

Project Data Explorer - Mohne

Projektbericht

Entwicklung eines Fahrzeugs für autonome/ ferngesteuerte Datenerfassung inklusive der Umweltbedingungen

Eingereicht bei:

**Fachhochschule Kufstein Tirol Bildungs GmbH**

**Studiengang Smart Products & Solutions**

Verfasser:

**Anna Duregger, Christian Gruber, Christoph Jungwirth, Maximillian Möbes, Ludwig Paula, Markus Wiesmüller**

Abgabedatum:

**10.06.2017**

Table of Contents

[List of Figures IV](#_Toc516389708)

[List of Tables V](#_Toc516389709)

[List of Abbreviation VI](#_Toc516389710)

[Kurzfassung IX](#_Toc516389711)

[Abstract X](#_Toc516389712)

[1. Conecpt Development 9](#_Toc516389713)

[1.1 Konzept der Fahrmodi 9](#_Toc516389714)

[1.1.1 autonom 10](#_Toc516389715)

[1.1.2 ferngesteuert 11](#_Toc516389716)

[1.2 Fahrzeugaufbau 13](#_Toc516389717)

[1.2.1 CAD als Unterstützung für die Fertigung und Designentwicklung 13](#_Toc516389718)

[1.2.2 Elektronik 15](#_Toc516389719)

[1.2.3 Kommunikation 16](#_Toc516389720)

[2. Model Based Analytics 17](#_Toc516389721)

[2.1 Digitaler Zwilling 17](#_Toc516389722)

[2.1.1 Allgemein 17](#_Toc516389723)

[2.1.2 Projektbezogen 18](#_Toc516389724)

[2.1.3 Material und Methoden 19](#_Toc516389725)

[2.1.4 Mechanisches Modell des Fahrzeuges 19](#_Toc516389726)

[2.1.5 Ergebnisse 20](#_Toc516389727)

[2.1.6 Konklusion 22](#_Toc516389728)

[3. Simulation 23](#_Toc516389729)

[3.1 Pi Simulation 23](#_Toc516389730)

[3.2 Arduino Simulation 23](#_Toc516389731)

[4. Datenübertragung 23](#_Toc516389732)

[4.1 Thinkspeak 23](#_Toc516389733)

[4.2 Auswertung 24](#_Toc516389734)

[4.3 Twitter-Account 26](#_Toc516389735)

[5. Hindernis-Pacour 28](#_Toc516389736)

[5.1 Karte mit Fahrstrecke 28](#_Toc516389737)

[5.2 Karte mit Messpunkte 28](#_Toc516389738)

[5.3 Fahrstrecke mit Messungen 28](#_Toc516389739)

[6. Es wird doch alles anders als geplant 28](#_Toc516389740)

[7. Zusammenfassung 29](#_Toc516389741)

# ****List of Figures****

[Abbildung 3: Mechanisches Kräftemodell des Autos 41](#_Toc516355433)

[Abbildung 4: Diagramm für die Beschleunigung 42](#_Toc516355434)

[Abbildung 5: Diagramm für die Winkelgeschwindigkeit 43](#_Toc516355435)

# ****List of Tables****

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# ****List of Abbreviation****

# Kurzfassung

# Abstract

# Conecpt Development

Aufgrund der im letzten Semester geschaffenen Basis wurde die „Mohne“ weiterentwickelt. Alternativ gab es noch die Idee ein einfaches Auto zu konstruieren um damit die Anforderungen zu erfüllen. Letztendlich wurde die Entscheidung zugunsten der bereits gebauten Drohen getroffen und wir auf dieser Basis weiterentwickeln werden.

Für das neue Projekt wurden die folgenden Kriterien des Lektoren Teams gestellt:

* Eigenschaften des Fahrzeuges: Fahrmodus, Telemetrie Daten erfassen und übertragen
* Datenerfassung: Umweltbedingungen permanent erfassen
* Digitaler Zwilling in MATLAB: Fahrzeug (mechanisches Modell-Kräfte)
* Simulation: Raspberry Pi und Arduino Simulator
* Tracks: ferngesteuert und autonom

Die obenstehenden Punkte werden im nachfolgenden Bericht detailliert erläutert und darauf eingegangen.

## Konzept der Fahrmodi

Die beiden Konzepte der Fahrmodi bestehen aus den Modi autonom Fahren und ferngesteuertes Fahren.

Bei der Aufgabenstellung autonomes Fahren – Obstacle Detection - sind die Anforderungen folgende:

* Eindeutige markierte Hindernisse müssen umfahren werden
* Die Hindernisse müssen abfotografiert werden
* Es werden keine Begrenzungswände verwenden
* Startpunkt lieg bei (0,0)
* Endpunkt wird vorgegeben
* Erfassen der Daten der verbauten Sensoren via ThingsSpeak und MathLab

Beim ferngesteuerten Fahren - Datenerfassung sind folgende Anforderungen:

* Startpunkt bei (0,0)
* Abfolge von Messfeldern wird je Team vorgegeben
* Hindernisse müssen umfahren werden
* Explorer wird ferngesteuert (Eingabegerät offen; Eingabe von Koordinaten, Steuerbefehle mittels Joystick)
* Erfassen der Daten der verbauten Sensoren via ThingsSpeak und MathLab

### autonom

Um den Parcours autonom zu absolvieren wurden zusätzliche Sensoren verwendet, mit denen diese Aufgabe vereinfacht wird. Zum einen wurde ein Gyroskop angeschlossen um die Orientierung im Raum zu vereinfachen. Zum anderen ein Ultraschallsensor damit Hindernisse erkannt werden können.

Für die Programmierung wurde der Raspberry Pi und der Arduino verwendet. Auf dem Arduino ist die Motorsteuerung gespeichert und auf dem Pi die Kamerafunktion und der Array. Beide Computer kommunizieren mithilfe einer i2c-Verbindung und senden Zahlen hin und her, welche die verschiedene Funktionen auslösen. Das autonome Fahren beim Auto konnte leider nicht komplett absolviert werden, da das Auto selber nicht in einer Linie fährt, sondern eine Rechtsneigung besaß. Es fuhr so lange vorwärts, bis sich ein Hindernis 30 Zentimeter vor dem Fahrzeug befindet. Dieser Wert wird mit einem Ultraschallsensor berechnet. Dieser ist am Arduino angeschlossen und muss dem Raspberry Pi mitteilen, dass ein Hindernis vorhanden ist und ein Foto gemacht werden muss. Durch die i2c-Verbindung wird vom Arduino die Zahl 7 versendet. Das Problem hierbei ist, dass der Arduino nicht von sich aus Zahlen senden kann. Somit muss der Raspberry Pi regelmäßig den Arduino fragen, ob ein Hindernis vorhanden ist. Diese Frage wird mithilfe einer Zahl jede Sekunde gestellt und geantwortet wird mit einen von zwei Zahlen. Entweder ist ein Hindernis vorhanden oder die Bahn ist frei. Nachdem der Raspberry Pi die Antwort erhalten hat, wird entsprechend gehandelt. Bei einem Objekt wird die Kamerafunktion ausgelöst und das Bild wird auf Twitter geladen. Nachdem das Foto geschossen wurde, wird eine Rechtskurve gefahren und dann fährt das Auto weiter.

Die Motorsteuerung wurde aus dem Ersten Projekt im 1. Semester übernommen und angepasst. Um Platz auf dem Breadboard und Pins auf dem Arduino zu sparen wurde nur eine H-Brücke gebaut und die linken bzw. rechten Reifen parallelgeschaltet. Da die Motoren sehr schwach sind, wurde immer der höchste Wert von 255 verwendet. Bei den Links- und Rechtsdrehungen wird wie ein Panzer gefahren. Das heißt, dass beide Seiten gegenfahren bis sich das Auto in einem Winkel von 90 Grad gedreht hat.

### ferngesteuert

Bei der Aufgabe das Fahrzeug über den Parcour per Fernsteuerung zu fahren gibt es viele Möglichkeiten die Fernsteuerung zu kreieren. Wir haben uns dann schlussendlich für die Laptoptastatur entschieden. Dies hat den Vorteil, dass jeder den Code testen kann ohne eine physische Fernsteuerung mitnehmen zu müssen und falls es Probleme bei der Verbindung gibt kann die Mohne per Not-Stopp ausgeschaltet werden. Da das Fahrzeug bereits mit einem Samba Server verbunden ist war die Remote-Steuerung leicht zu verwenden und in der Testphase konnten die verschiedenen Programme schnell verändert werden. Der Code für die Fernsteuerung heißt ‚FernFoto.py‘ und kommt von ‚Fernsteuerung‘ und ‚Foto‘.

Die Motorsteuerung war auf dem Arduino gespeichert. Dies war bei der ursprünglichen Mohne, wie auch bei unserem endgültigen Auto der Fall. Auf die Motorsteuerung selber wurde bereits im Abschnitt Autonom eingegangen.

Der Code für die Fernbedienung wurde auf dem Raspberry Pi mithilfe eines Keyloggers geschrieben. Der verwendete Logger kommt aus die pynput Library. Dies hatte den Vorteil, dass es auch eine Key Release Funktion gab und dadurch die Mohne stoppt, wenn eine Taste losgelassen wurde. Der Nachteil war allerdings, dass der Keyloggers nicht zusammen mit Schleifen funktioniert. Somit konnte der Raspberry Pi nicht selbständig Fotos schießen. Dieses Problem wurde gelöst indem ein separater Tastendruck die Kamera auslöst und das Bild auf Twitter ho geladen wird.

Die Mohne wird klassisch mit den Pfeiltasten gesteuert. Beim Drücken der Leertaste entsteht das Foto und mit der Escape-Taste wird das Programm gestoppt. Wenn keine Taste gedrückt wird oder eine losgelassen wird werden die Motoren ausgeschaltet.

Jetzt müssen die Befehle nur noch an den Arduino geschickt wird, damit diese umgesetzt werden. Die im Unterricht gelernte i2c-Verbindung wurde verwendet und modifiziert. Jede Taste wurde einer Zahl zugeordnet, die an den Arduino geschickt wird. Wenn die Motoren stoppen sollen wird auch eine Zahl gesendet.

Der Code auf dem Arduino heißt ‚Fern\_1.0‘. Auf diesem wurden bereits Klassen für die jeweiligen Richtungen geschrieben. Also vorwärts, rückwärts, rechts und links. Dazu natürlich noch die Stopp Funktion. Durch die i2c-Verbindung wartet der Arduino auf die Zahlen, welche vom Raspberry Pi gesendet werden. Bei der Zahl 1 wird nach vorne gefahren. Bei der 2 nach rechts. Wird die 3 gesendet nach Links und bei der 4 rückwärts. Die Nummer 5 wird für die Stopp Funktion verwendet.

Die Fernsteuerung lief trotz vielen Schnittstellen sehr präzise und ohne Verzögerung.

## Fahrzeugaufbau

### CAD als Unterstützung für die Fertigung und Designentwicklung

Die Mohne besteht aus 66 Einzelteilen, Kabel, Schrauben, Muttern und Kabelbinder sind hierbei nicht berücksichtigt.

In Abbildung 1 ist das fertige CAD Modell als Explosionsdarstellung aufgezeigt, um die Komplexität des Modells zu veranschaulichen.

Ohne ein ausgereiftes CAD-Modell ist eine ordnungsgemäße Fertigung der Mohne nicht möglich. Außerdem hilft das digitale Modelle etwaige Lücken bzw. Versäumnisse in der Konzeption aufzudecken, noch bevor eine zeitintensive Fertigung gestartet wird.



Abbildung 1: Explosionsdarstellung Mohne



Abbildung 2: Rendering Ansicht Mohne

Abbildung 2 zeigt ein Rendering Bild der auskonstruierten Mohne. Im Zuge der Designentwicklung wurden folgende vier Designschwerpunkte hinsichtlich der NASA als Kunde getroffen:

1. Entwicklung eines individuellen und additiv gefertigten Gehäuses zum Schutz der Steuergeräte
2. Modulares Sensorplattenkonzept, welches einen Wechsel durch reine Schraub und Steckverbindungen ermöglicht
3. Einsatz von Kohlefaser als modernes, vielseitiges und gewichtsoptimiertes Material
4. Solarmodule zum Laden des Akkumulators

Das gesamte CAD Modell und das Designkonzept kann unter <https://github.com/gchrizZz/Mohne> abgerufen werden.

### Elektronik

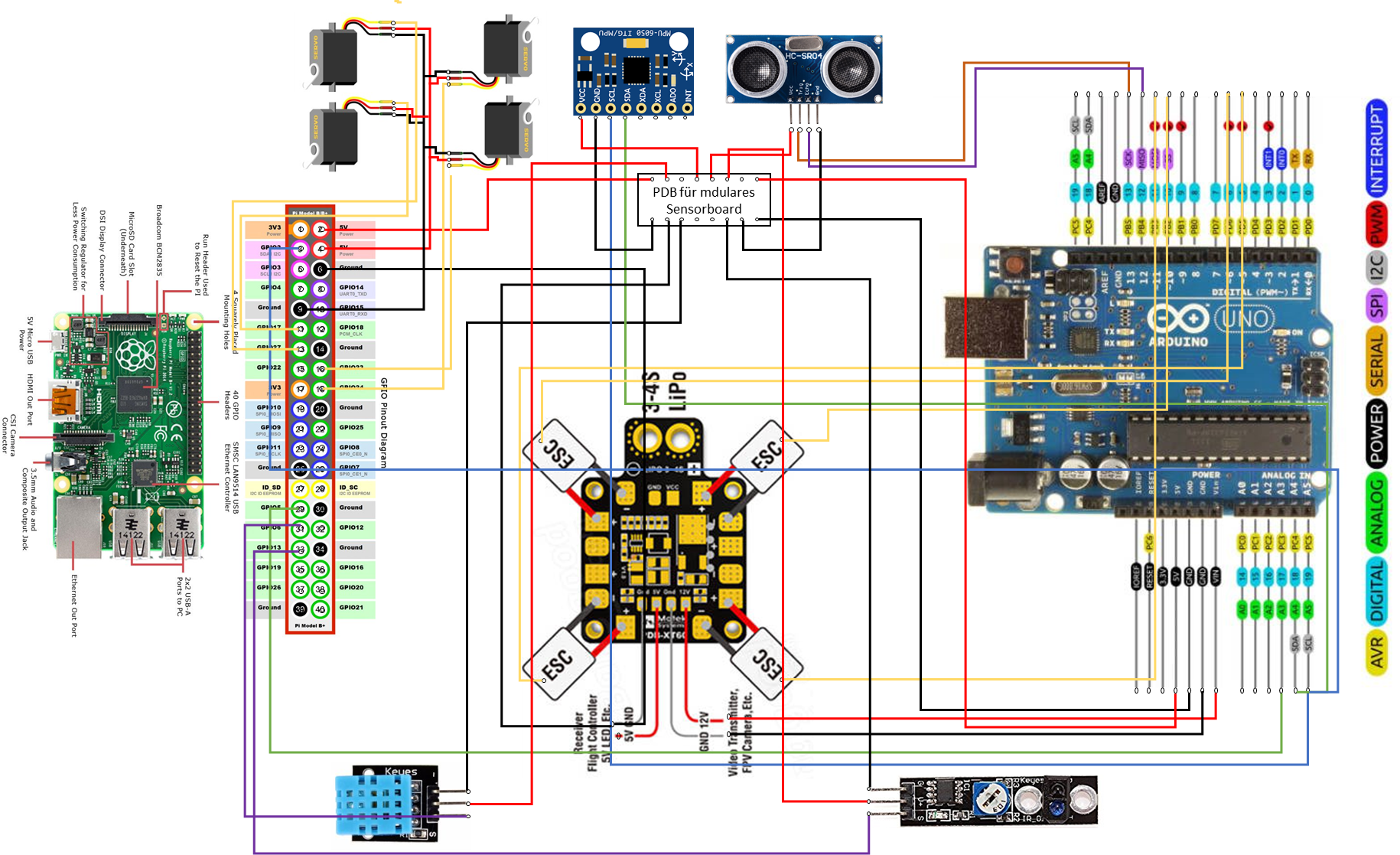


Abbildung 3: Elektronische Komponenten und Verkabelung Mohne

Abbildung 3 zeigt den Kabelbaum und die elektrischen Komponenten der Mohne auf. Dieser Plan berücksichtigt lediglich die geforderten Sensoren, dennoch ist die Designentscheidung einer modularen Sensorplatte bereits berücksichtigt. Hierfür wurde ein gesondertes PDB ausschließlich für die Stromversorgung aller angebrachter Sensoren eingearbeitet. Somit müssen zusätzliche Sensoren lediglich an diesem PDB und mit Ihren Signalkabeln an das vorgesehene Steuergerät (Pi oder Uno) angeschlossen werden. Löten ist nicht mehr notwendig.

### Kommunikation

Der Zugriff der Mohne wird nach wie vor über einen Samba Server realisiert. Samba ist ein freies Programmpaket, das es ermöglicht Windows-Funktionen wie die Datei und Druckdienste unter anderen Betriebssystemen zu nutzen und die Rolle eines Domain Controllers anzunehmen. Es implementiert hierfür unter anderem das Server Message Block/Common Internet File System-Protokoll (SMB/CIFS-Protokoll). Im Projekt wurde Samba als DC und DNS verwendet, damit man via RDP über WLAN auf den Pi zugreifen kann, ohne eine statische IP im Pi hinterlegen zu müssen. Der hostname der Drohne lautet „Mohne“.

Die Kommunikation zwischen dem MPU, dem Arduino Uno und dem Pi wurde via I²C realisiert. I²C (gesprochen "I quadrat C") ist ein synchroner serieller Zweidraht-Bus, der jeweils eine bidirektionale Daten- und Taktleitung verwendet und für die Kommunikation zwischen ICs über kleine Distanzen geeignet ist. Die Bezeichnung steht für IIC, Inter-Integrated Circuit.

Der Kombi-Sensor (KY-015), als auch der Linetracking Sensor (KY-033) sind mittels jeweils eines GPIO Pins mit dem Pi verbunden. Somit können die Daten von diesen direkt über das digitale Eingangssignal am Pi verarbeitet werden.

Da die Logikverarbeitung für die das autonome Fahren hinsichtlich Linetracking und Orientierung in einem Schachbrett-Array auf dem Pi abläuft, kann man mit dieser einfachen und direkten Kommunikation, die Logik am leichtesten realisieren.

Die Servos, welche für die Ausrichtung der Motorhalterung an allen vier Armen der Drohne verantwortlich sind, werden ebenfalls via GPIO Pins an den Pi angebunden. Jedoch wird diesen ein imitiertes PWM Signal gesendet, um sie zu steuern.

Der Ultraschallsensor (USS) ist mittels zwei IO Pins, direkt am Arduino angeschlossen. Auf diesem Weg kann die unmittelbare Hinderniserkennung am schnellsten realisiert werden. Da der USS ausschließlich mit dem Arduino Uno kommuniziert, muss die Logikverarbeitung hierfür im C++ Code auf dem Arduino stattfinden.

# Model Based Analytics

Der Grundgedanke von Model Based Analytics ist die virtuelle bzw. digitale Darstellung eines physischen Modells. Dieses Modell wird in der digitalen Welt analysiert und interpretiert. Dies ist eine Methode gewisse Problemstellungen virtuell darzustellen und diese als Simulation zu verwenden.

Für unser Projekt wurde der Digitale Zwilling gewählt um diese Problemstellung darzustellen und eine entsprechende Lösung zu finden.

## Digitaler Zwilling

Das Konzept eines digitalen Zwillings besteht aus drei Elementen: einem physischen Zwilling in der realen Welt, einem digitalen Zwilling im virtuellen Raum, der sowohl alle alten betrieblichen Daten als auch die Echtzeitdaten des physischen Zwillings abbildet, und die Interkonnektivität, die es den Zwillingen ermöglicht, die Informationen auszutauschen. Digitale Zwillingen können für Simulationen verwendet werden und für Einblicke in den arbeitenden Zwilling, um Prozesse zu überwachen und mögliche Fehler zu entdecken.[[1]](#footnote-1)

Im folgenden Abschnitt werden nähere Beschreibungen zum digitalen Zwilling angeführt.

### Allgemein

Ein digitaler Zwilling ist eine virtuelle Modellbildung zum Beispiel eines Prozesses, eines Produkts oder einer Dienstleistung, welches die reale und virtuelle Welt verbindet. Digitale Zwillinge verwenden reale Daten von installierten Sensoren, welche die Arbeitsbedingungen oder Position von Maschinen repräsentieren. Diese Kopplung der virtuellen und realen Welten ermöglicht die Analyse von Daten und die Überwachung von Systemen, um Probleme verstehen und bearbeiten zu können, bevor sie überhaupt auftreten. Ausfallzeiten können frühzeitig vermieden werden und dadurch neue Chancen entwickeln und mithilfe von Computersimulationen, damit die Zukunft besser geplant werden kann. Heutzutage werden digitale Zwillinge eine geschäftliche Notwendigkeit, weil der gesamte Lebenszyklus eines Produkts, Prozesses oder Geschäftsmodells abdeckt werden kann und somit die Grundlage für verbundene Produkte und Dienstleistungen bilden.

Für den sogenannten digitalen Zwilling sind folgende Anforderungen notwendig: das abzubildende reale Objekt, den digitalen Zwilling im virtuellen Raum und Informationen, welche die beiden miteinander verbinden. Digitale Zwillinge können dadurch die operative und auch finanzielle Leistungsfähigkeit eines Objekts wie einer Anlage oder einer Dienstleistung verbessern und in Echtzeit dargestellt werden.

In der ersten Phase „Design“ geht es um den Umgang mit komplexen Produktanforderungen, schnellen Entwicklungszyklen und strengen regulatorischen Anforderungen. In der zweiten Phase „Erstellung (manufacturing)“ kann der Zwilling helfen, bessere Effizienz, Qualität und höheren Ertrag in der Erstellung zu erreichen. In der dritten Phase „Nutzung (operate)" kann ein digitaler Zwilling u. A. eingesetzt werden, um die Verfügbarkeit von Objekten (z. B. Maschinen) zu verbessern. In der vierten Phase „Wiederverwertung (Recycling)“ kann der ein digitaler Zwilling z.B. für die Ersatzplanung oder der Eruierung von Upcycling-Potenzialen eingesetzt werden.

### Projektbezogen

Unser PlanB des Projektes Data Explorer wurde am Tag zu einem Auto umgebaut und deren Funktionen erweitert bzw. ergänzt. Am Auto herrscht ein mechanisches Kräftemodell, welches im folgenden Kapitel dargestellt wird. Dadurch haben wir entschieden, dass wir die Kraft des Motors mit der Beschleunigung als Digitalen Zwilling modellieren bzw. darstellen wollen. Interessant an der Darstellung wird, wie sich die Beschleunigung „a“ während der Datenerfassung verhält. Die Erfassung der Daten erfolgt mithilfe der Matlab Software und dessen Mobile App. Durch die Kommunikation zwischen den beiden Anwendungen kann eine Erfassung der Daten mit verschiedenen Sensoren wie Beschleunigungssensor erfasst werden. Es kann analysiert werden, ob bei der Absolvierung des vorgegebenen Parcours eine Verminderung der Beschleunigung eintritt, wenn ein Hindernis kommt oder eine Kurve nach links oder rechts gemacht wird. Der digitale Zwilling wird in einem Plot der Matlab Software dargestellt, damit das Diagramm entsprechend erläutert und interpretiert werden kann.

### Material und Methoden

Für die Erfassung des digitalen Zwillings unseres Sommerprojektes haben wir uns entschieden, dass wir die Anwendung der Matlab App verwenden werden, da wir kein Logfile über den Raspberry Pi erstellen. Für die Anwendung ist die Matlab Software und der Download der dazugehörigen Matlab Mobile App notwendig. Zu Beginn wird die Matlab Mobile App am Smartphone installiert, egal ob Android oder iOS Betriebssystem. Am Smartphone können die unterschiedlichsten Sensoren wie Beschleunigungs-, Magnetfeld-, Positions-, Winkelgeschwindigkeits- und Orientierungssensor aktiviert werden. In der Software ist eine Installation eines Add-Ons zu erledigen. Um eine Verbindung zwischen der Software und der App herzustellen, wird in der Kommandozeile der Software der Konnektor aktiviert. Es wird nun die funktionierende IP-Adresse angezeigt, welche im Smartphone eingegeben wird, um eine Kommunikation herstellen zu können. Als nächsten Schritt wird der Link von PC und Smartphone erzeugt und es können sowohl am Smartphone und in der Kommandozeile durch bestimmte Matlabbefehle die Sensoren aktiviert werden. Für eine korrekte Datenerfassung werden durch Befehle in der Kommandozeile die Datenaufnahme der Messwerte aktiviert und auch wieder beendet. Mit den erfassten Sensordaten des Smartphones können die Daten auf verschiedensten Arten analysiert und auch dargestellt werden. Die Daten können in einem Diagramm inklusiver Legende in Matlab dargestellt werden.

### Mechanisches Modell des Fahrzeuges

Im folgenden Bild wird das mechanische Modell unseres Fahrzeuges abgebildet. Das Kräftemodell des Fahrzeuges besteht aus der Reibungskraft, die Erdanziehungskraft, die Kraft des Motors in der Vorwärtsbewegung und die Kraft des Autos.

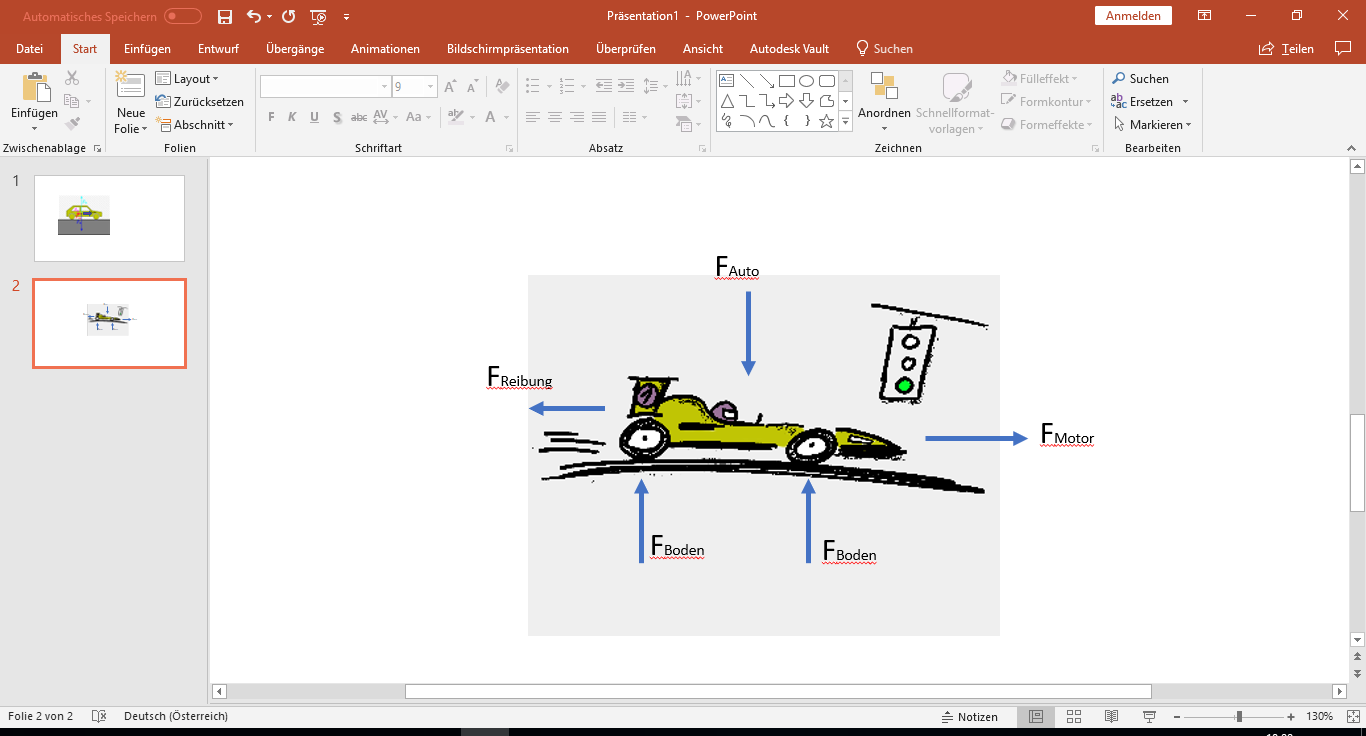


Abbildung 4: Mechanisches Kräftemodell des Autos

### Ergebnisse

Aus der Datenerfassung der Matlab App konnte die Kraft des Motors erfasst und gemessen werden. Die Daten wurden während unseres autonomen Fahrens, welches leider in der kurzen Zeit nicht möglich war fertig zu programmieren, aufgenommen. Anhand der Daten, die auf die Software von Matlab übertragen wurden kann herausgelesen werden, dass unser Programm eine relativ kontinuierliche Geschwindigkeit durchfährt. Daraus resultiert sich, dass unser Fahrzeug, wegen den nicht fertigen ferngesteuertem Programm, nicht immer Objekte bzw. Hindernisse erkennen kann und somit sich nicht mit einer kontinuierlichen Geschwindigkeit fortbewegt.

Mit der Darstellung der Beschleunigung ist keine kontinuierliche Beschleunigung ersichtlich. Der programmierte Code enthält leider keine konkrete Struktur bzw. Abfolge und somit resultiert sich ein relativ hektisches Beschleunigungsdiagramm. In der Mitte des Diagramms kann man erkennen, dass das Fahrzeug einen Stillstand hat und womöglich ein Hindernis erkennen konnte.

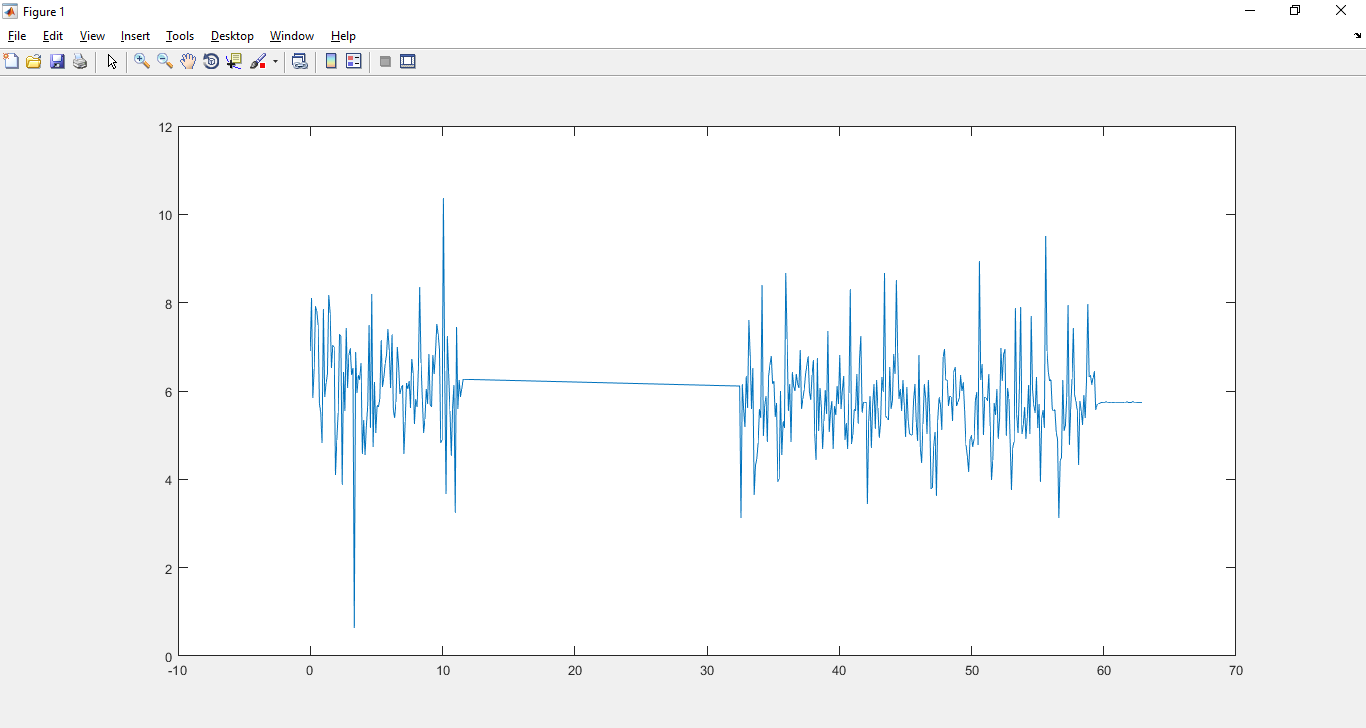


Abbildung 5: Diagramm für die Beschleunigung

Im folgenden Plot von Matlab ist die Winkelgeschwindigkeit dargestellt. Auf dieser Darstellung erkennt man wie sich das Fahrzeug in den Richtungen verändert. Wie schon vorher erläutert haben wir den Fernsteuerungsmodus leider nicht bis zum Ende programmieren können. Im Diagramm ist schön zu sehen, wie oft und schnell das Fahrzeug die Richtung ändert. Das ist der Grund, dass das Programm nicht in der Zeit fertig geworden ist. Deswegen fährt das Fahrzeug willkürlich durch den Raum und in der Gerade des Diagramm hat es an gehalten, weil es womöglich ein Hindernis erkannt hat. Ursprünglich ist es gedacht, dass das Fahrzeug einem Hindernis ausweicht, jedoch nach dem Diagramm sind auf unserem vorgegebenen Parcour sehr viele Hindernisse platziert worden.

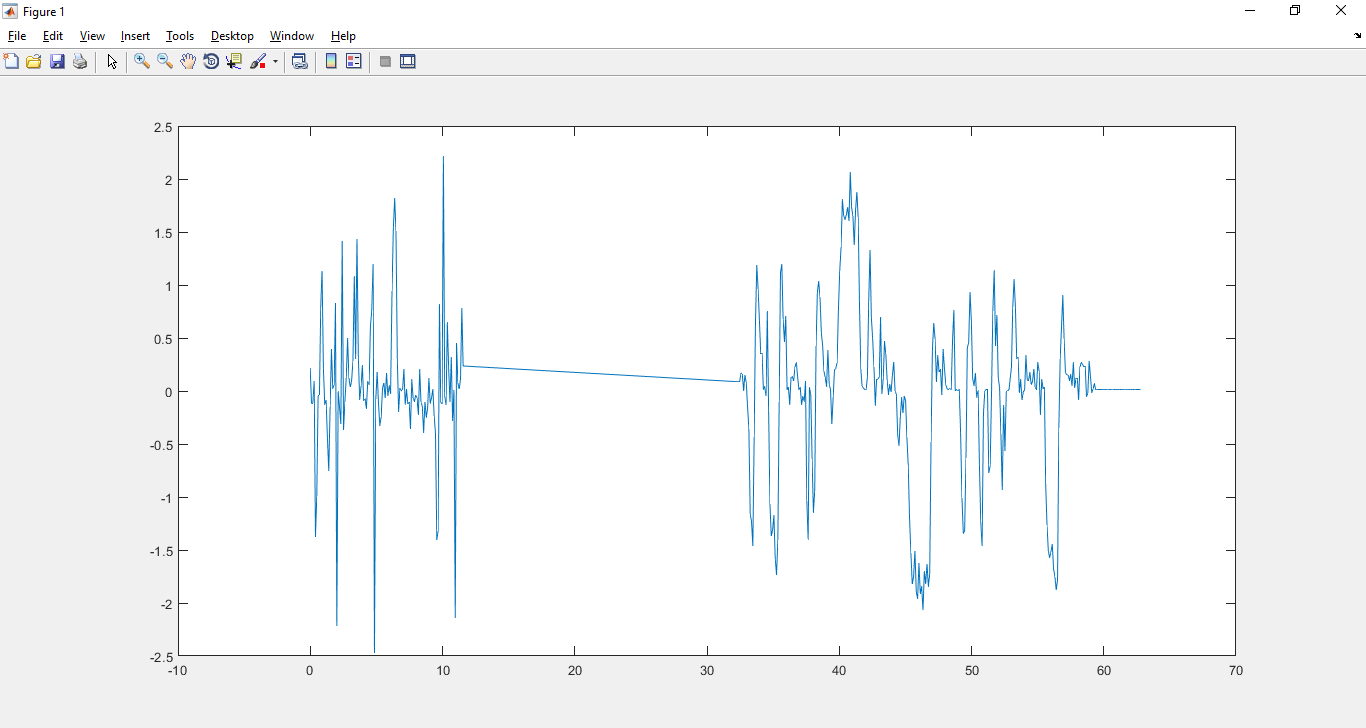


Abbildung 6: Diagramm für die Winkelgeschwindigkeit

### Konklusion

Am Beispiel der beiden Diagramme kann man erkennen, dass unser programmierter Quellcode leider nicht dem Fernsteuerungsmodus entspricht. Das Fahrzeug kann durch die Bilderkennung nicht kontinuierlich die Hindernisse erkennen und daraufhin die Geschwindigkeit reduzieren bzw. nach dem Stillstand wieder beschleunigen. In der Mitte des Diagramms ist ein Stillstand zu erkennen, weil hier womöglich ein Hindernis erkannt wurde. In beiden Abbildungen ist die Beschleunigung sowohl die Winkelgeschwindigkeit sehr hektisch, da das Programm aus zeitlichen Gründen nicht fertig geschrieben werden konnte.

# Simulation

## Pi Simulation

## Arduino Simulation

# Datenübertragung

## Thinkspeak

Der zentrale Teil der Anforderungen für das Projekt war die Übertragung von Messdaten der Fahrzeuge über das Web hin zu einer Cloudplattform, hier ThingSpeak. ThingSpeak ist eine Open IoT Platform (Cloudservice), die im Zusammenspiel mit MATLAB eine Menge Funktionen bietet, darunter auch die Übertragung von Messdaten. Dabei waren gewisse Parameter vorgegeben, andere wiederum waren, in einem bestimmten Rahmen, frei wählbar. Für die Übertragung der Daten kam sowohl das HTTP/REST-Protokoll als auch das MQTT-Protokoll in Frage. Die Entscheidung fiel zugunsten des MQTT-Protokolls, das speziell für embedded devices mit geringer RAM- und CPU-Leistung. Darüber hinaus verbraucht MQTT sehr viel weniger Bandbreite und eignet sich daher optimal für die Verwendung mit Arduino und Raspberry Pi. Zusätzlich gibt es die Möglichkeit, Nachrichten mit unsecured TCP zu senden, womit nochmal die Performance verbessert wird.

Um das Protokoll für die Übertragung der Messdaten zu nutzen, ist es erforderlich, eine entsprechende Library auf dem Raspberry Pi zu installieren, in diesem Fall die Paho Library.

In dem dazugehörigen Code <mqtt\_Telemetrie\_ThingSpeak\_V3> sind einige Funktionen derzeit auf „False“ gesetzt, da sie einerseits die Übertragung der entsprechenden Nachricht auf anderen Wegen ermöglichen (statt UnsecuredTCP zum Beispiel UnsecuredWebsockets) oder das Ansprechen eines zweiten Channels auf ThingSpeak ermöglichen. Die Verbindung zu ThingSpeak ließ sich im Nachhinein relativ einrichten, vor allem da es auch in einer Übungsveranstaltung im Studienfach Datenübertragung vorgestellt wurde. Das eigentliche Übertragen der Telemetriedaten wird unter Punkt X.X näher beschrieben.

## Auswertung

Nach erfolgreichem Herstellen einer Verbindung zu ThingSpeak sollten nun die geforderten Daten aufgenommen und automatisch an ThingSpeak übertragen werden. Die CPU-Temperatur, RAM-Auslastung, Data Up- und Download werden vom Raspberry Pi selbst erfasst, im Gegensatz zu Luftfeuchtigkeit und Umgebungstemperatur, die über einen Kombisensor (KY-015) ausgelesen werden.

Für diesen Kombisensor muss eine spezielle Library von Adafruit installiert werden, Adafruit\_DHT. Nach der Installation kann der Sensor ausgelesen werden, indem man einen entsprechenden GPIO-Pin definiert. Der Kombisensor gibt dann zuerst den Wert für die Luftfeuchte und dann den Wert für die Umgebungstemperatur aus. Die erfassten Werte für Luftfeuchte und Umgebungstemperatur können den Abbildungen 6 und 7 entnommen werden.

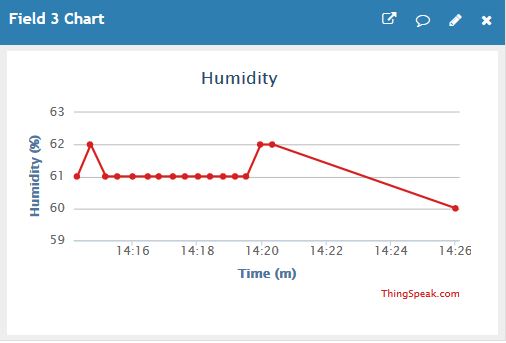


Abbildung 6: Luftfeuchtigkeit

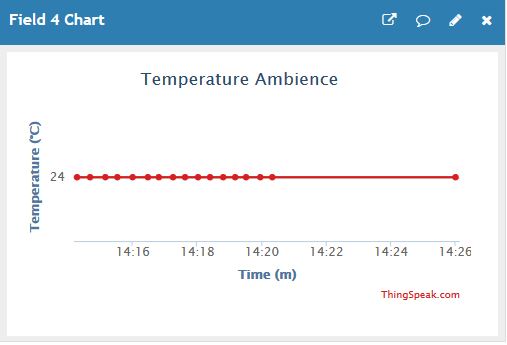


Abbildung 7: Umgebungstemperatur

Für die Übertragung der erfassten CPU-Temperatur und der RAM-Auslastung werden einfache, im Code <mqtt\_Telemetrie\_ThingSpeak\_V3> leicht ersichtliche Befehle verwendet. So ist zum Beispiel folgender Code ausreichend, um die RAM-Auslastung auszulesen:

ramPercent = psutil.virtual\_memory().percent

Bei der Übertragung der CPU-Temperatur musste der ausgelesene Datenstring etwas angepasst werden, damit lediglich ein Zahlenwert an ThingSpeak weitergegeben wird. Ohne diese Anpassung wäre es nicht möglich gewesen, die CPU-Temperatur auszulesen. Den Abbildungen 8 und 9 können die entsprechenden Werte für CPU und RAM entnommen werden.

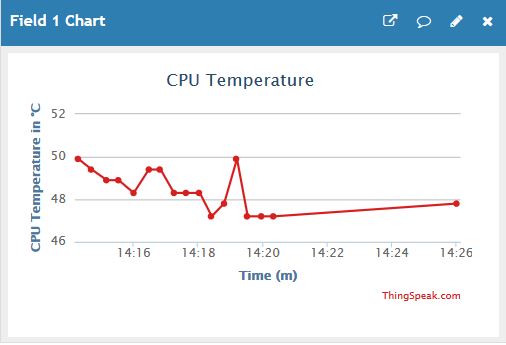


Abbildung 8: CPU-Temperatur

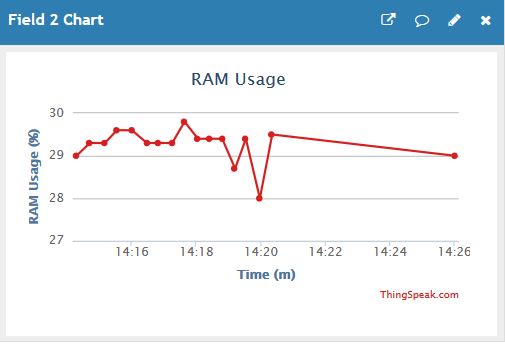


Abbildung 9: Arbeitsspeicher Auslastung

Die Übergabe der Data Up- und Downloads war relativ kompliziert, da der aktuelle Wert durch einen Vergleich zwischen aktuellem Datenübertrag und vorherigen Datenübertrag ermittelt wird. Die erste Messung zeigt also zu Beginn bei beiden Feldern den Wert 0, und ab der nächsten Messung werden die aktuellen Werte angezeigt. Data Up- und Download können den Abbildungen 10 und 11 entnommen werden.

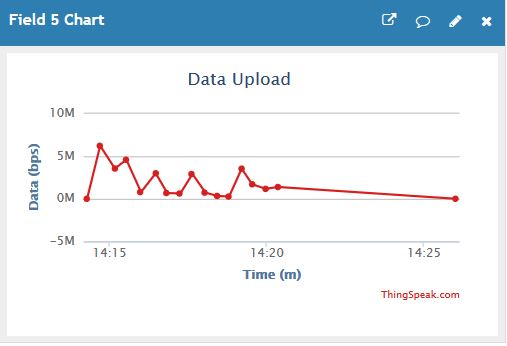


Abbildung 10: Data-Upload

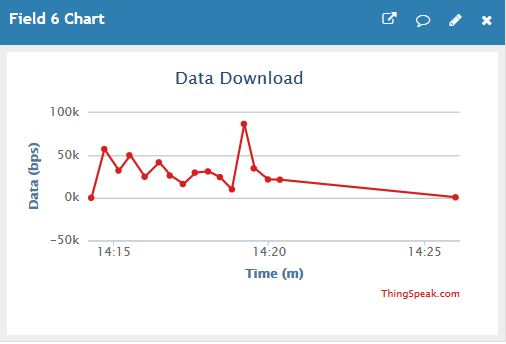


Abbildung 11: Data-Download

## Twitter-Account

Das automatische Hochladen eines aufgenommenen Fotos zu einem Twitteraccount, was zuerst als optionales Ziel für das Gesamtprojekt geplant war, entwickelte sich zum Ende der Projektphase zu einem entscheidenden Bestandteil des Gesamtkonzepts der Datenübertragung. Die Anforderungen des Projekts sahen lediglich die Speicherung der entstandenen Fotos vor, welche jeweils von einem Hindernis gemacht werden sollten.

Um nun die gespeicherten Bilder automatisch an den erstellten Twitteraccount der Mohne (Mohne\_2.0) zu pushen, muss über das Application Management von Twitter (apps.twitter.com) eine passende App erstellt werden. Hier hat, ähnlich wie bei ThingSpeak, die eindeutige Identifikation des Ziels für den Upload-Vorgang höchste Priorität. Über Consumer Key, Consumer Secret und Access Token, die bei erfolgreicher Einrichtung der App automatisch generiert werden, kann der Twitteraccount aus dem Python Code heraus adressiert werden.

Wenn nun der Ultraschallsensor der Hinderniserkennung ein Hindernis erkennt, wird ein Foto gemacht und automatisch zu Twitter gepusht. Das Foto kann auch manuell ausgelöst werden, allerdings nur wenn sich die Mohne im Fernsteuerungsmodus befindet. Ein weiterer Vorteil dieser Methode ist die Auslagerung der Speicherung der Fotos auf Twitter, da immer nur das aktuelle Foto gespeichert und nach erfolgtem Upload überschrieben wird.

Um das Erscheinungsbild der Tweets der Mohne etwas zu polieren, wurden noch einige Statustexte erstellt, die mit einem Hashtag versehen und mit dem entsprechenden Bild zusammen hochgeladen werden. Dabei werden die Statustexte mit der Funktion choice() zufällig ausgewählt.

Letztendlich hat sich das zuerst optionale Ziel, die Bilder automatisch zu Twitter zu pushen, als ein sehr passendes Experiment im Bereich der Datenübertragung erwiesen. Der Aufbau des Twitteraccounts ist auf Abbildung 12 gut zu sehen. Außerdem ist die Funktion aus dem Bericht angefügten Code <FernFoto.py> ersichtlich.

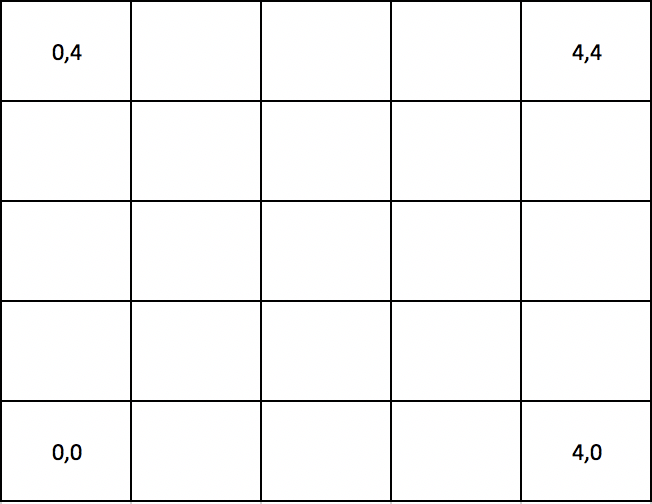


**Abbildung 12** (<https://twitter.com/0mohne?lang=de>)

# Hindernis-Pacour

## Karte mit Fahrstrecke

Das Team-Mohne bekam die Fahrstrecke D zugewiesen. Diese Strecke musste im ferngesteuerten Modus abgefahren werden. Hierbei muss jeder Punkt bzw. jedes Quadrat angefahren und durchfahren werden.



**Abbildung 13: Strecke D**

Für das autonome Fahren wurden Hindernisse auf derselben Karte ohne System verteilt und die Mohne musste das Ziel (4,4) erreichen. Wie die das autonome Fahren funktioniert hat, ist aus dem Code zu entnehmen  
(<https://github.com/gchrizZz/Mohne/tree/master/02_Quellcode>).

## Karte mit Messpunkte

## Fahrstrecke mit Messungen

# Es wird doch alles anders als geplant

Die Generalprobe unseres Marsrovers ging leider gehörig schief. Während einem der letzten Tests ist einer der vier ESC (Electrronic Speed Control) kaputt gegangen. Da die Mohne nur mit allen vier ESC fliegen kann, musste hier eine schnelle und unkomplizierte Lösung gefunden werden. Hierzu wurde die physikalische Grundlage des Arudino-Fahrezeuges gewählt. Nun wurde das Design und die notwendigen Komponenten auf das neue Grundgerüste umgebaut und die entsprechenden Programme und Codes an die neuen gegebenheiten angepasst. Innerhalb weniger Stunden ist es dem Team-Mohne gelungen ein fahrtüchtiges und einsatzfähiges Fahrezeug zu bauen. Mit diesem neu gestaltenen Fahrezeug wurde in den Wettbewerb gestartet und die Anforderungen umgesetzt.

# Zusammenfassung

1. Vgl. (Hewlett Packard Enterprise Development LP, 2018) [↑](#footnote-ref-1)