

# Der INSECT Roboter

Elias Hohl      Maximillian Philipp

25. April 2019

## Zusammenfassung

Der Name **INSECT** steht für **I**mmense **N**ew **S**ix-legged **E**xternally **C**ontrolled **T**elerobot. Hierbei handelt es sich um einen von uns gebauten und programmierten sechsbeinigen Roboter von in etwa zwei Metern Länge und einem Gewicht von circa 60 Kilogramm, welcher außer auf seinen Beinen gehen zu können auch einige Zusatzfunktionen aufweist:

- Fernkontrolle aller Funktionen über PHP-Interface
- 8 Megapixel Kamera-Livestream
- Audio-Livestream von Mikrofon
- Abspielen von im Browser aufgenommenen Audio
- Abspielen von inkludierten Audiodateien auf Befehl
- Sprachsteuerung von manchen Funktionen
- Selbst sprechen

Im folgenden werden wir sowohl den mechanischen Aufbau und die Funktionen des Roboters als auch die Computersteuerung weiter erläutern.

## Teil I Aufbau und Funktionen

### 1 Motivation

Die Robotik wird immer wichtiger in der heutigen Welt. Die meisten mobilen Roboter sind mit Rädern oder Raupen ausgestattet. Die Verwendung letzterer hat definitiv konstruktive Vorteile bzw. einen geringeren Konstruktionsaufwand. Allerdings werden in schwer begehbaren Gebieten auch Roboter mit Füßen, meistens sechs, eingesetzt (diese werden dann "Hexapod Robots" genannt). Solche Maschinen sind allerdings meistens extrem teuer und werden hauptsächlich vom Militär verwendet. Mit Materialkosten von nur 1000 Euro kann der INSECT-Roboter auch als etwas größeres Projekt gebaut werden. Warum solche Abmessungen? Ich halte es einfach für sinnvoller, einen Roboter in einer anständigen

Abbildung 1: Der INSECT-Roboter



Größe zu bauen, damit dieser dann auch etwas tun kann, anstatt immer aufpassen zu müssen, dass man auf einen Faustgroßen Roboter, der am Boden herumkrabbelt, nicht draufsteigt. INSECT kann zumindest leichte Trage- und Zieharbeiten verrichten, auch wenn er sich nur langsam fortbewegen kann. Außerdem stößt man bei der Konstruktion auch auf andere Probleme, die man zu lösen lernt, zum Beispiel, wenn eine Verstrebung verstärkt werden muss. Oder der Transport. INSECT wurde ursprünglich in einer Werkstatt im Keller zuhause gebaut, der Transfer in die Schule erfolgte mit dem Nachbarn (Vielen Dank an Thomas Amegah) um sechs Uhr in der Früh auf dem Radweg umgedreht auf einem Skateboard die sechs Kilometer lange Strecke entlang, weil der Roboter in kein normales Auto passt.

## 2 Abmessungen

Eines der beeindruckendsten Merkmale von INSECT ist definitiv seine Größe. Der Roboter ist 175cm lang, die breiteste Stelle (die mittleren Füße) ist 95cm lang, und die höchste Stelle ist 75cm vom Boden entfernt. INSECT hat ein Gesamtgewicht von in etwa 60kg.

## 3 Konstruktionsgrundlagen

INSECT besteht aus drei Segmenten, wovon das längste, mittlere aus Stahl und die anderen beiden aus Aluminium hergestellt sind. Insgesamt wurden zum Bau des Roboters eine Gesamtlänge von 80 Metern Stahl- und Aluprofilen benötigt. Die Segmente sind über motorgesteuerte (allerdings noch nicht per Computer kontrollierbare) Gelenke verbunden. Die Gelenke erlauben sowohl Auf- und Abwärtsbewegungen, als auch Bewegungen nach links und rechts. Jedes Segment ist mit je zwei Beinen und je einem Hebemotor ausgestattet, welcher immer abwechselnd die Beine links und rechts anhebt beziehungsweise auf den Boden setzt. Um die enormen Kräfte bei der Vorwärtsbewegung auszuhalten, wurden erstens SSchuhe bestehend aus Gummischläuchen an den Beinen befestigt (auch um den Boden nicht zu beschädigen) und zwei "Krücken" unter dem vorderen und hinteren Segment platziert, damit die Drehgelenksverbindungen zum mittleren Segment beim Gehen nicht nachgeben können. Die Maschine ist mit sechs Bleiakkus zu je 4,5 Amperestunden ausgerüstet. Vier davon betreiben in Parallelschaltung die Motoren, die anderen beiden dienen als getrennte Stromversorgungen für Lautsprecher und den Raspberry Pi, die zentrale Steuereinheit des Projektes. Der Roboter benötigt eine Leistung von in etwa 100 Watt, um sich zu bewegen. Außerdem verfügt INSECT über eine Kamera und ein Mikrofon, die am vorderen Ende montiert wurden. Die Kamera wird von einem Infrarotscheinwerfer unterstützt, um auch Nachtaufnahmen machen zu können.

## 4 Die Füße

Die sechs Füße von INSECT sind links und rechts auf allen drei Segmenten montiert, pro Segment drei. Die zwei Füße eines Segments sind immer über ein Gelenk mit Hebemotor gekoppelt. Die Füße selbst sind aus Aluminium konstruiert, das stellenweise mit Stahl verstärkt wurde. Vereinzelt wurden auch Bauteile aus Stahl, Edelstahl und Messing verwendet. Die zwei Endtaster geben dem Motor das Stopp-Signal, wenn der Fuß sich ganz nach hinten beziehungsweise ganz nach vorne bewegt hat. Eine sehr starke Feder (einige Newton pro Millimeter) sorgt zusätzlich für eine größere wirkende Leistung. Während der Fuß sich in der Luft befindet, wird sie gespannt. Danach, wenn dieser wieder auf den Boden aufgesetzt hat, zieht sie den Fuß in die Rückwärtsrichtung (damit sich der Roboter nach vorne bewegt). Dies verhindert zwar, dass INSECT sich rückwärts bewegen kann, allerdings müsste die Gesamtleistung der verwendeten

Abbildung 2: Die Akkubank



Motoren ansonsten in etwa um den Faktor 1,5 höher sein. Für mehr Informationen, wie die Füße angesteuert werden, siehe "Der Motorcontrol-Thread unter [Server.cünter Computersteuerung](#)".

## 5 Die Lautsprecher

Es wurden zwei billige Lautsprecher (Kosten ca. 5 Euro) und eine Verstärkerplatine von Daypower verwendet. Die Lautsprecher lassen sich mit einer Leistung von je 100 Watt betreiben, der Verstärker liefert allerdings nur 30 Watt Gesamtleistung (was aber definitiv ausreicht, um einen 40 Meter langen Gang zum Dröhnen zu bekommen, siehe Video). Standardmäßig einprogrammiert ist das Abspielen der österreichischen Bundeshymne, "Helter Skelter" von den Beatles, der Prolog im Himmeläus Goethes Faust, und "Daedalus und Ikarus" (auf Latein) von Ovid. Diese Kommandos lassen sich entweder über das Webinterface oder per Sprachbefehl triggern.

Abbildung 3: Ein Fuß

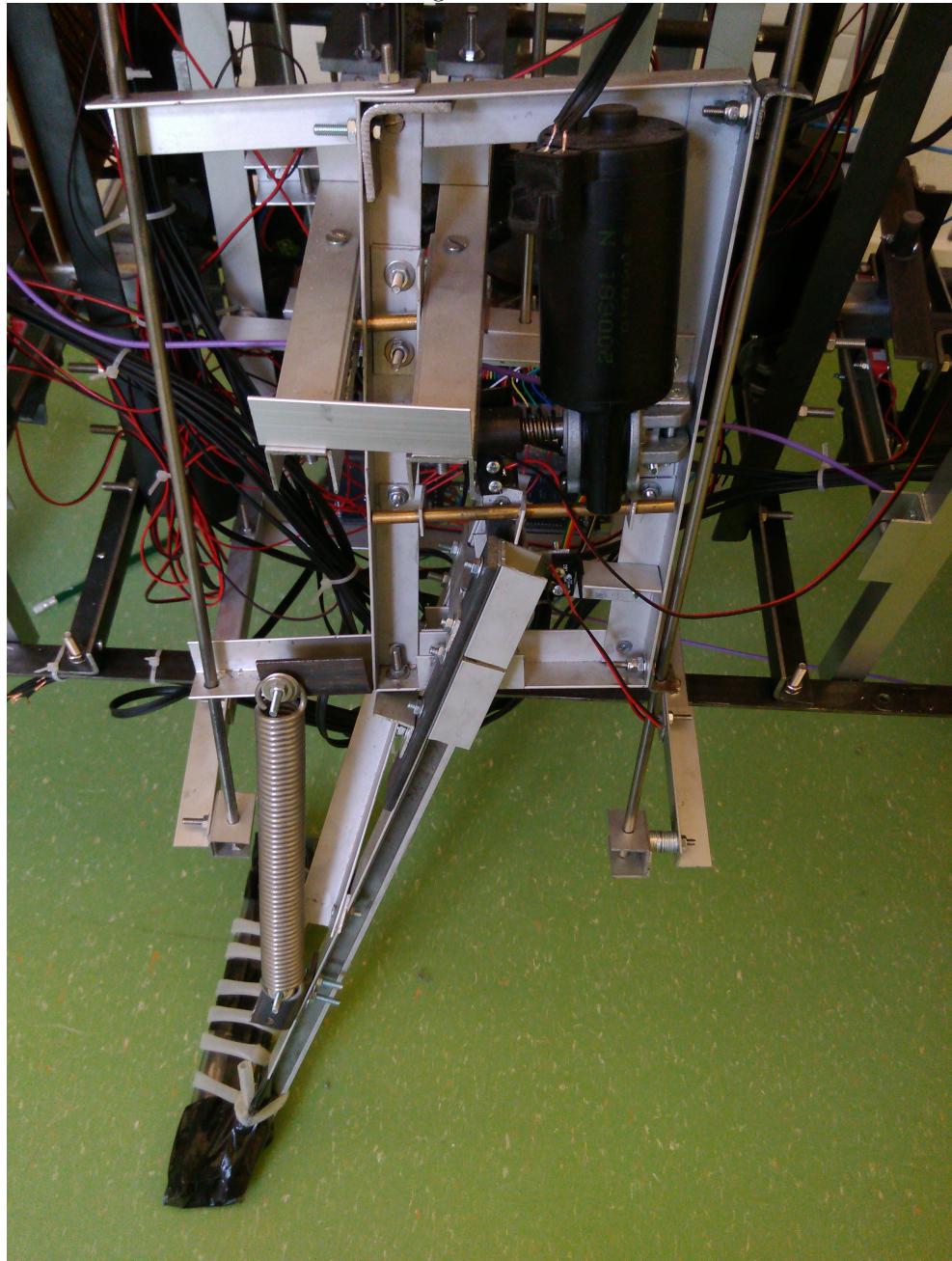
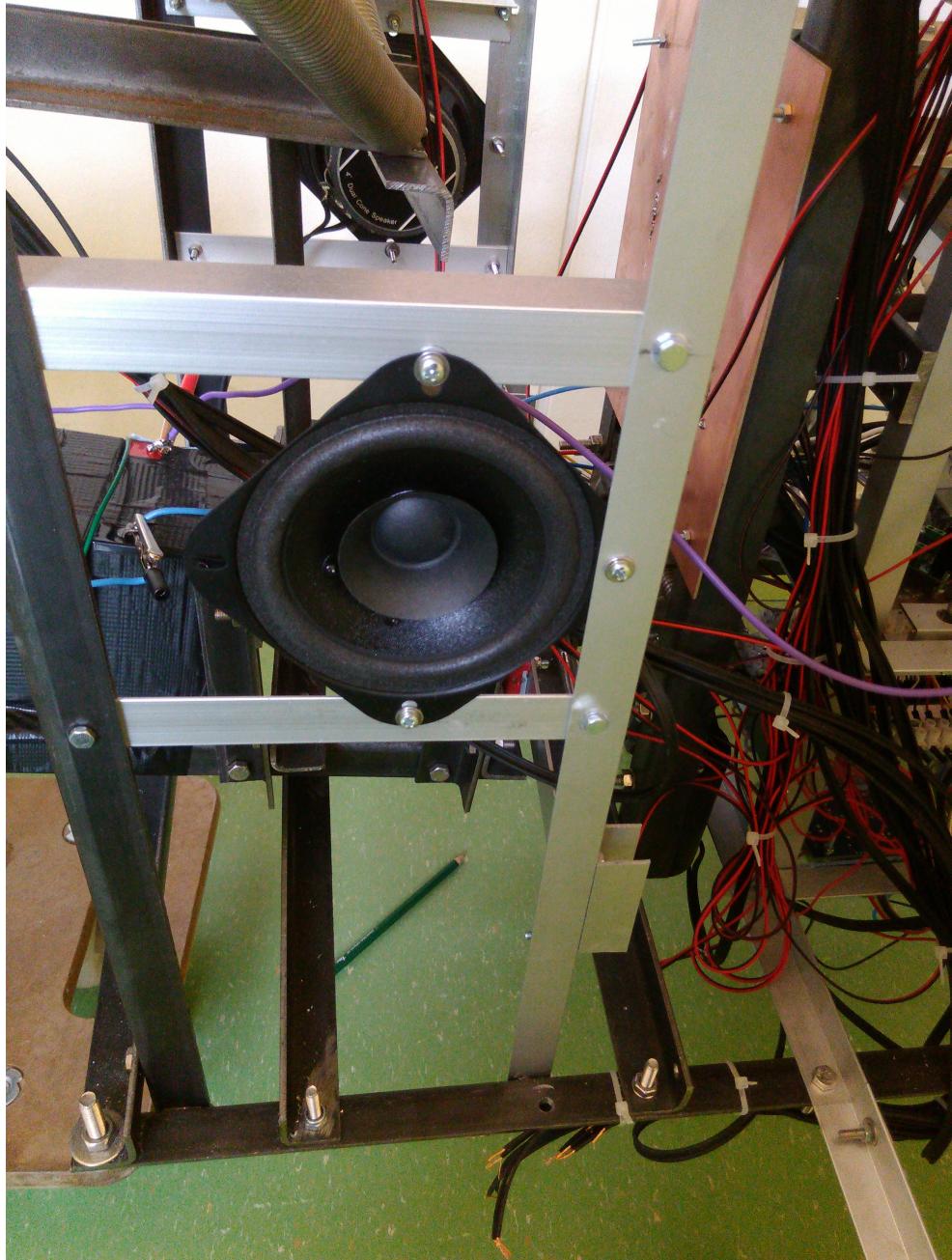


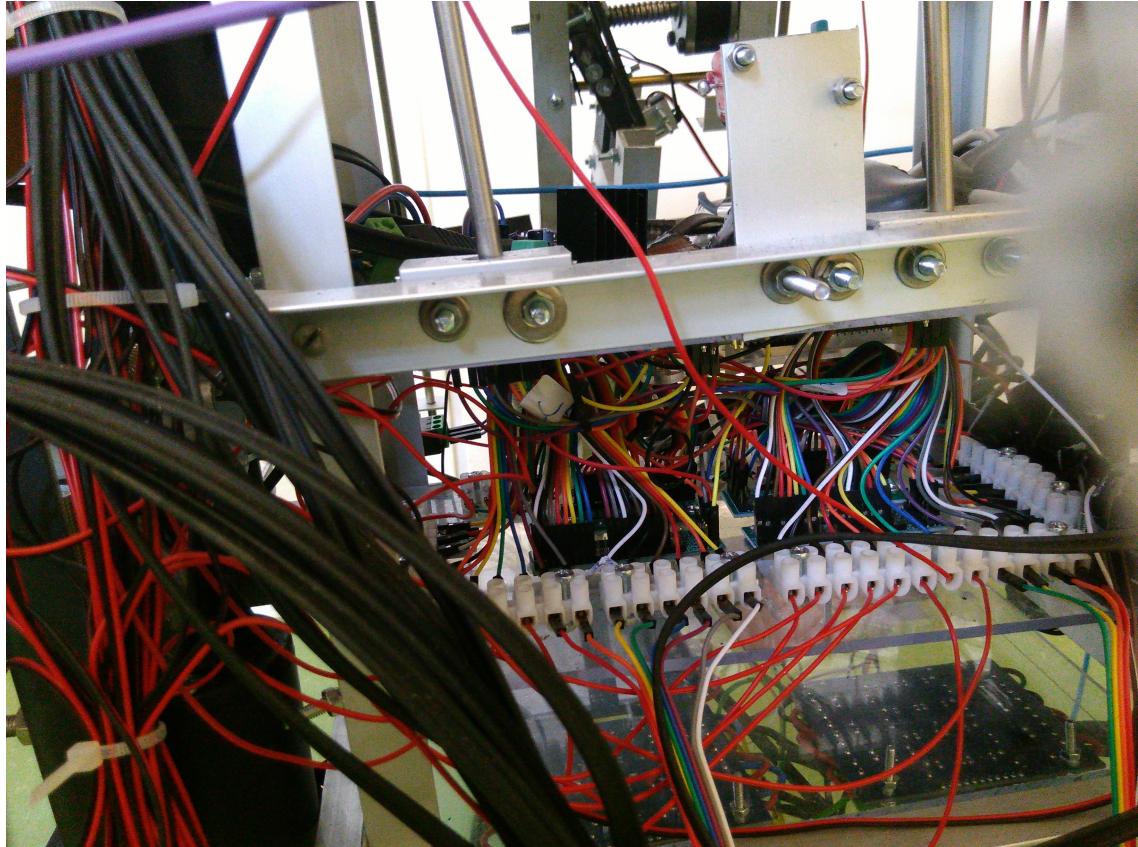
Abbildung 4: Einer der beiden 100W-Lautsprecher



## 6 Elektronische Schaltung

Die Kernelemente der Steuerung von INSECT sind:

Abbildung 5: Der Kabelsalat im Zentralnervensystem



- 1 Raspberry Pi 3B
- 4 Platinen mit dem Chip MCP23017
- 3 8-Kanal Relaiskarten

Die MCP23017-Platinen werden über die I2C-Schnittstelle des Raspberry Pi angesteuert. Diese fungieren als Porterweiterung (je 16 zusätzliche digitale Pins) weil der Raspberry Pi selbst zu wenige Pins hätte, um alle 17 Motoren und 34 Taster anzusteuern bzw. auszulesen. Die Taster sind so verbunden, dass der entsprechende Eingang des Chips bei einem Tastendruck auf Masse gesetzt wird. Ansonsten sind sie durch den internen 10 Kiloohm-Widerstand mit +3V3 verbunden. Die Ausgänge sind direkt mit den Relaiskarten verbunden. Wird der Ausgang des MCP23017 auf Masse gelegt, schaltet das Relais um. Immer je zwei Relais sind einem Motor zugeordnet. Die beiden festen Kontakte der Relais sind immer jeweils mit Masse oder +12V Betriebsspannung verbunden. Anfänglich liegen also beide Anschlüsse des (Gleichstrom-)Motors auf Masse. Wird nun ein

Relais eingeschalten, bewegt sich der mittlere Kontakt von der Masse zu Plus, einer der Kontakte des Motors liegt jetzt auf Plus und der Motor dreht sich. Wird das andere Relais eingeschalten, dreht sich der Motor in die andere Richtung. Das eingefüget Bild gibt einen guten Eindruck, wie es in der Steuerzentrale (ich nenne sie gerne SZentralnervensystem") von INSECT aussieht. Die dicken schwarzen Kabel führen zu den Motoren und Lautsprechern, die dünnen rot-schwarzen zu den Tastern, und die bunten verbinden die MCP23017-Platinen mit den SDA und SCL Ports der I2C-Schnittstelle des Raspberry Pi und der Spannungsversorgung.

## Teil II

# Computersteuerung

## 7 Grundsätzliche Organisation

Das Projekt ist aus mehreren Teilen aufgebaut, die miteinander kommunizieren. Es wurden folgende Programmiersprachen verwendet:

- C++
- PHP
- Javascript
- Bash
- Python

Die Abbildung 1 zeigt eine Übersicht über die Kontrollebenen des Projekts. Im folgenden werden wir die einzelnen Programmelemente und ihre Funktionsweise anhand von Codesnippets weiter erläutern. Der ganze Code ist unter <https://github.com/ehtec/insect> verfügbar.

## 8 Inkludierte Software

Um ein effizientes Programm zu schreiben, ist es eine Notwendigkeit, die bereits entwickelten zur Verfügung stehenden Libraries und Programme zu kennen und zu nutzen. Vor allem hardwarenahe Aufgaben, z.B. Audiotreiber, oder Software, die extrem komplexe Aufgaben erledigt (Sprachsteuerung), oder Programme, bei denen ein möglichst schneller Code wichtig ist und deren Entwicklung viel Zeit zur Optimierung braucht, z.B. Livestreamer, sollten an bereits existente Projekte übergeben werden. So vermeidet man, jedes Mal das Rad neu zu erfinden. An dieser Stelle möchte ich mich bei den Entwicklern der folgenden Open-Source Software bedanken, die in mein Projekt inkludiert sind:

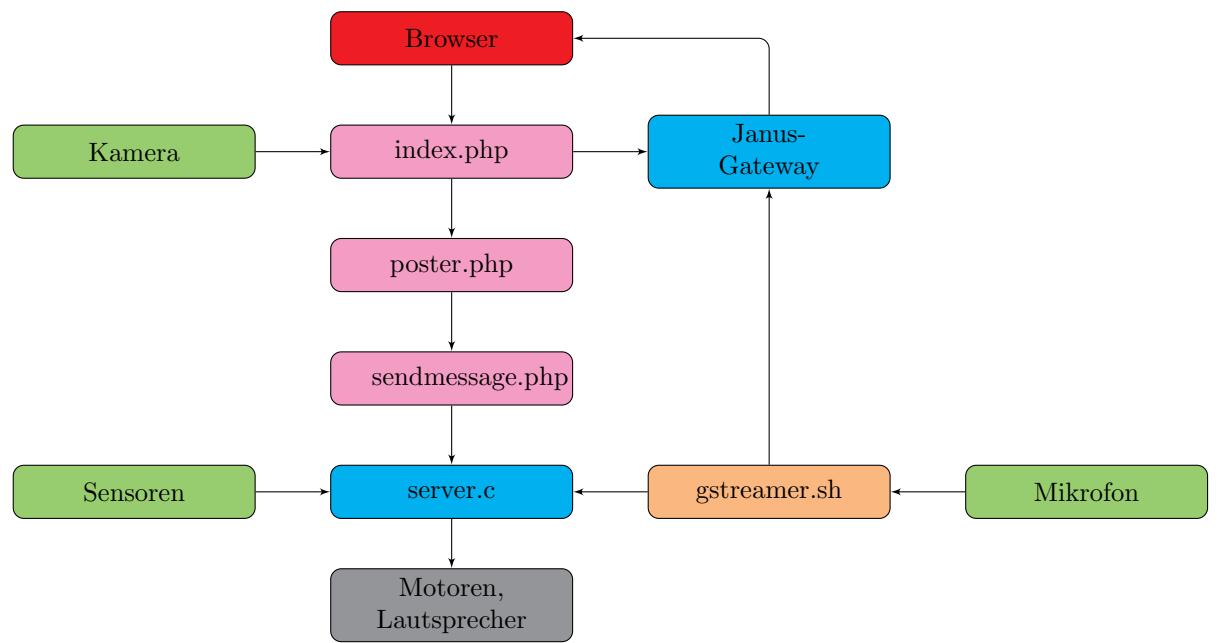
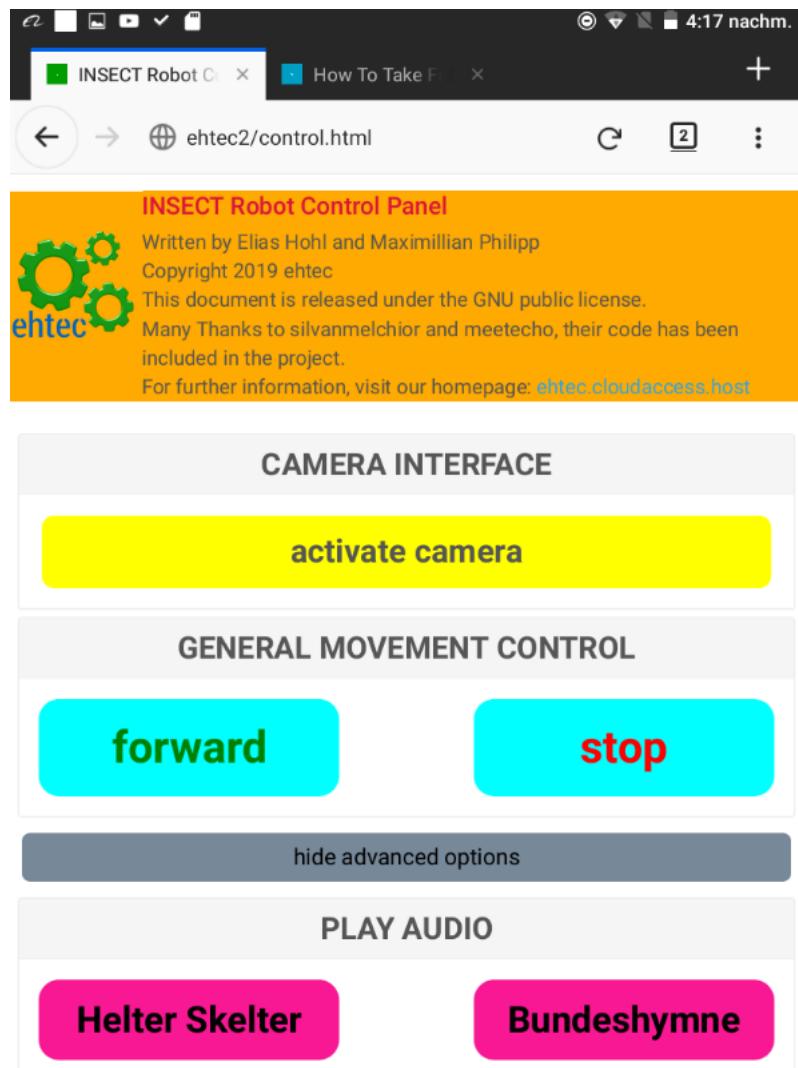


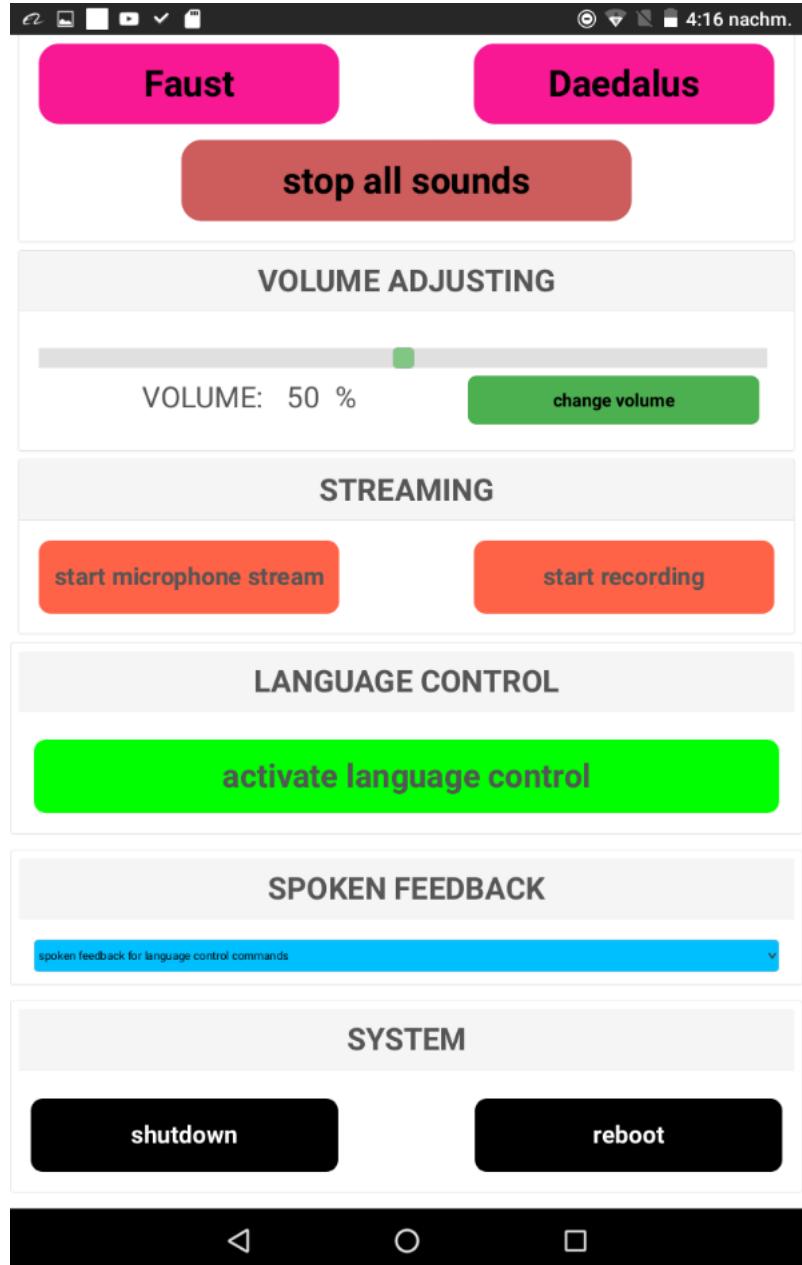
Abbildung 6: Organisation

- RPi\_Cam\_Web\_Interface von Silvanmelchior:  
[https://github.com/silvanmelchior/RPi\\_Cam\\_Web\\_Interface](https://github.com/silvanmelchior/RPi_Cam_Web_Interface) Die Implementation des Kamera-Livestreams wurde von hier übernommen.
- Janus-Gateway von Meetecho: <https://github.com/meetecho/janus-gateway> Die Implementation des Audio-Livestreams wurde von hier übernommen.
- CMUSphinx: <https://github.com/cmusphinx> Die Sprachsteuerung wird von Pocketsphinx übernommen.

## 9 PHP-Interface

index.php ist das 1000-zeilige, in PHP geschriebene, als HTML formatierte und mit javascript animierte Kontrollinterface. Das Layout ist für die Verwendung mit einem Tablet und Firefox optimiert. Der verwendete Webserver ist apache2. Der Code baut auf der gleichnamigen Datei von RPi\_Cam\_Web\_Interface auf, alle weiteren Features wurden hinzugefügt.





Das Script verfügt über einen unsichtbaren iframe namens poster

```
<iframe name="poster" style="display: none;"></iframe>
```

welche alle POST-Requests aufnimmt, um den Refresh der gesamten Seite bei einem Befehl zu verhindern. Gesendet werden die Requests von Knöpfen wie folgt:

```

<form action="poster.php" method="post" target="poster">
    <input class=button2 style="float: left; color: black;" 
           name="play=helterskelter" type="submit" value="Helter Skelter">
    <input class=button2 style="float: right; color: black;" 
           name="play=bundeshymne" type="submit" value="Bundeshymne">
    <br><input class=button2 style="float: left; color: black;" 
           name="play=fault" type="submit" value="Faust">
    <input class=button2 style="float: right; color: black;" 
           name="play=daedalus" type="submit" value="Daedalus">
    <br><center><input class=button2
        style="color: black; background-color: rgb(205, 92, 92); width: 60%;">
        name="stopsound" type="submit" value="stop all sounds"></center>
</form>

```

Dieses Snippet sendet auf Knopfdruck eine POST-Anfrage play=...”, wobei der in der name-Eigenschaft verankerte Sound ausgewählt wird. Gepostet wird mithilfe des iframes poster und auf die Datei poster.php:

```

<?php

require('sendmessage.php');

foreach($_POST as $key => $value){

    sendmessage("$key");

}

?>

```

wobei die in sendmessage.php deklarierte Funktion sendmessage sich mit dem von server.c erzeugten TCP-Socket verbindet und die Daten, in obigem Fall play=irgendeinsound, als Nachricht verschickt. Diese wird dann von server.c weiterverarbeitet. Ein Snippet aus sendmessage.php:

```

<?php
//...
if(!socket_send( $sock , $message , strlen($message) , 0))
{
    $errorcode = socket_last_error();
    $errmsg = socket_strerror($errorcode);

    die("Could not send data: [$errorcode] $errmsg \n");
}
//...
?>

```

## 10 app.js

app.js ist das Javascript, welches das Aufnehmen von Sound über das Control Panel und dessen Abspielen am Roboter ermöglicht. Nachdem der Sound mithilfe von getUserMedia() aufgenommen wurde, wird er auf den Server hochgeladen und abgespielt.

```
function createDownloadLink(blob) {

    //send everything
    var xhr=new XMLHttpRequest();
    xhr.onload=function(e) {
        if(this.readyState === 4) {
            console.log("Server returned: ",e.target.responseText);
        }
    };
    var fd=new FormData();
    fd.append("audio_data",blob, filename);
    xhr.open("POST","upload.php",true);
    xhr.send(fd);
    $.ajax({
        url: "poster.php",
        type: "POST",
        data: encodeURIComponent("play="+filename),
        success: function(data) {

    }
});
}
```

Das obige Snippet zeigt den Teil des Codes, der die Audiodatei mithilfe eines Skripts upload.php hochlädt, und dann per Ajax das play-Kommando (play=irgendeindateiname) per poster.php an server.c weitergibt.

## 11 Janus-Gateway

Das Janus-Gateway besteht aus zwei Teilen: Einem, der im Hintergrund läuft, den Mikrofon-Stream empfängt und in WebRTC-Medien umwandelt, und einem, der, geschrieben in Javascript, direkt vom Browser aufgerufen wird und die UDP-Pakete zurücksendet. Näheres zum Mikrofon-Stream erfahren Sie in der Section gstreamer.sh.

## 12 server.c

Das in C++ geschriebene Programm server.c ist eindeutig der komplexeste und wichtigste Teil der Software. In 1000 Zeilen werden der Server für den Empfang

der Kommandos vom Webinterface, die Steuerung der Bewegung aller Motoren, das Auslesen aller Sensoren, die Kontrolle der Lautsprecher und die Spracherkennung vereint.

## 12.1 Multithreading

server.c arbeitet mit mehreren Threads:

```
//managing sock thread
std::thread sock_thread (listen_for_msg);
puts("Socket thread started.");

//managing cmd thread
std::thread cmd_thread (check_for_msg);
puts("Cmd thread started.");

//managing gstdfree thread
std::thread gstdfree_thread (gstdfreeloop);
puts("Gstdfree thread started.");

//managing gstdwait thread
std::thread gstdwait_thread (gstdwaitloop);
puts("Gstdwait thread started.");

//managing motorcontrol thread
std::thread motorcontrol_thread (motorcontrolloop);
puts("Motorcontrol thread started.");

//managing language control thread
std::thread langctrl_thread (langctrlloop);
puts("Language control thread started.");

//speaking test line
if (::feedbacklevel > 0) {
    speakfeedback("Starting up system.");
}

//joining threads
sock_thread.join();
cmd_thread.join();
gstdfree_thread.join();
gstdwait_thread.join();
motorcontrol_thread.join();
langctrl_thread.join();0
```

- Der Socket Thread wartet auf Kommandos vom Webinterface. Wenn ein

Kommando empfangen wird, fügt er dieses in eine std::list namens cmdList hinzu.

- Der Cmd Thread überprüft ständig, ob cmdList Elemente, Kommandos, enthält. Wenn ja, wird das erste Kommando gelöscht und ausgeführt.
- Der Gstfree Thread wird benötigt, um abgespielte Sounds vorzeitig zu beenden.
- Der Gswait Thread überprüft, ob ein abgespielter Sound bereis zu Ende ist, und schaltet danach unter anderem die Sprachsteuerung wieder ein. Diese muss während dem Abspielen von Sounds abgeschalten werden, um eine Selbst-Triggerung zu vermeiden.
- Der Motorcontrol Thread bewegt und stoppt die Beine des Roboters.
- Der Language Control Thread wartet auf Sprachkommandos, und falls Sprachsteuerung aktiviert ist und die Kommandos existieren, der cmdList hinzu.

## 12.2 Der Socket

```
void listen_for_msg()
{
    //setup

    while (1) {
        unsigned int client = accept(server, (struct sockaddr *)
            &client_addr, &clientlen);

        // got a request, close the socket
        close(server);
        unlink(SOCKET_FILENAME);

        // read a request
        memset(buf, 0, buflen);
        nread = recv(client, buf, buflen, 0);

        printf("\nClient says: %s\n\n", buf);

        // echo back to the client
        send(client, buf, nread, 0);

        close(client);

        sleep(2);

        // re-bind and listen on the socket
    }
}
```

```

        bind_listen_socket(server, server_addr);

        ::cmdList.push_back(buf);
    }
}

```

Der Socket wird anfangs mit der Datei /tmp/server.sock erzeugt, dieser Name ist in der Konstante SOCKET\_FILENAME gespeichert. Wichtig ist dabei, dass der Apache-Webserver Schreibzugriff auf diese Datei hat, dazu ist PrivateTmp=False in apache.service zu setzen. Dieser Socket wartet nun auf eine Verbindung vom Webinterface, druckt die Nachricht zu Debug-Zwecken aus, und fügt das Kommando dann zur cmdList hinzu.

### 12.3 Der Cmd-Thread

```

void check_for_msg()
{
    while (1) {
        if (! ::cmdList.empty()) {
            std::string firstcmd = ::cmdList.front();
            ::cmdList.pop_front();
            std::list<string> result;
            boost::split(result, firstcmd, boost::is_any_of("="));
            std::string thekey = result.front();
            result.pop_front();
            if (! result.empty()) {
                thevalue = result.front();
            }
            //now checking commands
            if (thekey == "play") {
                //do something
            } else if (thekey == "camera") {
                //do something
            } else {
                //check for other commands
                puts("Invalid command.");
            }
        } else {
            sleep(0.1);
        }
    }
}

```

Der Cmd-Thread überprüft ständig, ob ein Kommando in der cmdList vorhanden ist. Wenn ja, wird dieses mit allen = Zeichen geteilt und in der Liste result gespeichert. Der erste Teil ist das Kommando, es wird in der Variable thekey gespeichert. Dann wird das erste Element aus result gelöscht und überprüft,

ob noch ein weiteres vorhanden ist. Wenn ja, wird dieses in der Variable thevalue gespeichert, diese speichert somit den Wert des Kommandos, zum Beispiel, welcher sound abgespielt werden soll. Dann wird mit if überprüft, welches Kommando thekey ist, und der entsprechende Block ausgeführt.

## 12.4 Der Motorcontrol-Thread

Die Motorsteuerung besteht aus einigen Teilen, die sich hauptsächlich mit der Ansteuerung und Initialisierung der Motoren beschäftigt. Zudem kümmert sich die Motorsteuerung um die korrekte Durchführung der Motorphasen. Die vier Phasen sind "SSchwenken Links", "Vorwärts Links", "SSchwenken Rechts" und "Vorwärts Rechts". Zwischen jeder Phase muss gewartet werden bis alle Beine des Roboters ihre Bewegung vollzogen haben, sonst würden sich die Beine im Boden verkeilen. Dadurch werden jegliche Fortbewegungsmöglichkeiten eingeschränkt.

### 12.4.1 Einbindung

Weil Python die beste Programmiersprache für Raspberry Pi Robotics ist, haben wir die Steuerung in Python geschrieben und den Code dann in dem C Programm eingebunden. Um Python in C einzubinden mussten wir Python.h für C werden. Wir haben den Python Interpreter für C initialisiert und dann PyRun verwendet um die Robot Klasse zu importieren und die Vorwärtsfunktion in dem Motorcontrol-Thread aufrufen zu können. Der Motorcontrol-Thread überprüft ständig, ob die globale Variable isgoing auf True gesetzt ist, und wenn ja, gibt er über Python den Befehl, einen Schritt zu machen.

```
void motorcontrolloop() {
    Py_Initialize();
    // importing and initializing the robot
    PyRun_SimpleString("from robot import Robot\n"
                       "r =Robot(4)\n"
                       "r.motorassignment(6,3,10)\n");
    while (TRUE) {
        // sleep to release the strain on the cpu
        sleep(0.05);
        // isgoing is a global variable set by the managing cmd thread
        while (::isgoing) {
            // run python function of the robot
            PyRun_SimpleString("r.forwards()\n");
        };
    };
}
```

#### 12.4.2 Initialisierung

Bei der Initialisierung werden alle Pins von den Erweiterungsboards (mcp23017) angesteuert um verschiedene Eigenschaften zu setzen. Zum Beispiel ob dieser Pin ein Output oder Input sein soll oder ob ein PullUp-Resistor eingeschalten werden soll. Nachdem die Eigenschaften gesetzt wurden, landen die Pins in einem von zwei Pools entweder bei den Inputs oder Outputs. Die Inputs und Outputs werden in Vierergruppen so zugeordnet, dass die zwei Inputs und zwei Outputs, der Gruppe, der Verkabelung entsprechen und einen Motor kontrollieren können.

```
# Setting up the extension boards to utilize the extra pins
wiringpi.wiringPiSetup()
for board in range(boardcount):
    gpin = setoff + ppins * board
    wiringpi.mcp23017Setup(gpin, 0x20 + board)
    for ppin in range(ppins):
        pin = ppin + gpin
        wiringpi.digitalWrite(pin, 0)
        # First half is set to input with pull-up
        if ppin < ppins / 2:
            wiringpi.pinMode(pin, 0)
            wiringpi.pullUpDnControl(pin, 2)
            self.inputs.append(pin)
        # Second half is set to output
        if ppin >= ppins / 2:
            wiringpi.digitalWrite(pin, 1)
            wiringpi.pinMode(pin, 1)
            self.outputs.append(pin)

#sort the pins correctly in grup
for x in zip(*[iter(self.inputs)]*8):
    rev = x[::-1]

    for tup in zip(*[iter(rev)] * 2):
        self.inpairs.append(tup)

    for x, y in zip(*[iter(self.outputs)] * 2):
        tup = (x, y)
        self.outpairs.append(tup)

    for inpair, outpair in zip(self.inpairs, self.outpairs):
        self.motopins.append(inpair + outpair)
```

#### 12.4.3 Ansteuerung

**Robot** Weiters simuliert die Robot Klasse den ganzen Roboter als eine Sammlung von Motorgruppen. Jede Motorgruppe besteht aus zwei Knöpfen und zwei Relays, durch welche man jeden Motor ansteuern kann. In der Robot Klasse werden die einzelnen Motoren in Funktionsgruppe zusammengelegt, wie eine Gruppe für Füße und eine Gruppe für die Schwenkmotoren. Somit ist es möglich die Motoren funktionell aufzurufen. Die Vorwärtsfunktion ruft zuerst die Schwenkmotoren auf und gibt den einzelnen Motoren ihre vorgesehene Richtung und die erste Phase ist erledigt. So werden auch die anderen Phasen erledigt, aber die Robot Klasse achtet darauf, dass jeder Motor seine Bewegung zu Ende bringt und erst dann die nächste Phase gestartet wird.

```
def forwards(self):
    ripdone = []
    feet = len(self.feet)
    swivels = len(self.swivels)

    # start the first phase of swiveling
    for swivel, num in zip(self.swivels, range(swivels)):
        swivel.forwards()

    # wait for the swiveling to finish
    while len(ripdone) < swivels:
        for swivel in self.swivels:
            if swivel.heading:
                swivel.pressed()
            if swivel.heading is None:
                ripdone.append(1)
    ripdone.clear()

    # start the first phase of foot movement
    for even, odd in zip(self.feet[::2], self.feet[1::2]):
        even.forwards()
        odd.backwards()

    # wait for the walking to finish
    while len(ripdone) < feet:
        for foot in self.feet:
            if foot.heading:
                foot.pressed()
            if foot.heading is None:
                ripdone.append(1)
    ripdone.clear()

    # start the second phase of swiveling
    for swivel, num in zip(self.swivels, range(swivels)):
```

```

        swivel.backwards()

        # wait for the second phase of swiveling to finish
        while len(ripdone) < swivels:
            for swivel in self.swivels:
                if swivel.heading:
                    swivel.pressed()
                if swivel.heading is None:
                    ripdone.append(1)
        ripdone.clear()

        # start the second and last phase of foot movement
        for even, odd in zip(self.feet[::-2], self.feet[1:-2]):
            even.backwards()
            odd.forwards()

        # wait for the last phase to finish
        while len(ripdone) < feet:
            for foot in self.feet:
                if foot.heading:
                    foot.pressed()
                if foot.heading is None:
                    ripdone.append(1)
        ripdone.clear()

```

**Motor** Zu der Robot Klasse gibt es die auch die Motor Klasse. Diese Klasse besitzt drei Bewegungszustände: vorwärts, rückwärts und ruhen. Man kann alle diese Bewegungszustände als Funktion aufrufen und der Motor bewegt sich in diese Richtung. Während die Bewegungsrichtungsfunktion gerufen wird dem entsprechend eine innere Variable Richtung gesetzt mit der man den Prozess der Fortbewegung beobachten kann. Außer den Bewegungsrichtungsfunktionen gibt es eine Überprüfungsfunktion “pressed”, welche überprüft, ob der gedrückte Knopf mit der gewollten Richtung übereinstimmt und den Motor dann in Ruhezustand versetzt. Dies sind alle Funktionen eines Motors.

```

def forwards(self):
    self.heading = self.front
    wiringpi.digitalWrite(self.forpin, 1)
    wiringpi.digitalWrite(self.backpin, 0)

def backwards(self):
    self.heading = self.rear
    wiringpi.digitalWrite(self.forpin, 0)
    wiringpi.digitalWrite(self.backpin, 1)

def chill(self):

```

```

    self.heading = None
    wiringpi.digitalWrite(self.forpin, 1)
    wiringpi.digitalWrite(self.backpin, 1)

    # this function test wether the button right button is pressed
def pressed(self):
    if self.heading:
        reading = wiringpi.digitalRead(self.heading)
    else:
        reading = None
    if reading is 0 and self.reversed is False or reading is 1 and self.reversed:
        self.last = self.heading
        self.chill()
        return True
    else:
        return False

```

## 12.5 Der LanguageControl-Thread

Der Sprachkontrollthread wartet ständig auf Sprachkommandos und fügt diese zur cmdList hinzu, falls das erwünscht ist. Die Sprachsteuerung wurde mit dem GStreamer-Plugin der OpenSource-Spracherkennungssoftware CMUSphinx realisiert.

```

void langctrlloop() {
    //Initialization
    loopx = g_main_loop_new(NULL, false);
    //create gstreamer elements
    pipelinex = gst_pipeline_new("pipelinex");
    sourcex = gst_element_factory_make("tcpclientsrc", "tcp-source");
    decoderx = gst_element_factory_make("pocketsphinx", "asr");
    sinkx = gst_element_factory_make("fakesink", "output");
    if (!pipelinex || !sourcex || !decoderx || !sinkx) {
        g_printerr("One element could not be created. Exiting.\n");
    }
    //set keyword file
    g_object_set(G_OBJECT(decoderx), "kws", "/home/pi/insect/keyword.list", NULL);
    //set tcpserver host and port
    g_object_set(G_OBJECT(sourcex), "host", "127.0.0.1", NULL);
    g_object_set(G_OBJECT(sourcex), "port", 3000, NULL);
    //add message handler
    busx = gst_pipeline_get_bus(GST_PIPELINE(pipelinex));
    bus_watch_id = gst_bus_add_watch(busx, bus_callx, loopx);
    gst_object_unref(busx);
    //add elements to the pipeline
    gst_bin_add_many(GST_BIN(pipelinex), sourcex, decoderx, sinkx, NULL);
}

```

```

//link elements together
gst_element_link_many(sourcex, decoderx, sinkx, NULL);
gst_element_set_state(pipelinex, GST_STATE_PLAYING);
//Iterate
puts("Running language control...");
g_main_loop_run(loopx);
//clean up
puts("Returned, stopping playback.");
gst_element_set_state(pipelinex, GST_STATE_NULL);
puts("Deleting pipeline.");
gst_object_unref(GST_OBJECT(pipelinex));
g_source_remove(bus_watch_id);
g_main_loop_unref(loopx);
}

```

Der Kernbaustein von GStreamer sind sogenannte pipelines. Diese sind eine Kombination aus aneinandergehängten Elementen, die entweder Daten, zum Beispiel Audio- oder Videodaten, empfangen, alsasrc ist das Element zur Aufnahme mit dem Mikrofon oder Daten weiterverarbeiten, zum Beispiel ein Ogg-Vorbis-demuxer (welcher zum Abspielen der Audiodateien beim play-Kommando benötigt wird) oder das Pocketsphinx-Element. Es gibt auch noch sogenannte sinks, die Daten in irgendeiner Form ausgeben, in etwa alsasink (spielt Audio über die Lautsprecher ab), filesink (schreibt Daten in eine Datei) oder fakesink (verwirft die Daten), welcher speziell in dem Fall mit Pocketsphinx benötigt wird, weil das pocketsphinx-Element schon die nötige Weiterverarbeitung übernimmt und die Daten danach keine Verwendung mehr haben. Eine Übersichtsgrafik über alle im Projekt verwendeten GStreamer-Pipelines finden Sie in der Section "gstreamer.sh".

Die bei der Spracherkennung verwendete Pipeline besteht aus drei Elementen, die mit `gst_element_factory_make` erzeugt werden:

- sourcex: Die Source der Daten ist ein TCP-Server. Dieser wird von gstreamer.sh erzeugt, mehr dazu unter gstreamer.sh.
- decoderx: Der pocketsphinx-decoder verarbeitet die Daten zu gesprochenem Text, wobei nur Stichwörter, die in der Datei keyword.list genannt werden, beachtet werden.
- sinkx: Ein fakesink, der nur benötigt wird, damit die Pipeline komplett ist.

Mithilfe von `g_object_set` werden die maßgebenden Eigenschaften der Elemente gesetzt: Der Dateipfad von keyword.list, und Host und Port des TCP-Servers. Dann werden alle Elemente verbunden, und der Thread überprüft nun laufend, ob Sprachkommandos erkannt wurden, und fügt diese, wenn erwünscht, zur cmdList hinzu.

## 12.6 Gesprochenes Feedback

```
void speakfeedback(const char addstr[100]) {
    vector<std::string> phrases { "Aye aye sir.", "Of course sir." , "You are the boss.", "Can I help you?" };
    std::random_device seed;
    std::mt19937 engine(seed());
    std::uniform_int_distribution<int> choose(0, phrases.size() - 1);
    std::string elem = phrases[choose(engine)];
    std::cout<<"Random element picked: "<<elem<<"\n";
    char thespeakstr[300];
    strcpy(thespeakstr, elem.c_str());
    strcat(thespeakstr, " ");
    strcat(thespeakstr, addstr);
    espeak(thespeakstr);
}
```

Die Funktion speakfeedback wird, abhängig von der globalen Variable feedbacklevel, entweder nach einem Sprachbefehl, immer nach einem Befehl oder nie ausgeführt. Der Funktion wird eine Zeichenkette als Argument mitgegeben, automatisch wird eine Phrase aus dem Vector phrases vorne hinzugefügt, und der gesamte Text wird dann mithilfe des Kommandozeilenprogramms espeak gesprochen (wobei die verwendete Funktion espeak einfach das Kommandozeilenprogramm mit dem gegebenen Argument aufruft). Beim Start des Programms wird mithilfe dieser Funktion zum Beispiel gesprochen: Äye aye sir. Starting up system.” (siehe Code-Snippet weiter oben).

## 13 gstreamer.sh

gstreamer.sh ist das in Bash geschriebene Skript, welches mithilfe von gstreamer-launch-1.0, der Kommandozeilenapplikation von GStreamer, einen Großteil der benötigten pipelines startet. Die einzige nicht von gstreamer.sh abhängige Pipeline ist playbin, ein vorgefertigtes Template, welches verwendet wird, um innerhalb von server.c Audiodateien abzuspielen.

```
#!/bin/bash

sleep 30

gst-launch-1.0 alsasrc ! audio/x-raw, endianness=1234, signed=true, width=16, depth=16,
    rate=44100, channels=1, format=S16LE ! audioresample ! tee name=t \
t. ! queue ! audioconvert ! audioresample ! tcpserversink host=127.0.0.1 port=3000 \
t. ! queue ! audioamplify amplification=100 ! webrtcdsp noise-suppression-level=high !
    webrtccepprobe ! audioresample ! audio/x-raw, channels=1, rate=16000 !
    opusenc bitrate=20000 ! rtpopuspay ! udpsink host=127.0.0.1 port=5002
```

Das Pausieren von 30 Sekunden ist notwendig, weil ansonsten beim Systemstart die Pipeline zu erzeugen versucht wird, bevor noch alle Audiotreiber richtig

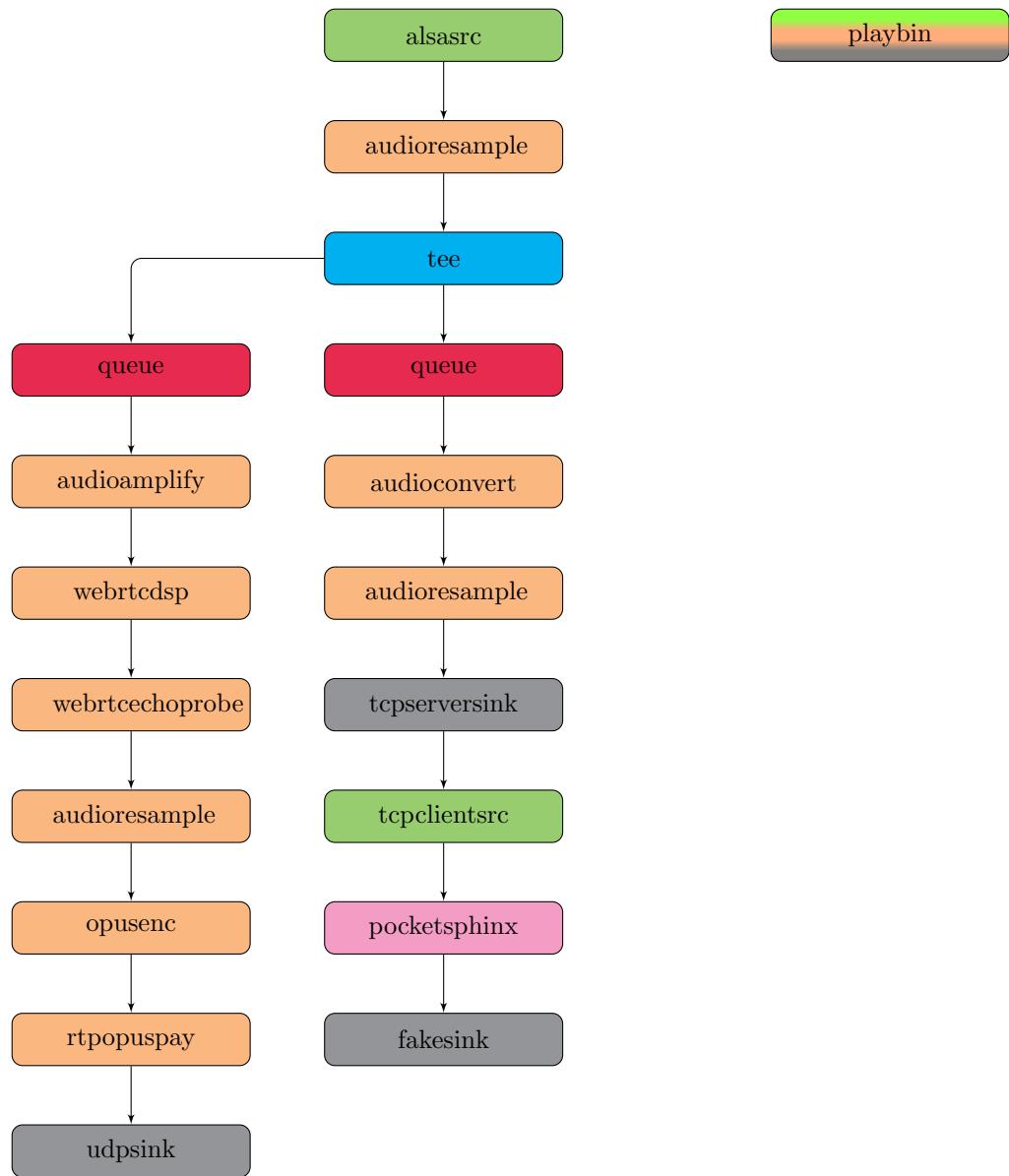


Abbildung 7: GStreamer-Pipelines

initialisiert wurden, was zu einem Kollaps der gesamten Audiofunktionen des Raspberry Pi führt.

Erläuterung der Elemente in der Grafik:

- alsasrc: der Mikrofon-Eingang.
- audioresample: stellt die richtige Samplerate sicher.
- tee: dupliziert den Stream in zwei Streams, wobei einer zu Pocketsphinx und einer zum Janus-Gateway weitergeleitet wird.
- queue: stellt ein gleichmäßiges Abspielen der Audiodaten nach tee sicher.
- audioconvert: stellt das korrekte Format des Streams sicher.
- tcpserversink: erstellt einen TCP-Server am localhost auf Port 3000, wo es die Audiodaten hinleitet.
- tcpclientsrc: dieses Element wird bereits in server.c erzeugt und bezieht Audiodaten vom TCP-Server.
- pocketsphinx: verarbeitet Audio zu Text.
- fakesink: beendet die Pipeline korrekt.
- audioamplify: verstärkt das Audio für den Stream zum Browser um den Faktor 100, damit auch weiter entfernte Geräusche wahrgenommen werden können.
- webrtcdsp und webrtcchoprobe: verringern das Rauschen.
- opusenc und rtpopuspay: kodieren die Audiodaten zu einem RTP-Stream.
- udpsink: erzeugt einen UDP-Server am localhost auf Port 5002, woher Janus-Gateway seine Audiodaten bezieht.

## 14 Die Rolle von systemd

systemd spielt eine Zentrale Rolle für die Software, weil er dafür sorgt, dass alle einzelnen Komponenten des Projekts bei jedem Systemstart und in korrekter Reihenfolge gestartet werden. Einmal muss im Terminal folgender Code ausgeführt werden (um systemd zu konfigurieren):

```
sudo systemctl enable apache2
sudo systemctl enable insectgst
sudo systemctl enable janus
sudo systemctl enable insectdaemon
```

wobei die Dateien insectgst.service, janus.service und insectdaemon.service aus der systemd-Folder des Projekts (siehe github) in /etc/systemd/system kopiert werden müssen. Die Datei insectdaemon.service:

```

[Unit]
Description=Insect daemon
After=network.target apache2.service insectgst.service janus.service

[Service]
Type=idle
Restart=on-abnormal
ExecStart=/home/pi/insect/sh/insectdaemon.sh
User=pi
Environment=DISPLAY=:0

[Install]
WantedBy=multi-user.target

```

Systemd wird also beim Systemstart das Script /home/pi/insect/sh/insectdaemon.sh starten, nachdem network.target, apache2.service, insectgst.service und janus.service gestartet wurden. Ein Vorteil von systemd ist auch, dass im Falle eines Programmabsturzes das Skript erneut gestartet wird. Das Setzen von Environment ist notwendig, weil ansonsten alle Audiofunktionen nicht funktionieren. Der Inhalt von insectdaemon.sh:

```

#!/bin/bash

sleep 45

#This should be run by insect service at startup

#source the variables for pocketsphinx and gstreamer-1.0
echo "Sourcing environment variables."
source /home/pi/insect/sh/pock.sh

#check if /tmp/server.sock file was not removed correctly last time, deleting if exists
if [ -f /tmp/server.sock ]; then
    echo "File found, removing."
    sudo rm /tmp/server.sock
fi

echo "Starting insect daemon."
/home/pi/insect/bin/server.out

echo "Finished."

```

Anfangs wird ein Delay von 45 Sekunden ausgeführt. Dies ist notwendig, damit das Programm nicht zu früh startet, obwohl der Typ des Services auf Idle gesetzt ist. Das Skript pock.sh enthält Umgebungsvariablen, die gesetzt werden müssen, damit Pocketsphinx und GStreamer korrekt funktionieren. Es wird auch überprüft, ob die Datei /tmp/server.sock noch existiert (dann wird sie

gelöscht), dies könnte der Fall nach einem Programmabsturz sein, zum Beispiel nach einem Speicherzugriffsfehler.

## 15 Zugriff außerhalb des Lokalen Netzwerks

Die Fernsteuerung beziehungsweise der Zugriff auf das PHP-Webinterface funktionieren nach obigen Methoden nur, wenn sich der Roboter und das Steuergerät (Tablet) im gleichen Netzwerk befinden. Wir haben einen zweiten Raspberry Pi als WLAN-Router verwendet, der ein WPA2-PSK verschlüsseltes "Hidden Network" erzeugt, um das Eindringen von neugierigen Passanten zu vermeiden. Das Interface kann dann mithilfe des Hostnamen aufgerufen werden:

`http://ehtec1/html/index.php`

Das Installationsskript von RPi\_Cam\_Web\_Interface ermöglicht einen Passwortschutz des Verzeichnisses, was auch genutzt wurde.

Ein entfernter Zugriff ist mit SSH-Tunnels möglich. Dies bietet den zusätzlichen Vorteil, dass die Verbindung per SSH verschlüsselt ist. Wir haben einen normalen Router mit Portforwarding konfiguriert. Der SSH-Server darf überall lokalisiert sein und muss sich nicht im selben Netzwerk wie der Roboter befinden, Es müssen nur die Ports 80, 8088 und 22 offen und in `sshd_config` Remote Port Forwarding erlaubt sein. Alle Funktionen bis auf den Audiolivestream funktionieren makellos auch über das World Wide Web. Für letzteren müsste eine VPS verwendet werden, wobei Janus-Gateway auf der VPS installiert und der Audiostream dorthin gerichtet werden müsste. In diesem Fall sollte der Stream auch verschlüsselt werden, um Eavesdropping zu verhindern und mit dem Roboter im Prinzip ein Spionagegerät im eigenen Haus zu installieren.

Für das Erzeugen der SSH-Tunnel bietet sich das Programm `autossh` an. Es funktioniert wie `ssh` mit der Zusatzfunktion, dass der Tunnel bei einem Absturz neu gestartet wird. Auf dem Raspberry Pi kann `autossh` installiert werden mit

`sudo apt-get install autossh`

und gestartet wird der Tunnel wie folgt:

```
autossh -R 80:localhost:80 root@remotehost.ddns.net
autossh -R 8088:localhost:8088 root@remotehost.ddns.net
```

Die erste Zeile startet den Tunnel des Webservers auf Port 80, und die zweite verbindet sich mit Janus-Gateway, um "Is the server down?" Alerts zu verhindern. Es muss vorher DynDns auf dem Router richtig aufgesetzt und `remotehost` durch die korrekte Subdomain ersetzt werden. Im `.ssh`-Ordner des Roboters müssen die SSH Keys des Remote Hosts platziert sein.

Aufgerufen werden kann das Webinterface nun von allen Netzwerken aus mit:

`http://remotehost.ddns.net/html/index.php`

Natürlich bleibt die Passwort-Autorisierung erhalten.

Das folgende Diagramm stellt die Netzwerktopologie der verwendeten Geräte dar:

