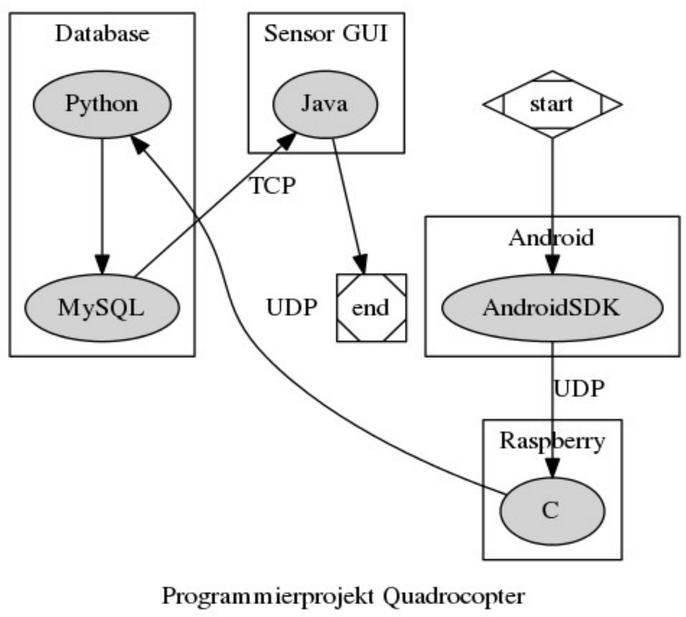
Programmierprojekt SS16
PiSense mit Quadrocopter

Teilnehmer: Marcel Früh, Philipp Gackstatter,
Dominik Heinrich, Jascha Petter, Christoph Weik

Betreuer: Vikas Agrawal
Arbeitsbereich Eingebettete Systeme

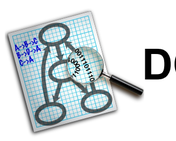
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Architektur:



Programmierprojekt Quadrocopter

```
dtgraph Mygraph2 {
graph [label="Programmierprojekt Quadrocopter" ]

subgraph cluster_0 {
node [style=filled];
C;
label ="Raspberry";
}

subgraph cluster_1{
node [style=filled];
Python -> MySQL;
label = "Database";
}

subgraph cluster_2{
node [style=filled];
Java;
label ="Sensor GUI";
}

subgraph cluster_4{
node[style=filled];
AndroidSDK;
label = "Android";
}

AndroidSDK->C[label="UDP"];
start->AndroidSDK;
C -> Python [constraint=false][label="UDP"];
Java->end;
Java -> MySQL[label="TCP", dir=back];
start [shape="Mdiamond"];
end [shape=Msquare];
}
```



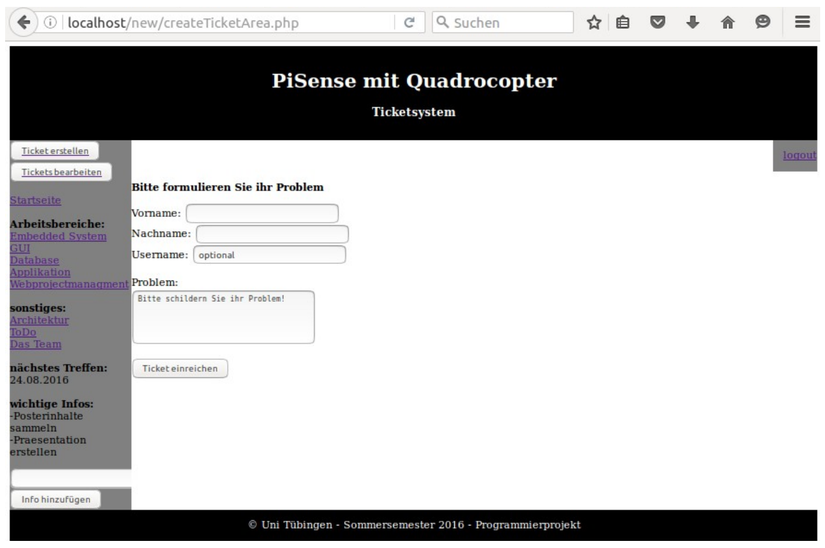
DOT (GraphViz)

WebProjectManagement:

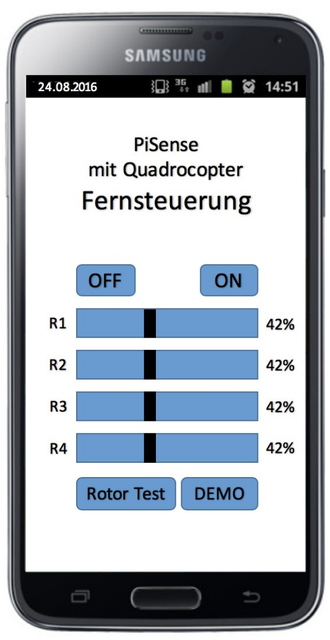
Funktionen:
-Ticketsystem
-Login & Admin Bereich
-Content Management System

LAMP
Linux Apache MySQL PHP,Perl,Python

HTML



Applikation:



ANDROID SDK

Funktionen:
-Motor starten/stoppen
-Rotortest
-Einzelansteuerung der Rotoren
-Demo

UDP

Embedded Systems:

Funktionsweise:
Das Raspberry Pi bekommt seine Befehl von der App via UDP und führt je nach Protokoll unterschiedliche Funktionen aus. Jeder der vier Rotoren lässt sich einzeln ansteuern und die Umdrehung festlegen.

```
    // Motor Data
    sendto(clientSocket, Motor1, m1, 0,
           (struct sockaddr *) &serverAdress, addressSize);
    sendto(clientSocket, Motor2, m2, 0,
           (struct sockaddr *) &serverAdress, addressSize);
    sendto(clientSocket, Motor3, m3, 0,
           (struct sockaddr *) &serverAdress, addressSize);
    sendto(clientSocket, Motor4, m4, 0,
           (struct sockaddr *) &serverAdress, addressSize);

    //Sending Values over I2c

    sendBuffer[0] = pwmValue;

    g_lldI2c_WriteI2c_bl(BLCtrlADRExecuteOrder[0],
           &sendBuffer[0], 1);
    g_lldI2c_WriteI2c_bl(BLCtrlADRExecuteOrder[1],
           &sendBuffer[0], 1);
    g_lldI2c_WriteI2c_bl(BLCtrlADRExecuteOrder[2],
           &sendBuffer[0], 1);
    g_lldI2c_WriteI2c_bl(BLCtrlADRExecuteOrder[3],
           &sendBuffer[0], 1);
    usleep(100000);

    pwmValue = pwmValue + STEPSIZE;

}

while (pwmValue > 5) {
    g_halImu_triggerImuReading_bl();
    g_halImu_triggerBaroReading_bl();
    g_halImu_triggerGyroReading_bl();
    g_halImu_triggerAccReading_bl();
```

```
pi@helikopter: ~

pi@helikopter ~ $ ./HELIKOPTER
pleasefly
Received string is pleasefly
Beschleunigungstest
Eingabe maximaler PWM Wert, 2 Ziffern < 70
100
Eingabe Stepsize, 1 Ziffer, < 10
1
AFAF
TimS1434498113
TimM993
AccX-0.026346
AccX-0.026346
AccZ-10.641397
MagX0.000004
MagY-0.000003
MagZ0.000021
GyrY-0.579852
GyrP0.885037
GyrR-0.671407
Temp28.512501
Pres969.748291
```

RaspberryPi

C

Zudem möchten wir uns bei Chris Mönch, Oliver Breuning, Jürgen Schmidt für die Raspberry Pi Programmierung bedanken.

UDP

GUI:

Funktionsweise:
Die GUI liest aus der Datenbank die gespeicherten Sensordaten aus und stellt diese in Graphen dar. Nutzer können die GUI vorkonfigurieren oder während der Anzeige der Daten die Analyse durch eine Vielzahl von Optionen vornehmen.

Für die GUI Programmierung möchten wir uns bei den Vorarbeitern Alexander Deitche und Juan-Carlos Barradas-Palmeros bedanken.

TCP

Java

python

DB:

Speichert:
-TIMESTAMP
-ACC_X,Y,Z
-MAG_X,Y,Z
-G_ROLL,PITCH,YAW
-TEMP/PRESS
-M1,M2,M3,M4

MySQL

```
dbc.db.query("""
INSERT INTO `SenseData`.`DATA`
(`PITIME`,
`ACC_X`,`ACC_Y`,`ACC_Z`,
`MAG_X`,`MAG_Y`,`MAG_Z`,
`G_ROLL`,`G_PITCH`,`G_YAW`,
`TEMP`,`PRESS`,
`M1`,`M2`,`M3`,`M4`)
VALUES
(FROM_UNIXTIME("""+ts+"""),
"""+ACC_X+""","""+ACC_Y+""","""+ACC_Z+""",
"""+MAG_X+""","""+MAG_Y+""","""+MAG_Z+""",
"""+G_ROLL+""","""+G_PITCH+""","""+G_YAW+""",
"""+TEMP+""","""+PRESS+""",
"""+M1+""","""+M2+""","""+M3+""","""+M4+""");
""")
dbc.db.commit()
```

DATABASE

Zukünftige Ziele: Real Time Code Generation mit MatLab Simulation