

Beta-Abgabe - Smart Urban Farming

Beate Scheibel, Markus Zila

Modelling Language

Der Teil "Modelling Language" war bereits bei der "Prototype"-Abgabe abgeschlossen. Hier wurde nichts mehr verändert.

Physical Implementation

Der physische Teil der Implementation ist zu einem Großteil abgeschlossen. Das Farming Modul ist fertig gebaut und betriebsbereit. Ebenso kamen alle Sensoren zeitgerecht an und sind bereits in das Programm integriert. Es können Temperatur, Licht, Luftfeuchtigkeit und Erdfeuchtigkeit ausgelesen werden.

Mechanisms and Algorithms

Das XML kann importiert und mithilfe eines selbst geschriebenen Parsers ausgelesen werden. Die daraus generierten Daten werden in eine embedded apache derby Datenbank eingelesen. Es existieren folgende tables, wobei die ersten drei der "Prototype"-Implementierung entsprechen.

- Location
- Farming_module
- Crop
- Sensordata: hier werden alle Sensordaten eingelesen und gespeichert. Primary key ist jeweils die Kombination aus farming_module-Name und timestamp (zum Zeitpunkt der Speicherung in die Datenbank). Bei jedem Einlesen in die Datenbank werden alle 4 Messwerte (Temperatur, Luftfeuchtigkeit, Licht, Erdfeuchtigkeit) gespeichert.

Finalisierung des Arduino

Für das Rule-Based-System werden weiterhin die Überlegungen aus der "Prototype"-Abgabe angenommen. Derzeit werden Messdaten gesammelt um Richtwerte für die Rules zu erhalten.

Die Sensor Eingabe ist auf den Arduino implementiert. Er besitzt nun Zugang zu vier Sensoren.

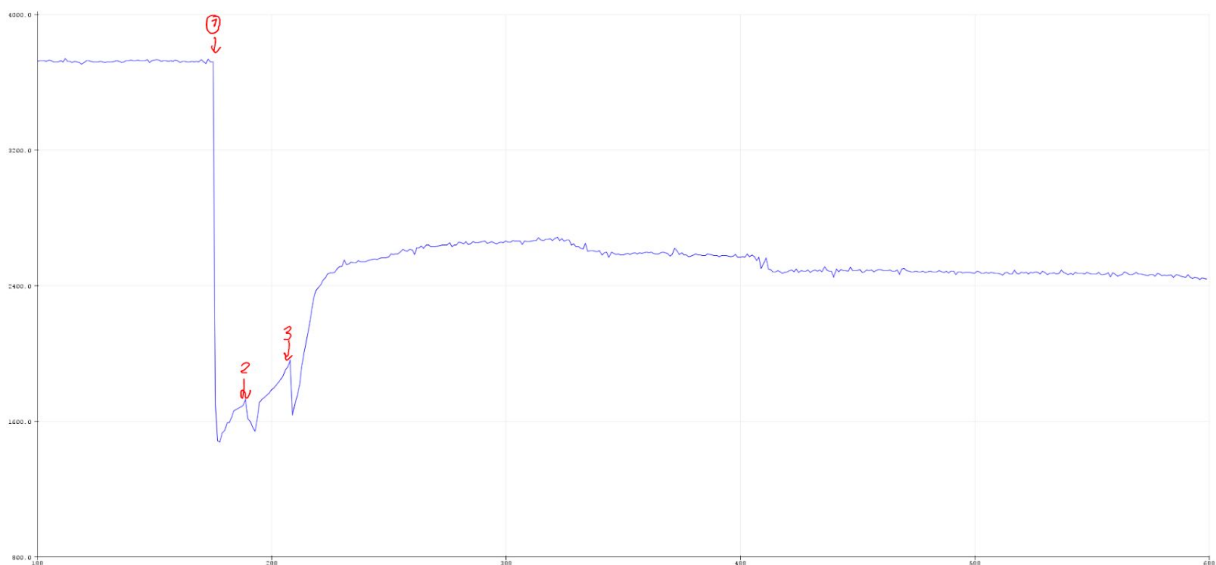
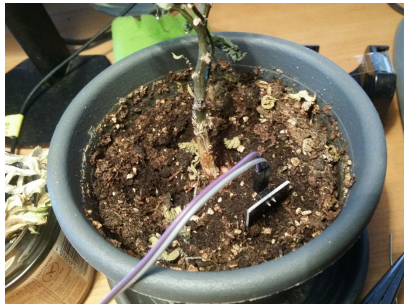
Luftfeuchtigkeit und Temperatur werden digital ausgelesen und werden als float Prozentzahl ausgelesen. Die Bodenfeuchtigkeit und das Licht werden aus einem analogen Lesevorgang

ermittelt. Der Lichtsensor misst den Widerstand (je höher desto mehr Licht und umgekehrt) hier war nur eine Bestimmung von den minimal und maximal betrag notwendig um das Licht in Prozent zwischen minimal und maximal einzuordnen. Wir haben verzichtet das Licht in Lux zu messen, da eine Prozentuelle Bestimmung zwischen Dunkel und Sonnenlicht ausreichend ist.

```
188 //light
189 int lightMax = 4600;
190 int lightMin = 0;
191 int finalLightAdjusted = finalLight;
192 if (finalLightAdjusted > lightMax) {
193     finalLightAdjusted = lightMax;
194 }
195 if (finalLightAdjusted < lightMin) {
196     finalLightAdjusted = lightMin;
197 }
198 int lightDifference = lightMax - lightMin;
199
200
201 double finalLightAdjustedPercent = (double) (finalLightAdjusted * 100) / (double) lightDifference; //make it in %
```

Die Bodenfeuchtigkeit wird invers gemessen. Je höher der Widerstand desto geringer die Feuchtigkeit. Dies haben wir durch Beobachtung der analogen Werte festgestellt:

Setup:



Punkt (1) War die erste Wässerung. Hier erkennt man deutlich, dass der widerstand hinuntergeht. Bei den nächsten Wässerungen (2), (3) kommt der wert beinahe wieder auf das

Niveau der ersten Wässerung. Dann Wird der Widerstand schnell größer weil da das stehende Wasser im Topf versickert und nicht mehr an den Kontakten anliegt. Nach einiger Zeit merkt man, dass der Boden die Feuchtigkeit aufnimmt und der widerstand langsam wieder sinkt.

Durch dieses Experiment haben wir auch den Minimal und Maximal betrag ermittelt. Um eine intuitive Form für die Bodenfeuchtigkeit zu bekommen haben wir den Messbereich um den Minimalbetrag nach unten verschoben, sodass der Minimalbetrag 0 ist, dann haben wir die Messung Inventiert und das Vorzeichen geändert, damit viel Bodenfeuchtigkeit auch ein hohes Ergebnis ist und dann haben wir noch den Wert Normalisiert, sodass der Wert zwischen 0 und 1 ist.

```
//adjust analog Read
//hygrometer
int hygrometerMax = 4180;
int hygrometerMin = 1200;
//hygrometer low value is wet (high %) high value is dry (low %)

//cut out extrem values
int finalHygroAdjusted = finalHygro;
if(finalHygroAdjusted>hygrometerMax){
    finalHygroAdjusted = hygrometerMax;
}
if(finalHygroAdjusted<hygrometerMin){
    finalHygroAdjusted = hygrometerMin;
}
int hygroDifference = hygrometerMax-hygrometerMin;
finalHygroAdjusted = finalHygro-hygrometerMin; //set base line
finalHygroAdjusted = (finalHygroAdjusted - (hygrometerMax-hygrometerMin)) * {-1}; //invers numbers
double finalHygroAdjustedPercent = (double)(finalHygroAdjusted*100)/(double)hygroDifference; //make it in %
```

Da uns aufgefallen ist das die analogen Messungen stark fluktuieren haben wir öfter in der Sekunde gemessen und diese Messungen dann gemittelt. Das Ergebnis ist eine konstantere und verlässlichere Messung. Digitale Messungen dürfen nicht so schnell hintereinander gemacht werden, da sonst ein Fehler passiert.

```
118 int timeIntervalForAVGRead = 20; // in ms
119 int targetIntervalTime = 1000; // in ms
120 int numberOfReads = targetIntervalTime/timeIntervalForAVGRead;
121 int targetGeneralIntervalTime = 10000; // in ms
122 int timeIntervalForGeneralMesure = (targetGeneralIntervalTime) - targetIntervalTime; // ReadInterval - AVG Mesure time -> target 1-5 min
123

for (int i = 0; i <= numberOfReads; i++)
{
    delay(timeIntervalForAVGRead);
    // read sensors

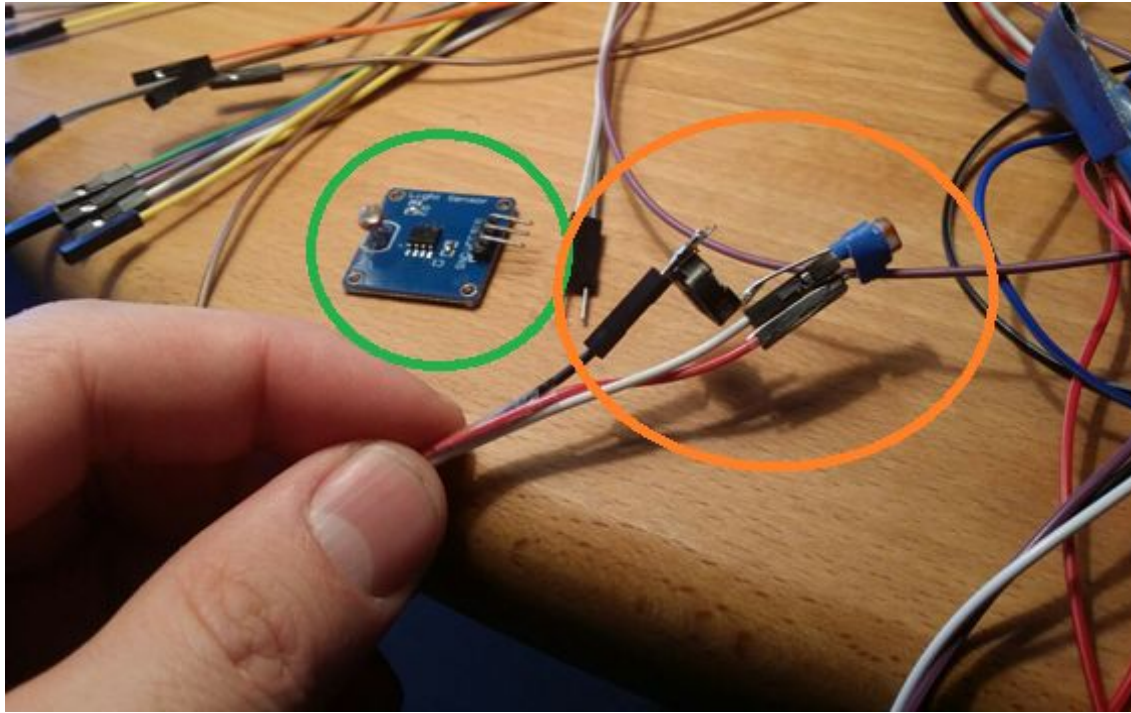
    int light = readLight();
    int hygro = readHygrometer();

    //sum the values upper_bound
    finalLight += light;
    finalHygro += hygro;
}

//make the AVG out of the sum
finalTemperature = temperature;
finalHumidity = humidity;
finalLight = finalLight/numberOfReads;
finalHygro = finalHygro/numberOfReads;
```

Den Lichtsensor mussten wir neu zusammenlöten, da dieser leider keine Gute Performance gebracht hat. Die analogen Messungen waren ständig außerhalb des Messungsbereichs und somit am Max oder Min Widerstand. Deshalb haben wir ein Potentiometer zu dem Lichtsensor

gelötet und den Rest der Elektronik entfernt. Nun funktioniert die Bestimmung des Lichtlevels hervorragend.



Grün (Original Sensor (Einstellung des Lichtniveau nicht möglich))

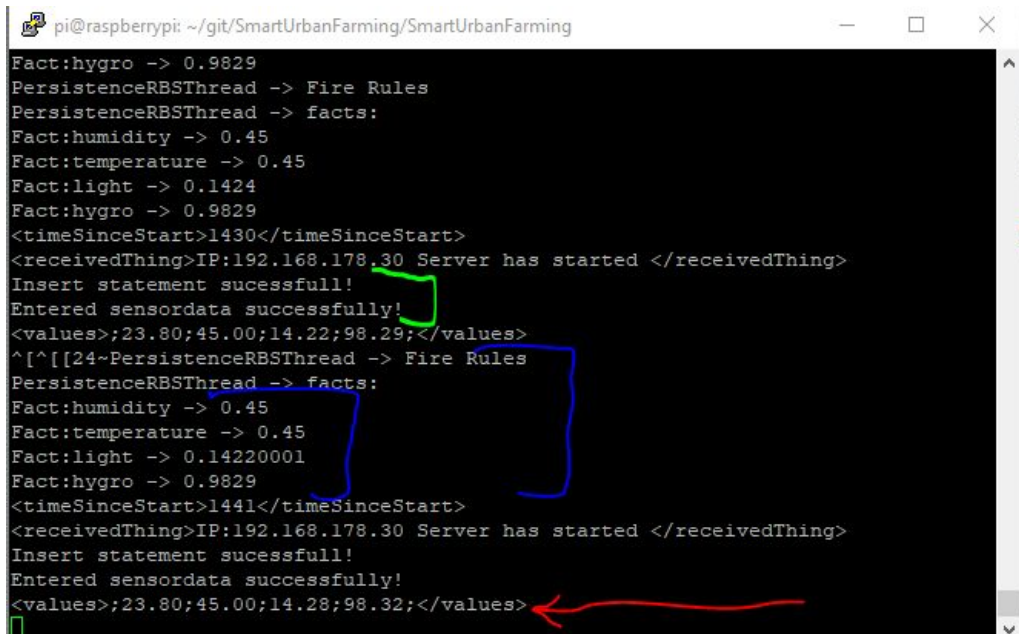
Orange (Eigenbau (Einstellung des Lichtniveau durch 10 KOhm Potentiometer möglich))

Die Sensor Daten werden am Display des ESP32 TTGO angezeigt:



Server weiterentwicklung

Und auch zum Server geschickt (ROT):



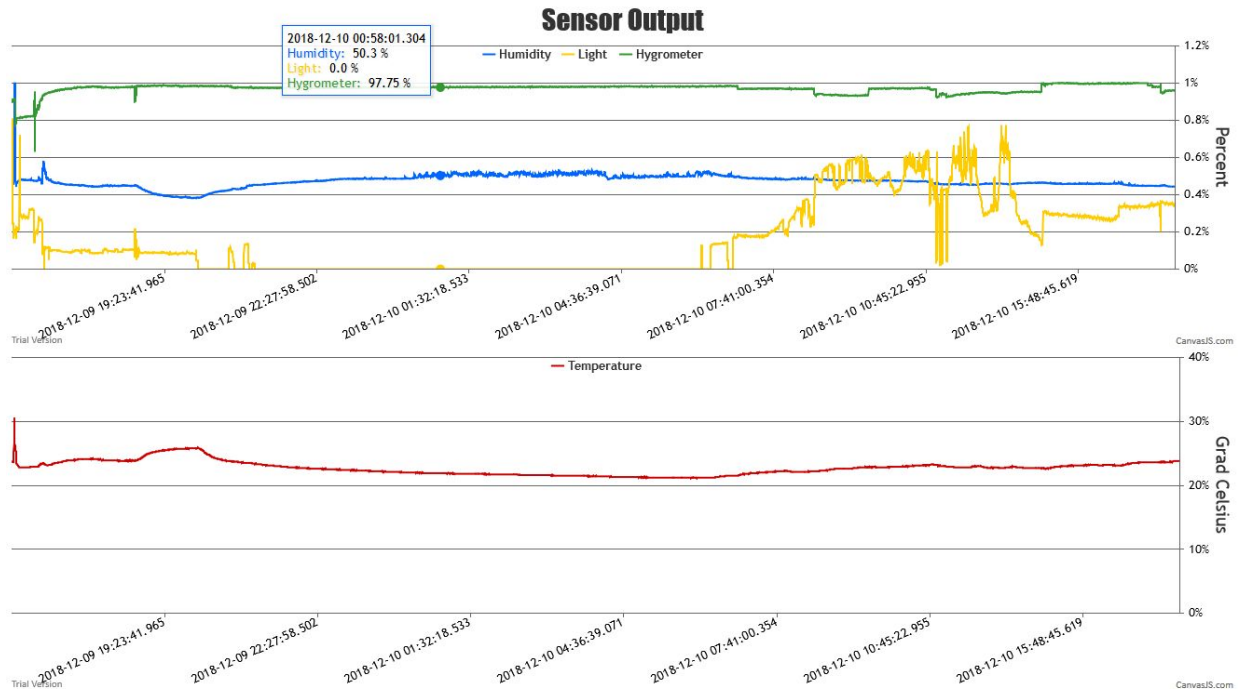
```
pi@raspberrypi: ~/git/SmartUrbanFarming/SmartUrbanFarming
Fact:hygro -> 0.9829
PersistenceRBSThread -> Fire Rules
PersistenceRBSThread -> facts:
Fact:humidity -> 0.45
Fact:temperature -> 0.45
Fact:light -> 0.1424
Fact:hygro -> 0.9829
<timeSinceStart>1430</timeSinceStart>
<receivedThing>IP:192.168.178.30 Server has started </receivedThing>
Insert statement successfull!
Entered sensordata successfully!
<values>;23.80;45.00;14.22;98.29;</values>
^[^[[24~PersistenceRBSThread -> Fire Rules
PersistenceRBSThread -> facts:
Fact:humidity -> 0.45
Fact:temperature -> 0.45
Fact:light -> 0.14220001
Fact:hygro -> 0.9829
<timeSinceStart>1441</timeSinceStart>
<receivedThing>IP:192.168.178.30 Server has started </receivedThing>
Insert statement successfull!
Entered sensordata successfully!
<values>;23.80;45.00;14.28;98.32;</values>
```

The terminal output shows two cycles of data processing. In the first cycle, the sensor data is `<values>;23.80;45.00;14.22;98.29;</values>`. In the second cycle, it is `<values>;23.80;45.00;14.28;98.32;</values>`. A green bracket highlights the first cycle's data entry, and a blue bracket highlights the second cycle's data entry. A red arrow points to the second cycle's data entry.

Dort werden diese Informationen in der Datenbank gespeichert (Grün).

Am Server gibt es eine Funktion, dass man die Sensordaten in ein CSV Datei speichern kann. Dies wird all 100 Sensormessungen gemacht. Weiter wird immer die neuste Messung in der Faktenbasis des RBS gespeichert (Blau). Die Regeln vom RBS werden durch einen Thread der beim Serverstart gestartet wird verwaltet. In der aktuellen Konfiguration werden alle 10 Sekunden alle Regeln des RBS gefeuert.

Die Sensordaten können auch in einer Java Server Page (JSP) eingesehen werden. Die Visualisierung haben wir mit CanvasJS.com einer Diagramm Visualisierung Software gemacht.



Wir haben die Bodenfeuchtigkeit, Luftfeuchtigkeit und Licht in ein Diagramm zusammengefügt, da alle Prozent darstellen. Die Temperatur mussten wir extra darstellen, da diese nicht zwischen normalisiert zwischen 0 und 1 ist.

Das RBS besitzt eine Regel, dass wenn das Licht unter 10 % ist, dass die Regel getriggert wird. Dies ist nur zu Test- und Demonstrations-Zwecke gedacht, da man ein niedriges Lichtlevel leicht herbeiführen kann. Diese ruft eine Funktion auf, dass eine Benachrichtigung an alle Hinterlegten Mailadressen gesendet wird. (Dies funktioniert derzeit noch nicht. Möglicherweise benötigt man dafür eine API key. Wie benutzen die GMail API von Google <https://www.mkyong.com/java/javamail-api-sending-email-via-gmail-smtp-example/>) . Mailadressen kann man auf einer JSP Seite des Server hinterlegen.

Mail	Delete Mail
MaxMustermann@gmx.at	<input checked="" type="checkbox"/>
MartaMusterfrau@gmail.com	<input type="checkbox"/>
<input type="text" value="newMail@gmx.at"/>	<input type="button" value="Daten absenden"/>

Die Mailadressen kann man hinzufügen (Blau) und entfernen (Rot).

Next Steps

Für die Endabgabe muss das Rule-Based-System entwickelt werden (mithilfe von heuristischen Daten, die derzeit gewonnen werden). Ebenso soll das User-Interface noch ausgebaut werden um eine ansprechende Benutzer-Oberfläche zu gewährleisten.

Abschließend wird noch ein Video, dass die Funktionen des "Smart Urban Farming"-Produktes darstellt, produziert.