

Dokumentation

Mechatronischen Projekt

AMBOS DATA FUSION

Patrick Lietze

Hanrui Li

Marco Alonso Barajas Contreras

Marcel Bersdorf



Idee für den Aufbau des Arbeitsplatzes

Zu Beginn des Projektes fanden wir einen Aufbau auf, der aus einem Gestell für den Raspberry Pi 4b mit Kamera bestand. Die Arbeitsfläche bestand aus einem alten Laborwagen. Die Waagen wurden bereits zum Teil in das Programm eingebunden, allerdings war die Hardwareumsetzung der Waagen ebenfalls noch ausbaufähig.

Aus diesem Grund haben wir mit Hilfe von ITEM einen Tisch konstruiert, welcher aus Aluminiumprofil gebaut werden sollte. Die Grundidee war eine zweite Ebene für die Waagen, damit diese optimal platziert werden können. Der Aufbau bestand nun aus dem Gestell für die Kamera, sowie einer Tischfläche, welche durch Aussparungen die Waagen aus der zweiten Ebene hochführt.

Da es sich um einen Aufbau handelt, der oft den Platz wechseln soll, wurden Rollen hinzugefügt, um diesen später leicht bewegen zu können.

Aufgrund des geringen Budgets für das Mechatronische Projekt, konnte diese Konstruktion nicht gekauft werden. Aktuelle Pläne für eine spätere Verbesserung werden in einen extra Verzeichnis auf dem Raspberry Pi gespeichert.

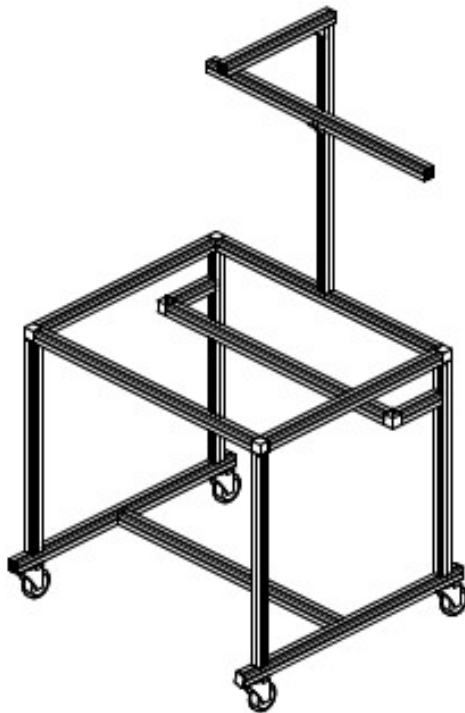


Abbildung 1: Konstruktion Tisch von ITEM

Umsetzung des Arbeitsplatzes

Dem geringen Budget geschuldet, wurde nun der aktuelle Aufbau modifiziert. Das Gestell für den Raspberry Pi wurde behalten. Es konnte ein Tisch gefunden werden, welcher nicht mehr benötigt wird. Auf diesen Tisch konnte nun das Gestell platziert werden, sodass man nun einen zusammenhängenden Arbeitsplatz hatte.

Allerdings war die Voraussetzung, dass der Tisch so gut es geht nicht beschädigt wird. Die Idee einer zweiten Ebene musste deshalb nochmal überdacht werden.

Letztendlich wurde eine Erhöhung gebaut, welche es ermöglicht sämtliche Elektronik und vor allem die Waagen zu verstecken. Für die Erhöhung wurde eine Höhe von 10cm gewählt. Zudem wurde eine Rampe angeschraubt. Auf dieser Rampe konnte nun auch der bereits vorhandene LED-Strip angeklebt werden und bietet auch Platz für weitere Erweiterungen. Die Aussparungen wurden so gewählt, dass diese für zwei Waagen ausreicht.

Hardware-Umbau der Waagen

Die zuvor verwendeten Holzbretter für die Waagen wurden abgeschraubt und durch Holzbretter mit Senkungen für die Befestigungsschrauben angebracht. Dies ermöglicht eine bessere Oberfläche und die Behälter können auf die Waage platziert werden, sodass diese optimal messen können.

Umbau der Elektronik

Die Spannungsversorgung und die weiteren Pins des Raspberry Pis wurden neu verdrahtet. Damit man weitere elektrische Bauteile schnell anschließen kann, wurden die Pins mit einem neuen Kabel verlötet. Das Ende des Kabels wurde wiederum auf eine Platine gelötet. Die einzelnen Pins können nun beliebig, schnell und einfach verwendet werden. Eine genaue Übersicht befindet sich ebenfalls auf dem Raspberry Pi.

Änderungen im Code wurden mit „/******Mechatronisches Projekt WS21/22*****“ /*****/markiert

Anzeige durch 74Hc595 und 7-Segment

Da man im normalen Betrieb keinen Bildschirm zur Verfügung hat und man wissen möchte wie viele Teile aus dem jeweiligen Behälter genommen werden müssen, wurden drei 7-Segmentanzeigen in dem Arbeitsplatz aufgenommen.

Diese Pinbelegung kann man ebenfalls aus der Übersicht entnehmen. Da man nur noch eine geringe Anzahl an freien Pins hatte, musste ein Schieberegister 74hc595 verwendet werden, welche vor jede Segmentanzeige geschaltet wird.

Dadurch lassen sich alle LEDs mit nur drei Ausgängen steuern. Die 74hc595s werden über den Baustein-PIN 9 verkettet, d. h. der PIN des vorherigen ICs geht auf den Baustein-PIN 14.

Somit lassen sich theoretisch unendlich viele Anzeigen steuern. Der erste Baustein wird mit den Baustein-PINS 14 (Storage_Register_Clock), 12 (Shift_Register_Clock) und 11 (Serial_Data_IN) mit dem Raspberry Pi verbunden. (siehe Raspberry Pi Pin-Belegung). Zudem benötigt der Baustein Spannungsversorgung über die Baustein-PINS 10 (+) und 8 (-). Die Anzeigen werden jeweils genau gleich mit den Ausgängen an den Bausteinen verbunden.

Der Programmcode kann beliebig eingefügt werden. Um den Baustein zu verwenden müssen die PINs richtig eingestellt werden. Hierbei muss man aufpassen, welche Pin-Bibliothek verwendet wird. Auf dem Raspberry Pi wird mit WiringPi gearbeitet. Die Pinbeschreibung muss extra nochmal nachgeschaut werden, da diese Belegung des Raspberry Pis nicht entspricht.

```

#define PIN_SH_CP 7 // Pin ist GPIO-#
#define PIN_ST_CP 9 // Pin ist GPIO-#
#define PIN_DS 8 // Pin ist GPIO-#

#define PIN_SHIFT_REGISTER_CLOCK PIN_SH_CP
#define PIN_STORAGE_REGISTER_CLOCK PIN_ST_CP
#define PIN_SERIAL_DATA_IN PIN_DS

#define LSBFIRST 0
#define MSBFIRST 1

bool merker_sevensegment = FALSE;
int prodcountarray[100];
int countersevensegment = 0;
bool merker_sevensegmentfinished = FALSE;

u_int8_t digits0To10[] =
{
//unten rechts//nix//mitte//links oben//oben//rechts oben//unten//links unten
0b10011111, //0
0b10000100, //1
0b00101111, //2
0b10101110, //3
0b10110100, //4
0b10111010, //5
0b10111011, //6
0b10001100, //7
0b10111111, //8
0b10111110 //9
};

```

Abbildung 2: Festlegung der Port, Bitmuster der Zahlen

Der Baustein verlangt eine Bitfolge, um dessen Ausgänge zu steuern. Hierfür wurde die zwei Funktionen verwendet.

shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val) überliefert über den Baustein-PIN 4 eine Bitfolge. Gespeichert wird die Bitfolge mit **triggerLatch()**. Um den Code richtig verwenden zu können muss nun an der gewünschten Stelle die Funktionen **shiftOut** und **triggerLatch** eingefügt werden. Eine häufigere Verwendung der Funktion **shiftOut** ermöglicht eine längere Bitfolge mit dem **triggerLatch** zu speichern.

```

// set pins as output
pinMode( PIN_SHIFT_REGISTER_CLOCK,  OUTPUT );
pinMode( PIN_STORAGE_REGISTER_CLOCK, OUTPUT );
pinMode( PIN_SERIAL_DATA_IN,        OUTPUT );

// initialize output pins to LOW
digitalWrite( PIN_SHIFT_REGISTER_CLOCK,  0 );
digitalWrite( PIN_STORAGE_REGISTER_CLOCK, 0 );
digitalWrite( PIN_SERIAL_DATA_IN,        0 );

if (!merker_sevensegment)
{
    countersevensegment = wp->m_nrBoxes - 1;
    for(size_t j = 0; j < static_cast<size_t>(wp->m_nrBoxes); j++)
    {
        int boxIndex = static_cast<int>(j);
        curBox = &(wp->m_boxes[boxIndex]);
        prodcountarray[boxIndex] = order->m_compList[curBox->m_componentID].m_quantity;
    }

    do
    {
        shiftOut(PIN_SERIAL_DATA_IN, PIN_SHIFT_REGISTER_CLOCK, LSBFIRST, ~digits0To10[prodcountarray[
        triggerLatch(PIN_STORAGE_REGISTER_CLOCK);
        cout << "Counter " << countersevensegment << "\n";
        cout << "Products " << prodcountarray[countersevensegment] << "\n";
        countersevensegment--;
    }while(countersevensegment != -1);

    /*
    shiftOut(PIN_SERIAL_DATA_IN, PIN_SHIFT_REGISTER_CLOCK, LSBFIRST, ~digits0To10[8]);
    shiftOut(PIN_SERIAL_DATA_IN, PIN_SHIFT_REGISTER_CLOCK, LSBFIRST, ~digits0To10[8]);
    shiftOut(PIN_SERIAL_DATA_IN, PIN_SHIFT_REGISTER_CLOCK, LSBFIRST, ~digits0To10[8]);
    */
    //triggerLatch(PIN_STORAGE_REGISTER_CLOCK);
    merker_sevensegment = TRUE;
}
}

```

Abbildung 3: Einbindung der 7-Segmentanzeige in den bestehenden Code

Erweiterung der Waage

Um die entnommenen Artikel besser einschätzen zu können wurden Waagen in den Versuchsaufbau integriert. Diese Waage wird mit Hilfe einer Wägezelle und eines HX711 Verstärkers realisiert.

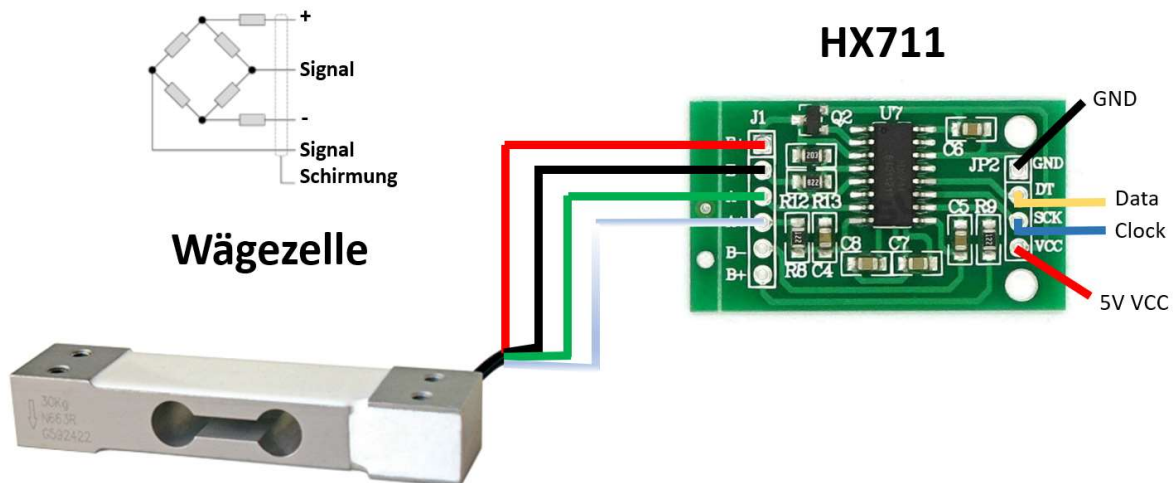


Abbildung 4: Waage Aufbau

Oben ist der schematische Aufbau zu sehen. Die Wägezelle ist mit 4 Widerständen bestückt. Wenn nun Gewicht auf der einen Seite der Wägezelle kommt, verändern die gegenüberliegenden Widerstände gleichmäßig ihre Form. Dadurch ändert sich der Widerstand. Mit einer Messbrücke kann dann das Gewicht als Analogwert ausgegeben werden. Dieser Analogwert wird dann auf der Platine HX711 verstärkt und von dort an den Raspberry Pi weitergegeben. Die 4 Leitungen zur Platine sind die Leitungen, die oben links in der Abbildung dargestellt sind. Von der Platine zum Raspberry geht einmal die Spannungsversorgung mit 5 V und GND und dazu noch die Pins Data und Clock. Über den Clock- und Data-Pin kann nun ein Analogwert ausgelesen werden.

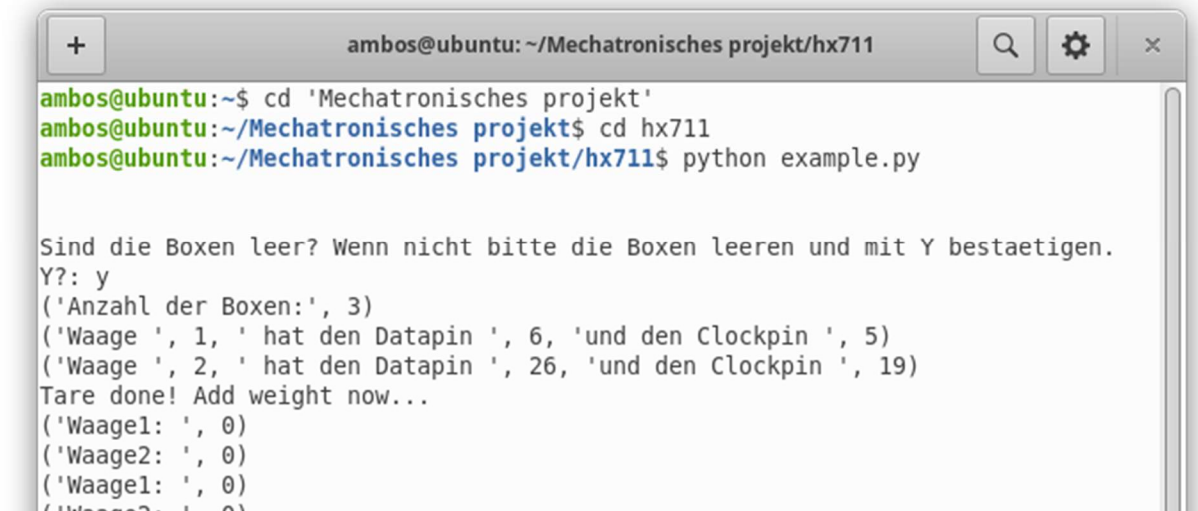
Um Waagen für Boxen zu deklarieren müssen diese im current.json eingegeben werden, wie im in der nachfolgenden Abbildung.

```
"@class" : "de.fraunhofer.ambos_3d.model.Workplace",
"BoxList" :
[
  {
    "componentID" : 0,
    "height" : 260,
    "boxID" : 1,
    "width" : 230,
    "x" : 50,
    "y" : 100,
    "waage" : 1,
    "offset" : 1838,
    "datapin" : 6,
    "clockpin" : 5,
    "productweight" : 50
  }
]
```

Abbildung 5: current.json

Wenn eine Waage bei der Box vorhanden ist muss bei waage 1 eingetragen werden. Dazu braucht die Waage einen Offset der erst einmal 0 ist. Auch die Pins für Data und Clock werden hier für jede Waage hinterlegt. Und zu dem noch das Gewicht eines einzelnen Produkts.

Wenn das current.json bearbeitet wurde muss das Programm einmal gestartet werden. Das Programm startet man so:



```
ambos@ubuntu: ~$ cd 'Mechatronisches projekt'
ambos@ubuntu:~/Mechatronisches projekt$ cd hx711
ambos@ubuntu:~/Mechatronisches projekt/hx711$ python example.py

Sind die Boxen leer? Wenn nicht bitte die Boxen leeren und mit Y bestaetigen.
Y?: y
('Anzahl der Boxen:', 3)
('Waage ', 1, ' hat den Datapin ', 6, 'und den Clockpin ', 5)
('Waage ', 2, ' hat den Datapin ', 26, 'und den Clockpin ', 19)
Tare done! Add weight now...
('Waage1: ', 0)
('Waage2: ', 0)
('Waage1: ', 0)
('Waage2: ', 0)
```

Abbildung 6: Start Pythonskript

Es gibt 2 Möglichkeiten die Waagen zu kalibrieren:

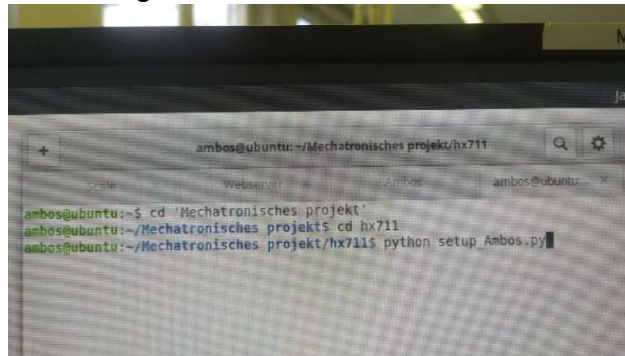
1. Hierzu wird eine leere Box aufgelegt. Dann wird ein bekanntes Gewicht aufgelegt, dieses sollte ca. die Hälfte des Maximalgewichts der Wägezelle betragen. Der Wert der angezeigt wird muss notiert werden. Dieser Wert wird dann durch das aufgelegte Gewicht geteilt, daraus erhält man seinen Offset. Dieser Offsetwert muss nun im current.json hinterlegt werden.

Zum Bsp.:

Wägezelle mit maximal 5kg, Gewicht zum kalibrieren 2500g, gemessener Wert 882356

$$\text{Offset} = 882356 / 2500 = 352,94 = 353$$

2. Mit einem vorbereiteten Skript. Um dieses Skript auszuführen muss man in der Konsole folgendes eingeben:



```
ambos@ubuntu:~/Mechatronisches projekt/hx711
ambos@ubuntu:~/Mechatronisches projekt$ cd 'Mechatronisches projekt'
ambos@ubuntu:~/Mechatronisches projekt$ cd hx711
ambos@ubuntu:~/Mechatronisches projekt/hx711$ python setup Ambos.py
```

Abbildung 7: Starten Kalibrierung

Danach einfach den Anweisungen im Skript folgen. Der berechnete Offset wird in dem Textfile waageoffset.txt im Ordner hx711 gespeichert. Einfach diesen Offsetwert nehmen und im current.json eintragen.

Wenn die Kalibrierung beendet ist kann gemessen werden. Hierzu einfach das Programm noch einmal ausführen.

Wenn alle Boxen leer sind, kann man mit Y bestätigen und es werden die Waagen eingerichtet. Die Waagen werden zuerst genullt, danach kann das Ambos-Pi Programm gestartet werden. Die Übergabe zwischen den 2 Programmen funktioniert durch Text-files. Das Python-Skript schreibt das Gewicht in Gramm in das Textdokument und der Ambos-Pi Code liest das Textdokument dann aus.

Das Python-Skript ist eine Abwandlung dieser Anleitung:

<https://tutorials-raspberrypi.de/raspberry-pi-waage-bauen-gewichtssensor-hx711/>

Es wurde ergänzt mit einer Datenübergabe vom current.json und angepasst an 2 Waagen. Der Zeitintervall in dem die Waage ausgelesen werden, kann im unteren Teil des Python-Skripts angepasst werden.

Das Gewicht welches aus der Textdatei kommt wird dann im Ambos Programm in 2 Methoden verwendet.

```
//ORIGINAL CODE
bool Box::isBoxEmpty()
{
    //***** HS ESSLINGEN PROJEKT 21/22
    if (LoadCell == 1)
    {
        string line = "";
        string pathhx711 = "/home/ambos/Mechatronisches projekt/hx711/waage" + std::to_string(m_nr) + ".txt";
        std::ifstream myfile(pathhx711);

        if (myfile.is_open())
        {
            getline(myfile, line);
        }

        newweight = std::stof(line);

        myfile.close();
        merker_oldweight = false;

        //Merker 7 Segmentanzeige auf False
        merker_sevensegment = false;

        if (newweight > 2.0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    //*****
}
```

Abbildung 8: isBoxempty-Methode

Die erste Methode ist die isBoxempty-Methode. Hier wird zuerst bei der Box geprüft, ob eine Waage eingerichtet ist. Wenn dies der Fall ist, dann wird das Gewicht vom Textdokument ausgelesen. Wenn dieses Gewicht unter einem Grenzwert ist, gibt die Methode TRUE zurück. Das Programm meldet, dass die Box leer ist und aufgefüllt werden muss.

```

bool FrameHandler::checkBoxChange(int boxIndex)
{
    // Wir checken nur ob sich in der Box etwas geändert hat,
    // wenn die Anzahl der erkannten Hände in den Zeitschritten davor >=minDetThresh sind
    if ((boxDetFlags[boxIndex] >= minDetThresh) && curBox->changeInBox())
    {
        // Ist das Order objekt != Null so ist Ambos in der Version 1 gestartet, somit muss die
        // Prozesslogik direkt hier umgesetzt werden
        if (order != nullptr)
        {
            //***** HS ESSELINGEN PROJEKT 21/22
            float Differenz = 0;
            float f_takenProducts = 0;
            int i_takenProducts = 0;
            if (curBox->LoadCell == 1)
            {
                string line = "";
                string pathhx711 = "/home/ambos/Mechatronisches projekt/hx711/waage" + std::to_string(curBox->m_nr) + ".txt";
                std::ifstream myfile(pathhx711);

                if (myfile.is_open())
                {
                    getline(myfile, line);
                }

                curBox->newweight = std::stof(line);
                cout << "Gewicht nach Entnahme der Box " << curBox->m_nr << " als float: " << curBox->newweight << " gramm\n";
                myfile.close();
                curBox->merker_oldweight = false;
                Differenz = curBox->oldweight - curBox->newweight;
                if (Differenz < 0)
                {
                    Differenz = 0;
                }
                cout << "Die Differenz ist: " << Differenz << " gramm\n";
                f_takenProducts = Differenz/curBox->productweight;
                //cout << "Float: " << f_takenProducts << "\n";
                i_takenProducts = lround(f_takenProducts);
                cout << "Es wurden " << i_takenProducts << " Teile aus der Box " << curBox->m_nr << " entnommen.\n\n";
                curBox->m_componentCounter = curBox->m_componentCounter + i_takenProducts;
            }
            else
            {
                // Setze den Zähler der Komponente hoch
                curBox->m_componentCounter++;
            }
        }
    }
}

```

Abbildung 9: checkBoxChange-Methode

Die zweite Methode ist die checkBoxChange-Methode. Diese wird immer aufgerufen, wenn die Kamera eine Hand erkennt. Bevor die Methode ausgeführt wird, wird in der Box-Klasse die Variable oldweight mit dem aktuellen Gewicht beschrieben. Wenn die Methode ausgeführt wird, wird das neue Gewicht ausgelesen und in der Variable newweight gespeichert. Aus diesen beiden Variablen wird die Differenz gebildet.

Die Differenz wird dann durch das Gewicht eines einzelnen Produktes geteilt. Somit kann berechnet werden wie viele Produkte entnommen wurden. Die Zählvariable m_componentCounter wird dann um die berechneten Produkte erhöht. Falls keine Waage hinterlegt ist, sich jedoch das Frame geändert hat wird der else-Teil aktiviert und die Zählvariable wird um 1 inkrementiert.

Gleitender-Mittelwert-Filter für Sensoren

Die Leistung von Sensoren wird immer durch elektrische und andere Geräusche behindert, daher ist es sinnvoll, einen Filter zu verwenden. Aus diesem Grund wurde ein C++ Class Gleitender-Mittelwert-Filter (GMF) erstellt. Ein GMF ist ein Tiefpass-Filter, dessen Eingang einer diskreten Zeitreihe (Die Werte der Sensoren) ist. Die mathematische Darstellung des GMF ist

$$y(t) = \frac{1}{n} \sum_{i=0}^{n-1} x(t - i)$$

Dabei steht n für die Ordnung des Filters und x für den Wert des Sensors. Die Ordnung des Filters hat eine direkte Auswirkung auf die Leistung des Sensors (siehe Abbildung 1).

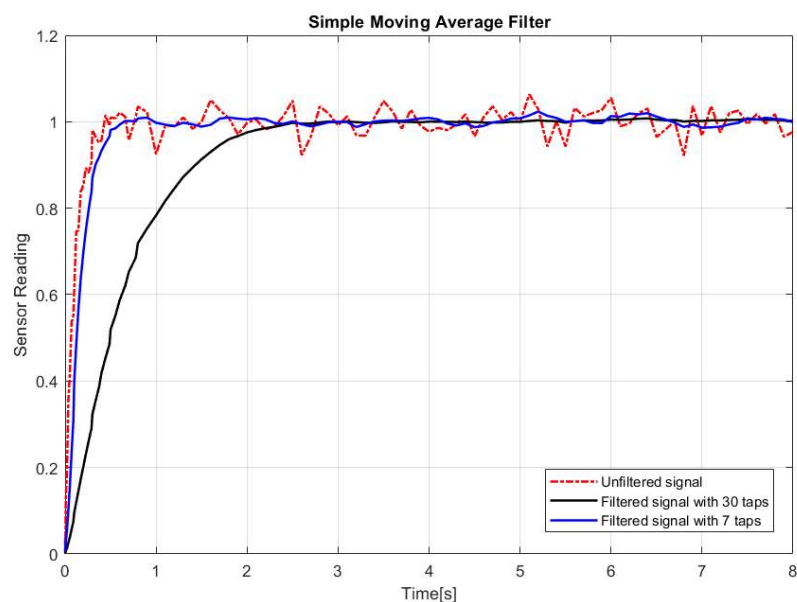


Abbildung 10: Sensor Werte mit verschiedenen GMF-Ordnungen

Die Abbildung zeigt, wie sich die Leistung des Sensors mit der Ordnung des Filters ändert. Bei einer hohen Ordnung arbeitet der Sensor langsamer, aber fast geräuschlos. Mit einem Filter niedriger Ordnung arbeitet der Sensor schneller, aber mit mehr Geräusch.

Für den Projekt, ein C++ Class *Filter* wurde erstellt, um die Sensoren zu filtern. Der Filter ist in der *Namespace AB*. Die Abbildung zeigt der *hpp* Code

```

namespace AB{
typedef unsigned int uint;
class Filter{
private:
//Number of taps in the filter
uint N = 1;
//Output of the filter
double y;
//Vector to store the previous values for the filter
std::vector<double> x;

//Private initialization function
void _init(){ x.resize(N,0.0); }
void _reset(){ y = 0;}

public:
//Constructors
Filter(){_init();}
Filter(const uint & n ): N(n){_init();}
//Calculate the output of the value
double filter(const double & );
void print() const;
//Destructor
~Filter(){};
};
}

```

Abbildung 11: hpp file mit dem Class Filter

Das Class hat zwei private Werte: N (die Ordnung des Filters) und y (der Ausgang des Filters) und einen Vektor für die Eingangswerte mit der Größe N . Außerdem gibt es zwei private Funktionen, die die privaten Werte setzen und zurücksetzen. Schließlich gibt es noch zwei öffentliche Funktionen: $print()$ für Debug, und $filter(const double \&)$ für die Filterung der Werte. Um ein Filterobjekt zu erstellen, muss die Ordnung des Filters angegeben werden.

Schlafmodus der Anlage

Um das AMBOS-System zu optimieren, wurde ein Schlafmodus-Algorithmus entwickelt. Das Ziel der Schlafmodus ist, dass das System nicht funktioniert, wenn niemand vor dem Tisch sitzt. Ein Infrarot Sensor unter dem Tisch erkennt, ob jemand vorsitzt. Die Abbildung zeigt die Komplette Algorithmus.

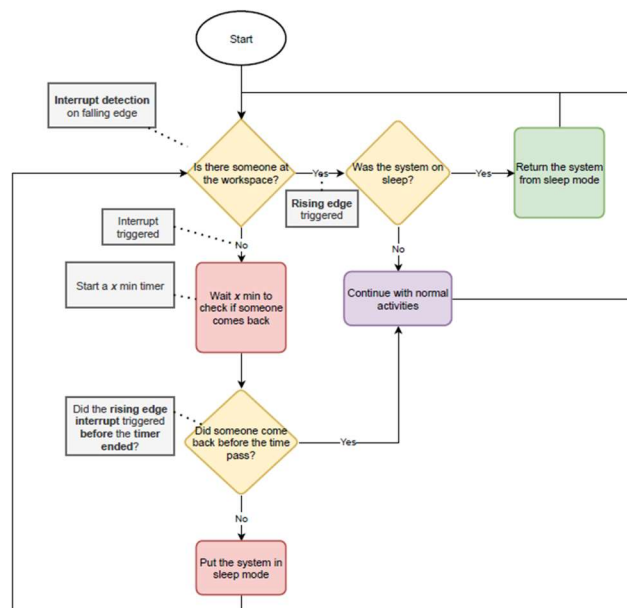


Abbildung 12: Ablaufdiagramm Schlafmodus

Für die Implementierung des Algorithmus in C++ und Raspberry Pi wurden *GPIO-* und *Timer-Interrupts* verwendet. Das System verwendet einen 10-Sekunden-Timer, ohne dass eine Erkennung vor dem Tisch stattfindet, um den Schlafmodus zu aktivieren. Wenn das Schlafmodus eingeschaltet wird, gibt es eine Schleife, dass die Kamera deaktiviert. Weitere Funktionen können hinzugefügt werden.

Das Schlafmodus funktioniert mit vier Dokumente: *GPIO_interrupt.cpp*, *GPIO_interrupt.hpp*, *Timer.cpp* und *Timer.hpp*. *Timer.cpp* , wird von *GPIO_interrupt.cpp* verwendet, um den *Timer* ein- und auszuschalten mit der Logik von Abbildung 3. *Timer.cpp* benutzt auch *signal.h* und *sys/time.h libraries*.

```
//Configure the timer
//The additional code must be use to start the timer
//setitimer(ITIMER_VIRTUAL, &T, NULL);
void timer_init(int wait_time, struct itimerval *T, struct sigaction *S, void(*T_handler)(int)){ /*Initializa the timer with wait_time seconds*/
    /*Install timer handler as the signal handler for SIGVTALRM*/
    memset (S, 0, sizeof (*S));
    S->sa_handler = T_handler;
    sigaction (SIGVTALRM, S, NULL);

    /* Set the timer to "wait time" seconds*/
    T->it_interval.tv_usec = 0;
    T->it_interval.tv_sec = 0;
    T->it_value.tv_usec = 0;
    T->it_value.tv_sec = wait_time;
}

//Stop the timer function
void stop_timer(struct itimerval *T, struct sigaction *S, void(*T_handler)(int)){ /*Stop the timer before time ends*/
    timer_init(0, T, S, T_handler);
    setitimer(ITIMER_VIRTUAL, T, NULL);
}
```

Abbildung 13: timer.cpp Code

GPIO_interrupt.cpp hat zwei *ISRs* (Interrupt Service Routine), *GPIO ISR* für positive und negative Flanken, um zu erkennen, ob jemand vor dem Tisch sitzt und *Timer ISR*, um das Schlafmodus zu aktivieren. Außerdem gibt es noch eine weitere Funktion, *GPIO_interrupt_init()* (die Funktion benötigt *wiringPi.h*), die an den Anfang des Codes gestellt werden sollte. Abbildung zeigt diese Funktionen.

```

#define IR_SENSOR_PIN 2;

//Variables for the timer
struct itimerval T;
struct sigaction S;
unsigned int wait_time = 5; //Seconds

int sleepMode = 0;

//Initialization of the GPIO as Interrupt
void GPIO_interrupt_init(){
    wiringPiSetup();
    wiringPiISR(IR_SENSOR_PIN, INT_EDGE_BOTH, GPIO_handler);
}

void timer_handler (int signum){ /*Interrupt Handler*/
    static int SleepMode = 1;
}

//Interrupt Service Routine
void GPIO_handler(){
    int ir_sensor_val = !digitalRead(IR_SENSOR_PIN);

    if ((ir_sensor_val && !sleepMode)|| (ir_sensor_val && sleepMode)){
        //Rising edge was detected
        if (!sleepMode) stop_timer(&T,&S,timer_handler); //Stop the timer
        else sleepMode = 0; //Wake up from sleep mode
    }
    else if(!ir_sensor_val && !sleepMode){
        //Falling edge was detected
        //Configure timer
        timer_init(wait_time,&T,&S,timer_handler);
        //Start timer
        setitimer(ITIMER_VIRTUAL, &T, NULL);
    }
}

```

Abbildung 14:GPIO_interrupt.cpp Code

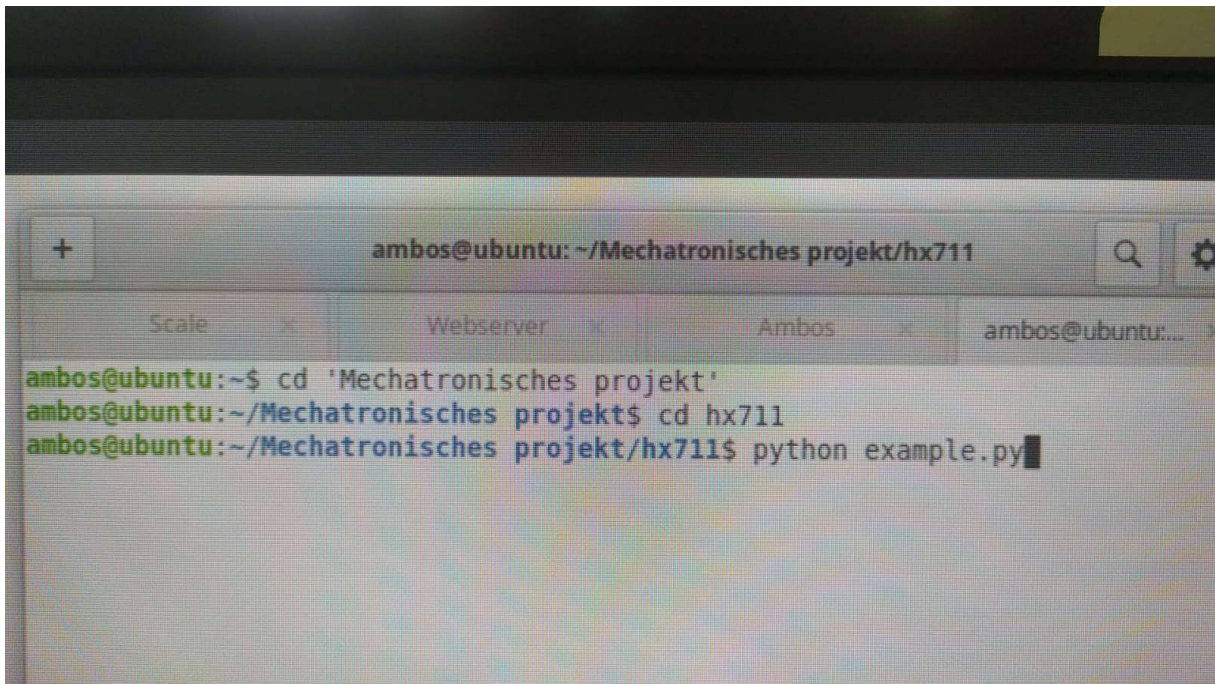
Abbildung 14 zeigt zwei Einstellungen die wichtig sind. Die Werte *wait_time* und *IR_SENSOR_PIN* müssen definiert werden.

Starten des Programms

Normalerweise wird das Programm im Autostart geöffnet. Um die Autostarteinstellung zu bearbeiten einfach im Terminal `gnome-session-properties` eingeben.

Wenn das Programm geschlossen wird oder abstürzt gibt es 2 Möglichkeiten.

1. Nur das Ambosterminal ist abgestürzt, die Waagen messen aber weiter. Dann einfach auf dem Desktop die `Ambos_Restart` mit einem Doppelklick öffnen.
2. Wenn alles zu ist muss man folgende Befehle in der Konsole eingeben:



```
ambos@ubuntu: ~/Mechatronisches projekt/hx711
Scale x Webserver x Ambos x ambos@ubuntu:...
ambos@ubuntu:~$ cd 'Mechatronisches projekt'
ambos@ubuntu:~/Mechatronisches projekt$ cd hx711
ambos@ubuntu:~/Mechatronisches projekt/hx711$ python example.py
```

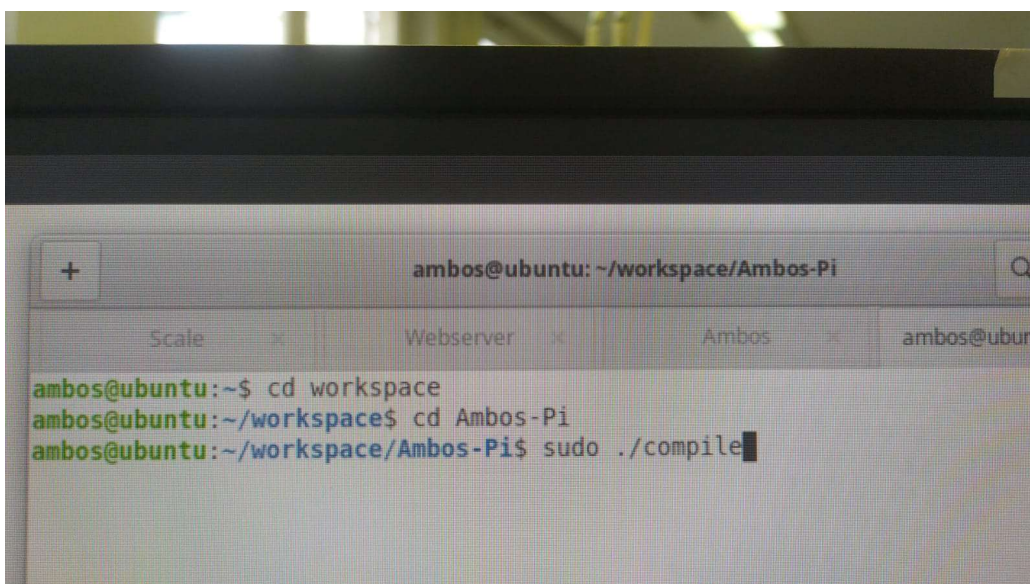
Abbildung 15: Starten Python-Skript

Mit diesen Befehlen wird das example.py Skript gestartet. Dieses Skript ist dazu zuständig, dass die Waagen funktionieren. Es startet aber auch das Skript webserver.py welches dann wiederum das Ambos Programm ausführt.

Kompilieren

Das C-Programm kann über Codelite editiert werden. Wenn das Programm editiert wurde einfach über Build->Rebuild Project das Projekt neu erstellen. Danach kann das Programm über Build->Build and Run Project gestartet werden.

Um das neue Programm auch im Autostart zu haben muss es kompiliert werden. Hierzu müssen folgende Befehle im Terminal eingegeben werden:



```
ambos@ubuntu: ~/workspace/Ambos-Pi
Scale x Webserver x Ambos x ambos@ubunt
ambos@ubuntu:~$ cd workspace
ambos@ubuntu:~/workspace$ cd Ambos-Pi
ambos@ubuntu:~/workspace/Ambos-Pi$ sudo ./compile
```

Das Kompilieren kann einige Minuten dauern.

Ausblick

Eine weitere Erweiterung könnte in Zukunft die Waage sein. Diese wird im Moment über ein extra Python-Skript gesteuert und könnte noch in den bestehenden C-Code aufgenommen werden. Hierbei ist zu beachten, dass bereits vorinstallierte Bibliotheken auf dem Raspberry befinden. Unter anderem müssen die Portbelegungen der Waage mit WiringPi erfolgen.

Des Weiteren sind Hardware-Verbesserung möglich. Sowohl die Anzahl der 7-Segmentanzeigen, aber auch das Gestell für die Waage kann weiter optimiert werden.

Durch 3D-Druck hergestellte Waagenbehälter als Alternative

Um den Einbau der Waagen in das Hardwaresystem einfacher und genauer zu gestalten und auch die Struktur des Systems schöner und leichter zu machen, wird es vorgesehen, die Waagenbehälter für die Kraftsensoren (0 ...1 kg, YZC-133) und Wägezellenverstärker (HX711) mittels 3D-Druck herzustellen.

Modelle

Für den Einbau von Wägezellen gibt es bereits einige 3D-Modelle im Internet. Nach der Bezugnahme auf mehrere Modellmuster wird ein Typ mit einer kleinen Box für Einbau der HX711 gewählt. Nach Überprüfung und Anpassung der Modelle ist die endgültige Modellgröße und Bemaßung gut für das System geeignet. Wobei werden Bemaßung der in diesem System verwendeten Kraftsensoren und Größe des Installationsraums berücksichtigt.

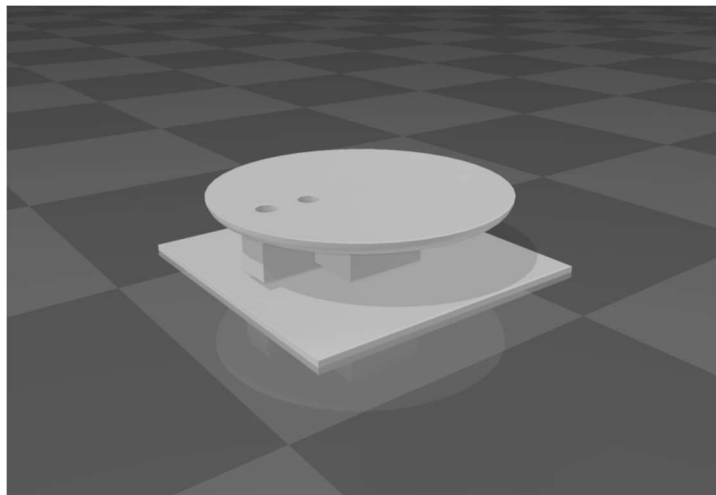


Abbildung 16: Beispielhalterung

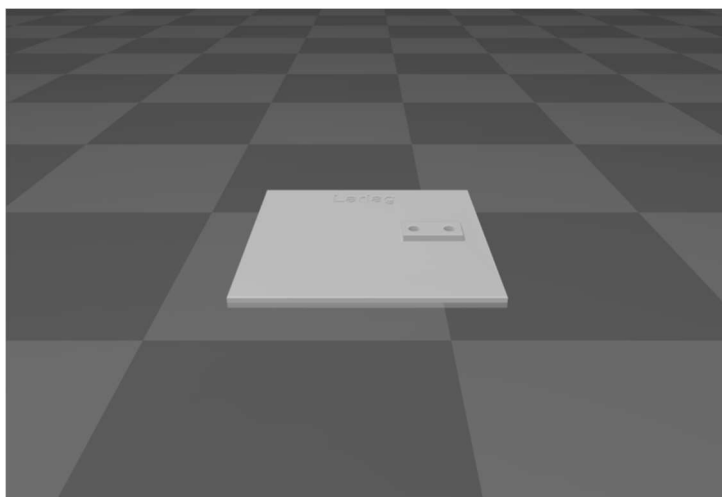


Abbildung 17: Bodenplatte

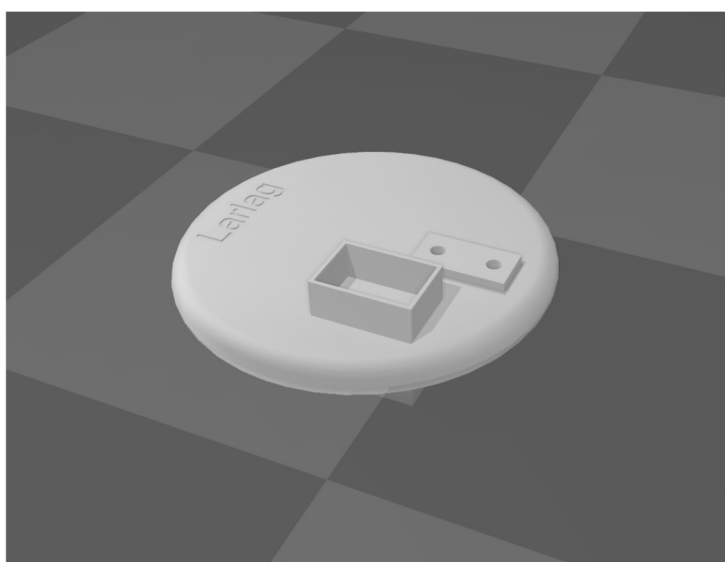


Abbildung 18: Kopfplatte mit Box für HX711

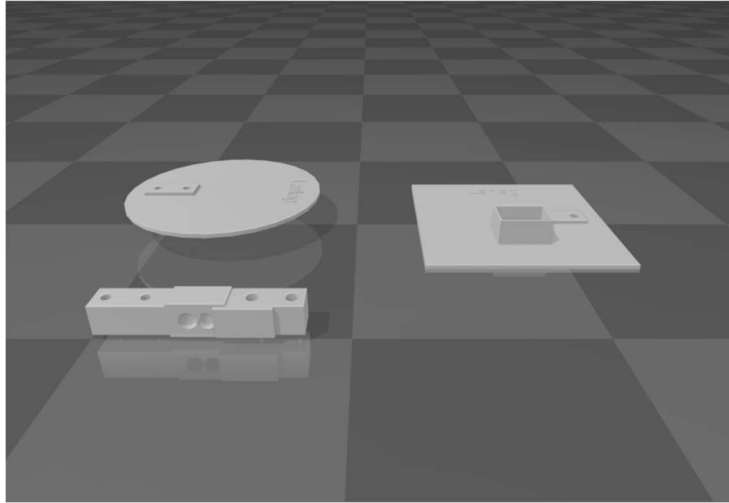


Abbildung 19: zu druckenden Teilen und Wägezelle

Kosten

Durch verschiedene Kostenbewertungsfunktionen im Internet lässt sich feststellen, dass diese Alternative mittels 3D-Druck mindestens 60 Euro kosten würde. Angesichts der begrenzten Projektkosten ist der 3D-gedruckte Waagenbehälter vorerst nicht realisierbar, wenn die notwendigen Bauelemente schon bestellt werden. Aber für die spätere Verbesserung und Erweiterung des AMBOS-Systems sollte diese Möglichkeit noch berücksichtigt werden.

Verbesserungsvorschläge

1. Da im Moment die gesamte Stromversorgung über ein Netzteil erfolgt, fehlt dem Raspberry Strom, wodurch er nur verzögert reagiert. Damit die Stromversorgung aufgeteilt werden kann, muss die Stromversorgung der weiteren Teile erneut verschaltet werden.
2. Da es viele einzelnen elektrotechnische Komponenten im Aufbau gibt und diese durch zusätzliche Erweiterungen im nur ergänzt wurden, müsste man für die Zukunft einen genauen Schaltplan entwickeln, sowie auch die Lötstellen neu überdenken.
3. Die Waagen werden im Moment über ein Python-Skript gesteuert. Die Messwerte werden in einem Text-File gespeichert, auf das das Hauptprogramm zugreift. Wird während dem Schreiben der Waagen ein Teil entnommen, so stürzt das Programm ab, da dieses keinen Wert im Text-File erkennt. Es muss für einen reibungslosen Betrieb eine andere Lösung gewählt werden. Im Optimalfall wird die Waage in das C-Skript eingefügt, um den Zwischenschritt mit dem Text-File zu umgehen.
4. Neben den 7-Segmentanzeigen können weiter visuelle Erweiterungen (z.B. LCD-Display) hinzugefügt werden, um den Monitor nur für Erweiterungen anzuschließen.