdruck aktuell erkannt wird.

2.4 Ausgabedaten per Netwerk-Stream

Unter Einstellungen kann in FaceShift ein Netzwerk-Stream (Abbildung 2.3) eingestellt werden. Dieser sendet Daten per TCP/UDP an eine feste IP-Adresse. Diese Daten können von einem Netzwerk-Client empfangen werden und im Benutzerprogramm weiterverarbeitet werden.

Die Konfiguration erfolgt unter „File/Preferences/Streaming“ Hier muss als Protokoll UDP und die Host-IP eingestellt werden(Abbildung 2.4).

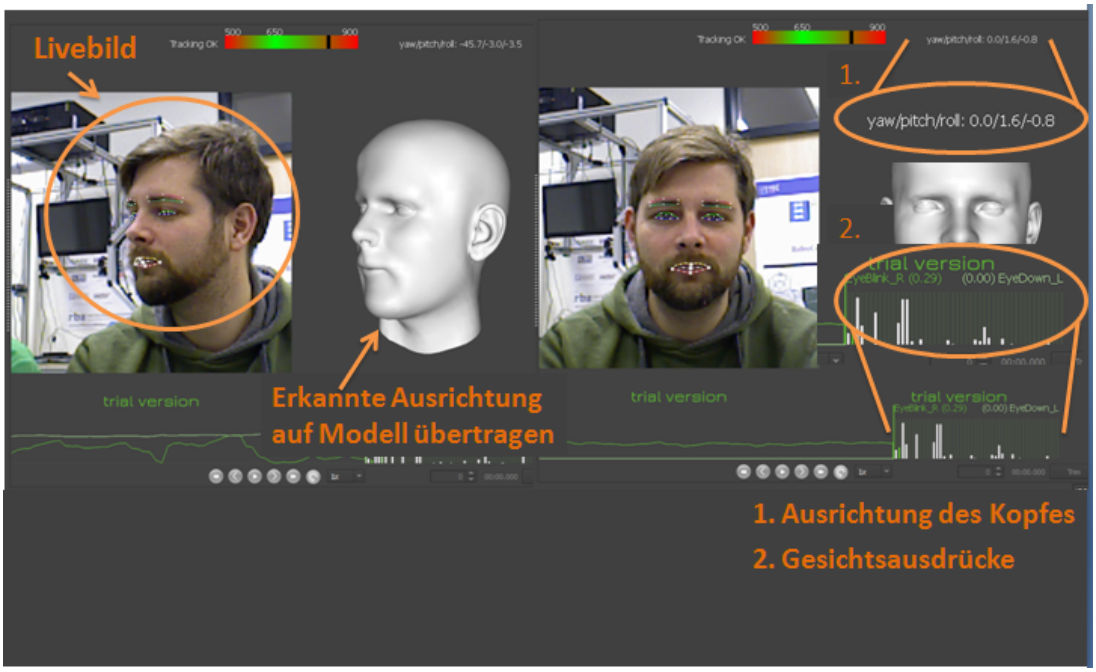

Abbildung 2.2: FaceShift Live-Tracking

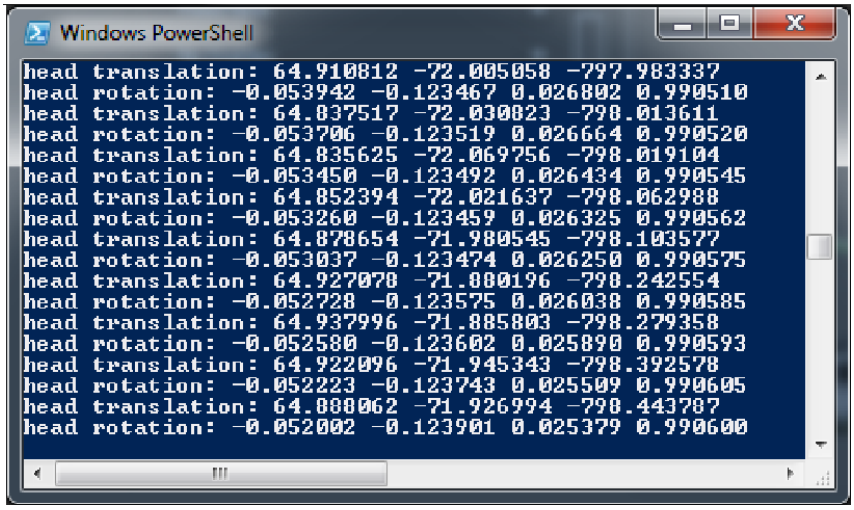

Abbildung 2.3: FaceShift Netzwerk-Stream in der Kommandozeile

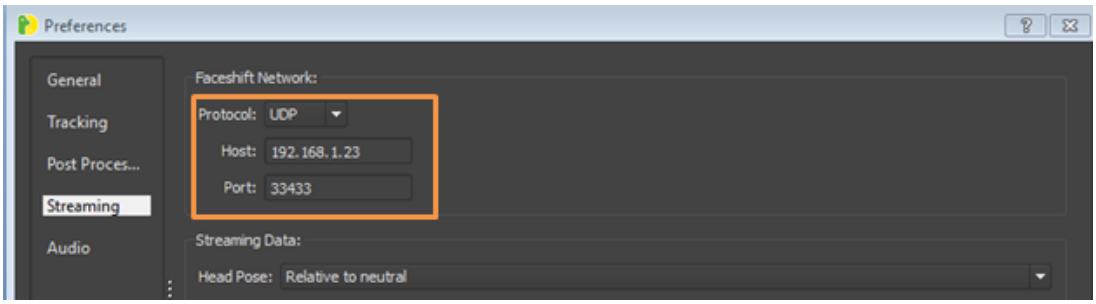

Abbildung 2.4: FaceShift Einstellen der IP-Adresse

3 Implementierung am SCITOS

3.1 Einrichten einer VirtualBox für Windows

Da von FaceShift zur Zeit noch keine Linux Version erhältlich ist und das Mira-Center des Scitos nur auf Linux-Basis betrieben werden kann ist es notwendig eine VirtualBox auf Linux zu installieren. Auf dem virtuellen Windows kann FaceShift installiert und in Betrieb genommen werden.

Hierfür wird die VirtualBox 4.3.6 von Oracle auf dem Linux System des Scitos installiert. Das Benutzer Passwort zum Zugriff auf Windows lautet: scitos. Die Software kann unter: `https://www.virtualbox.org/wiki/Downloads` heruntergeladen werden.

Nach aktuellem Stand vom 08.01.2014 funktioniert die Anbindung der Asus Xtion Pro und die FaceShift Software. Allerdings ist die virtuelle Maschine mit der Verarbeitung der Daten überlastet. Die Bilderkennung stockt und es werden maximal 3fps erreicht.

Allerdings können aufgezeichnete Videos mit ca. 10fps abgespielt und gestreamt werden.

3.2 Programmierung des Netzwerk-Clients

Um die Daten aus FaceShift zu empfangen wird auf dem Scitos ein UDP-Netzwerk-Client programmiert. Der Programmcode ist im Anhang A hinterlegt. Das Anwendungsprogramm wird alle 250ms aufgeführt. Dies hat dazu geführt das von FaceShift zeitverzögert alte Daten abgerufen wurden.
Um das Problem zu beheben wird der UDP-Socket nach Empfang der Daten wieder geschlossen und wird bei jedem Programmzyklus neu aufgerufen. Durch diese Änderung wurde erreicht, dass immer aktuelle Positionsdaten des Kopfes empfangen werden.

3.3 Anlegen einer Mira Komponente

Das Anlegen eines neuen Mira Projekts wird an Hand der offiziellen Anleitung durchgeführt. Diese kann unter:
`http://www.mira-project.org/MIRA-doc-devel/TutorialCreatePublisherUnitPage.`

`html` gefunden werden.

Um eine neue Komponente anzulegen wird im Terminal der Mirawizard mit dem Befehl „mirawizard“ aufgerufen. Daraufhin sollte das in Abbildung 3.1 dargestellte Fenster erscheinen. Im Wizard wird nun „Create a new External Project“ ausgewählt.

Abbildung 3.1: Mira Wizard - New External Project

Im Folgenden Menü wird der gewünschte Projektname und der Projektpfad angegeben. Im nächsten Fenster muss ein Häckchen bei „Set environment variables automatically“ gesetzt werden. Zuletzt muss das Häckchen bei „Perform the following steps afterwards“ entfernt werden.

Als nächstes wird mit dem Wizard eine neue Komponente mit „Create a New Component“ erstellt (Abbildung 3.2

Im nächsten Fenster, welches in Abbildung 3.3 dargestellt ist, werden Projektname, Namespace und der Pfad angegeben, in dem das neue Projekt angelegt werden soll.

Bei dem Projekt „FollowHeadPose“ wurden folgende Eingaben getätigt:

Name: FollowHeadPose
Namespace: mira
Source Path: /MIRA-faceshift/domains/behaviours/FollowHeadPose

Sobald der Vorgang mit dem „Commit“ Button abgeschlossen ist, enthält der Source Path ein vorgefertigtes C-File (FollowHeadPose.c), das mit User Code befüllt werden kann.

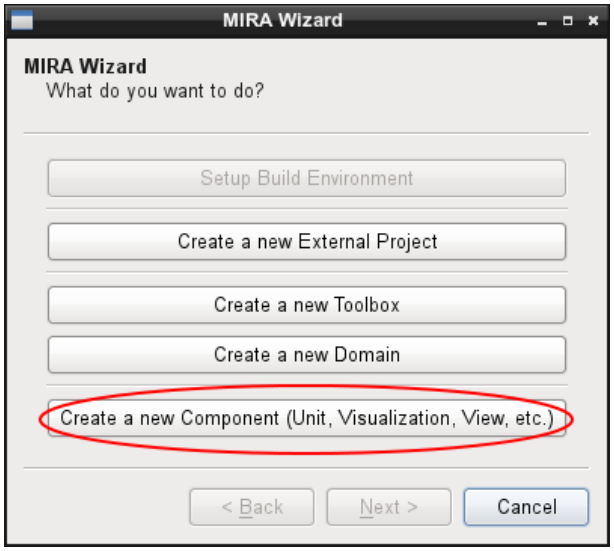

Abbildung 3.2: Mira Wizard - New Component

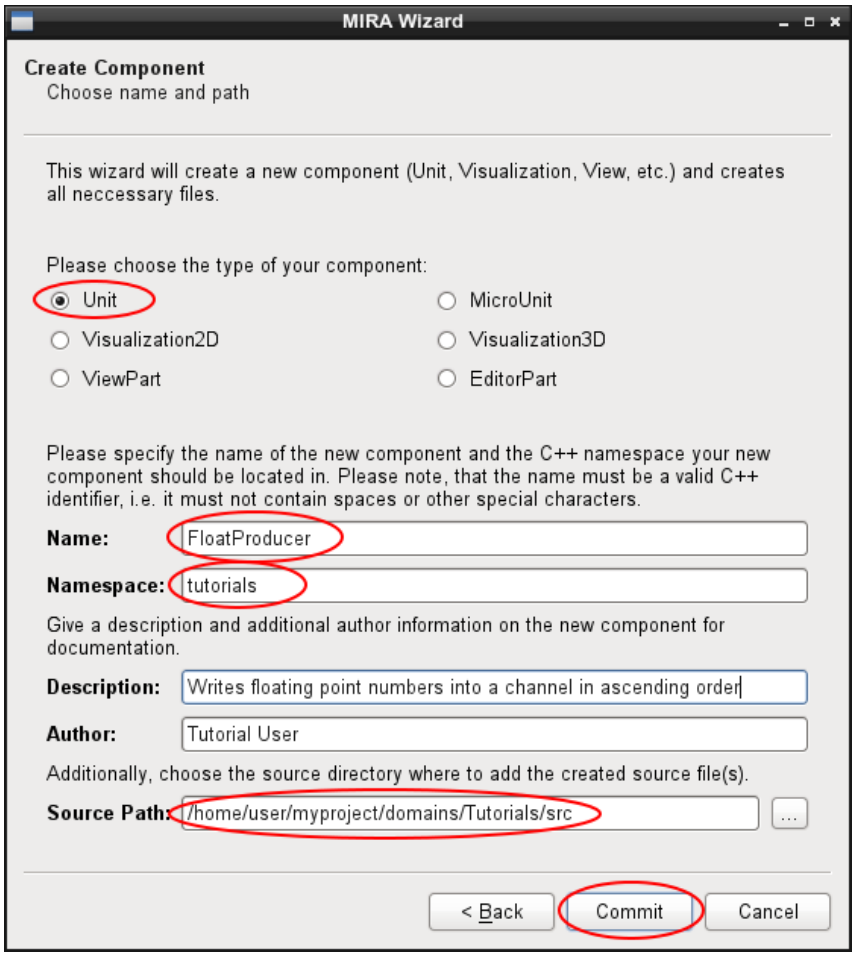

Abbildung 3.3: Mira Wizard anlegen einer Unit

Gleichzeitig wurde über dem neu erstellten Komponenten Ordner eine Datei mit dem Namen „CMakeLists.txt“ angelegt. In dieser müssen die eventuell benötigten Abhängigkeiten

von weitern C/C++ Files angeben werden.

Bevor eine neu erstellte Komponente das erste Mal compilliert werden kann, muss die Path-Variable neu eingelesen werden. Dies wir am einfachsten über das Ab- und Neuanmelden des Benutzers in Linux erreicht.

3.4 Anwendungsprogramm „FollowHeadPose“

Das Anwendungsprogramm FollowHeadPose (Anhang A) befindet sich im Pfad: `/MIRA-faceshift/domains/behaviours/FollowHeadPose`.

Im Aufruf des Hauptprogramms kann die Zykluszeit eingestellt werden, in der das Programm aufgefrufen wird. Um eine schnelle Reaktion auf Kopfbewegungen sicherzustellen, sollten hier 250ms eingestellt sein.

„`FollowHeadPose::FollowHeadPose() : Unit(Duration::milliseconds(250))`“

Am Anfang des Programmes wird ein UDP-Socket geöffnet und die Nachrichten von FaceShift werden per UDP-Client empfangen. Danach wird der UDP-Socket wieder geschlossen. Aus diesen Daten wird die YAW-Position des Kopfes mit dem Befehl `data.m_headRotation.y` ausgelesen. Des Weiteren wird mit dem Befehl `data.m_trackingSucessful` abgefragt, ob sich aktuell eine Person im Sichtfeld der Kamera befindet und getrackt wird. Sollte dies nicht der Fall sein, stellt sich der Roboterkopf auf die Positon 0°. Wird eine Person gefunden, imitiert der Roboterkopf die Blickrichtung der getrackten Person.

Die awareness recognition wird über die Augen des Scitos realisiert. Erkennt dieser keine Person, bleiben die Augen geschlossen. Sobald sich eine Person im Sichtfeld des Scitos befindet öffnen sich die Augen.

```
callService<void>("/robot/Robot", "moveEyeLidUpDown",eye_select,eye_pitch);
```

Die Variable „`eye_select`“ wird in initialize() definiert und kann mit den Werten:

0 = Linkes Auge

1 = Rechtes Auge

2 = Beide Augen

belegt werden.

Mit der Variable „`eye_pitch`“ wird der Öffnungswinkel des Augenlieds eingestellt. Aktuell sind die Werte:

40 = Augenlieder geschlossen

100 = Augenlieder offen

eingestellt.

Sollte die Verbindung zwischen dem FaceShift Server und dem Anwenderprogramm FollowHeadPose nicht funktionieren, muss die IP-Adresse in FaceShift und in FollowHeadPose.c auf Gleichheit überprüft werden. Es muss die IP-Adresse des Scitos (Linux-PC) eingetragen sein.

FollowHeadPose.c: `saddr.sin_addr.s_addr = inet_addr("192.168.1.31");`

Falls Änderungen an FollowHeadPose.c durchgeführt werden, muss das Anwenderprogramm neu kompiliert werden. Hierfür muss ein Terminal geöffnet werden. In diesem muss der Projektordner aufgerufen werden. Zum kompilieren wird der Befehl „make“ eingegeben.

4 Bedienerhandbuch

Im folgenden wird Schrittweise erklärt wie das System gestartet wird. Es stehen hierbei zwei Alternativen zur Auswahl:

- Streaming von einem beliebigen PC,
- Abspielen eines aufgezeichneten Streams von der VirtualBox.

4.1 Aktivieren/Deaktivieren von FollowHeadPose im Mira Center

Das Anwenderprogramm FollowHeadPose kann im Mira Center aktiviert/deaktiviert werden. Hierfür muss im Reiter „Authorities“ FollowHeadPose ausgewählt werden. Nun wird im Porperties|Authorities| Reiter die Property „Activate Movements“ angezeigt. Activate Movements ist Standardmäßig auf „1“ eingestellt und FollowHeadPose ist aktiviert. Wird eine „0“ eingetragen ist FollowHeadPose deaktiviert (Abbildung: 4.1).

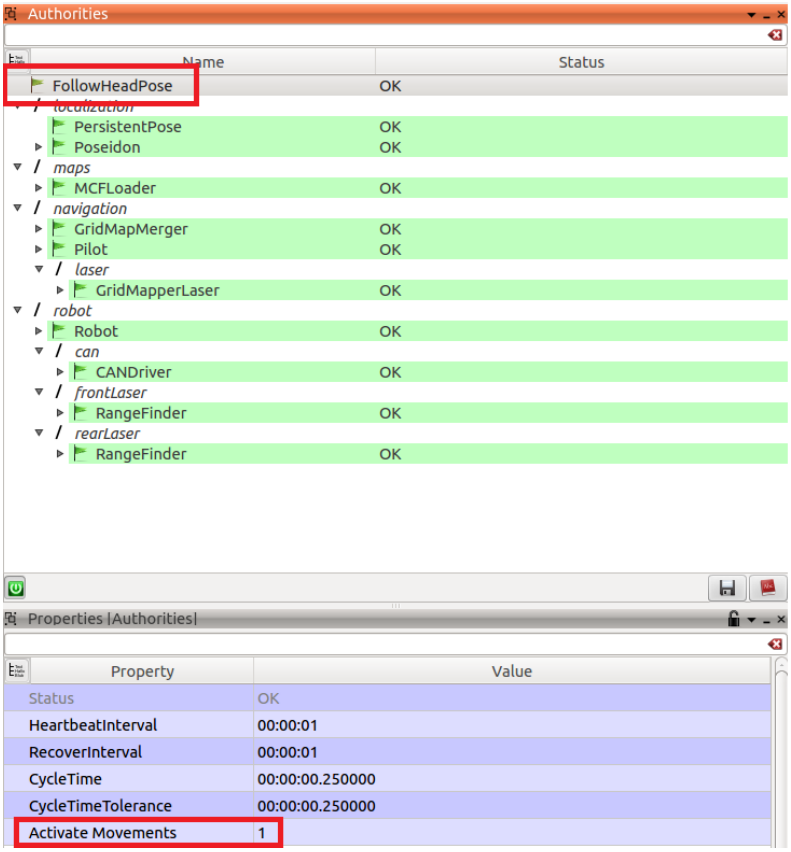

Abbildung 4.1: Aktivieren/Deaktivieren von FollowHeadPose

4.2 Streaming von einem beliebigen PC

1. SCITOS starten,
2. Im Bootmanager muss die ausgewählte Partition eingestellt bleiben,
3. FaceShift auf PC Installieren und starten,
4. Netzwerk-Stream in FaceShift konfigurieren und starten,
5. Terminal aufrufen und den Befehl: „miracenter /opt/MIRA-commercial/domains/ robot/SCITOSConfigs/etc/ SCITOS-Pilot.xml /home/demo/MIRA-faceshift/domains/ behaviours/FollowHeadPose/ FollowHeadPose.xml –variables MCFFile=/home/ demo/etage1/etage1.mcf“ eingeben.
   Der Befehl ist in der Datei „start_JeThi“ hinterlegt,
6. Jetzt sollte das Mira-Center starten und das Anwenderprogramm ausgeführt werden.

4.3 Abspielen eines aufgezeichneten Streams von der VirtualBox

1. SCITOS starten,
2. Im Bootmanager muss die ausgewählte Partition eingestellt bleiben,
3. Windows VirtualBox starten,
4. FaceShift in der VirtualBox starten (PW: scitos),
5. Demo Videostream
   `C:\Users\lions\Documents\faceshift\profiles\Demo_LookAtMe.fsp` laden,
6. Terminal aufrufen und den Befehl: „miracenter /opt/MIRA-commercial/domains/ robot/SCITOSConfigs/etc/SCITOS- Pilot.xml /home/demo/MIRA-faceshift/domains/ behaviours/FollowHeadPose/FollowHeadPose.xml –variables MCFFile=/home/ demo/etage1/etage1.mcf“ eingeben.
   Der Befehl ist in der Datei „start_JeThi“ hinterlegt,
7. Jetzt sollte das Mira-Center starten und das Anwenderprogramm ausgeführt werden.

4.4 Fehlerbehandlung:

1. In FaceShift unter Rubrik Tracking im Scrollfeld zu Network scrollen. Hier Häckchen bei Network Streaming Ein/Ausschalten,

2. Sollte im Terminal mit dem das Mira-Center gestartet wurde die Meldung Binding Failed auftreten liegt ein IP-Adressen Konflikt vor,

3. IP-Adresse für Netzwerk-Stream kontrollieren,

4. IP-Adresse von Linux muss eingetragen sein: Terminal -> ifconfig,

5. PC und SCITOS müssen sich im gleichen Netzwerk befinden.

Literatur

[1] Mira Reference Documentation
`http://www.mira-project.org/MIRA-doc-devel/index.html`
Zuletzt gesehen: 05.01.2014

[2] FaceShift Support
`http://www.faceshift.com/support/`
Zuletzt gesehen: 05.01.2014

Anhang

A FolloHeadPose C-Code

```
/*
 * Copyright (C) 2012 by
 *   MetraLabs GmbH (MLAB), GERMANY
 * and
 *   Neuroinformatics and Cognitive Robotics Labs (NICR) at TU Ilmenau, GERMANY
 * All rights reserved.
 *
 * Contact: info@mira-project.org
 *
 * Commercial Usage:
 *   Licensees holding valid commercial licenses may use this file in
 *   accordance with the commercial license agreement provided with the
 *   software or, alternatively, in accordance with the terms contained in
 *   a written agreement between you and MLAB or NICR.
 *
 * GNU General Public License Usage:
 *   Alternatively, this file may be used under the terms of the GNU
 *   General Public License version 3.0 as published by the Free Software
 *   Foundation and appearing in the file LICENSE.GPL3 included in the
 *   packaging of this file. Please review the following information to
 *   ensure the GNU General Public License version 3.0 requirements will be
 *   met: http://www.gnu.org/copyleft/gpl.html.
 *   Alternatively you may (at your option) use any later version of the GNU
 *   General Public License if such license has been publicly approved by
 *   MLAB and NICR (or its successors, if any).
 *
 * IN NO EVENT SHALL "MLAB" OR "NICR" BE LIABLE TO ANY PARTY FOR DIRECT,
 * INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF
 * THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF "MLAB" OR
 * "NICR" HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * "MLAB" AND "NICR" SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING,
 * BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND "MLAB" AND "NICR" HAVE NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS OR MODIFICATIONS.
 */

/**
 * @file FollowHeadPose.C
 *
 *
 * @author Wagner, Jens; Frank, Thiemo
 * @date   2014/01/14
 */

#include <fw/Unit.h>
//NetworkClient
#include <sys/types.h>    // für socket()
#include <sys/socket.h>   // für socket()
#include <netinet/in.h>   // für socket()
#include <assert.h>       // für assert()
#include <netdb.h>        // für getprotobyname()
#include <unistd.h>       // für close()
#include <arpa/inet.h>    //für inet_ntop()
#include <netdb.h>        //für getaddrinfo()
#include <string.h>         // für memset()
```

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <fw/fsbinarystream.h>
#include<stdlib.h> //exit(0);

#define HAVE_EIGEN
namespace mira {

//////////////////////////////////////////////////////////////////////////////

/**
 *
 */
class FollowHeadPose : public Unit
{
MIRA_OBJECT(FollowHeadPose)

public:

    FollowHeadPose();

    template<typename Reflector>
    void reflect(Reflector& r)
    {
        Unit::reflect(r);
        r.property("Activate Movements", runTrack, "", 1); // publish activate movements
        option in miracenter
    }

protected:

    virtual void initialize();
    virtual void process(const Timer& timer);
    virtual void destruct();

private:

    float HeadPoseYaw;
    float HeadPosePitch;
    float eye_pitch;
    unsigned char eye_select; // define which eye is used.
    //NetworkClient
    int sock, status;
        unsigned int socklen;
        char buffer[1000];
        struct sockaddr_in saddr;
        struct ip_mreq imreq;
    int faceLost;
    fs::fsBinaryStream parserIn, parserOut;
    fs::fsMsgPtr msg;
    int runTrack; //activate movements

};

//////////////////////////////////////////////////////////////////////////////
```

```
FollowHeadPose::FollowHeadPose() : Unit(Duration::milliseconds(250)) //process duration
time: 250 ms
{
}

void FollowHeadPose::initialize()
{

    faceLost = 0;
    eye_select = 2; //(0->left;1->right;2->both eyes)
    eye_pitch = 0.0;

}

void FollowHeadPose::process(const Timer& timer)
{

    if (runTrack < 1) { //interrupt process if no movement is selected.
       return;
    }

    //NetworkClient
    // set content of struct saddr and imreq to zero
    memset(&saddr, 0, sizeof(struct sockaddr_in));
    memset(&imreq, 0, sizeof(struct ip_mreq));

    // open a UDP socket
    sock = socket(PF_INET, SOCK_DGRAM, 0);
    if (sock < 0) {
        perror("socket failed!");
    printf("opening socket failed.");
    return;
    }

    saddr.sin_family = PF_INET;
    saddr.sin_port = htons(33433); //change UDP port here, if necessary
    saddr.sin_addr.s_addr = inet_addr("192.168.1.31"); //change IP address here, if necessary
    status = bind(sock, (struct sockaddr *)&saddr, sizeof(struct sockaddr_in));
    if (status < 0) {
        perror("bind failed!");
        printf("Binding ip address failed. Is the correct one selected?");
    return;
    }

        socklen = sizeof(saddr);
        status = recvfrom(sock, buffer, sizeof(buffer), 0, (struct sockaddr *)&saddr, &
        socklen);
        close(sock);
        if (status < 0) {
            printf("recvfrom failed!\n");
        }

        buffer[status] = '\0';

        if ( status > 0 ){
                                parserIn.received(status, buffer);
                                while(msg=parserIn.get_message())
```

```
                {
                    if(dynamic_cast<fs::fsMsgTrackingState*>(msg.get()))
                    {
                        fs::fsMsgTrackingState *ts = dynamic_cast<fs::
                        fsMsgTrackingState*>(msg.get());
                        const fs::fsTrackingData &data = ts->tracking_data();

                    //activate both printf's to see all availiable data
                    //printf ("head translation: %f %f
                    %f\n",data.m_headTranslation.x,data.m_headTranslation.y,da
                    ta.m_headTranslation.z);
                    //printf ("head rotation: %f %f %f
                    %f\n",data.m_headRotation.w,data.m_headRotation.x,data.m_h
                    eadRotation.y,data.m_headRotation.z);
                    if(data.m_trackingSuccessful == 1){
                        HeadPoseYaw = data.m_headRotation.y*-100;
                        eye_pitch = 100.0;

                    }else {

                        if (faceLost > 5){ //when there is no face detected
                        for more than 6 iterations, set face as lost.
                            HeadPoseYaw = 0; //turn head straight
                            faceLost = 0;
                            eye_pitch = 40.0; //close eyes

                        } else {
                            faceLost++;
                        }
                    }
                    }
                }
            }
            if(!parserIn.valid()) {
                printf("parser in invalid state\n");
                parserIn.clear();
            }

        callService<void>("/robot/Robot", "moveHeadLeftRight",HeadPoseYaw);
        callService<void>("/robot/Robot", "moveEyeLidUpDown",eye_select,eye_pitch);

    printf("eye_pitch: %f\n",eye_pitch);
}

void FollowHeadPose::destruct()
{
    close(sock);
}

////////////////////////////////////////////////////////////////////////////////

}
MIRA_CLASS_SERIALIZATION(mira::FollowHeadPose, mira::Unit );
```