# Projekt 2

## Technische Dokumentation

Experte: Reto König
Student: Dominique Hofmann

## 1.1. Inhalt

# 2. Einleitung

In diesem Projekt 2 ging es darum, sich vertieft mit dem Produkt Anki Overdrive auseinanderzusetzen. Dabei lag der Fokus auf der Fragestellung, wie die ferngesteuerten Fahrzeuge mit einem Controller, im Detail, kommunizieren würden und wie sie erkennen können, wo sie sich auf der Strecke gerade befinden. Eine weitere Fragestellung, die sich im Fokus befand, war, ob man diese Streckenstücke selbst herstellen und eventuell sogar ausdrucken könnte. Die gewonnen Erkenntnisse wurden dabei in diesem Dokument festgehalten.

# 3. Bluetooth Low Energy

## 3.1. Ursprung

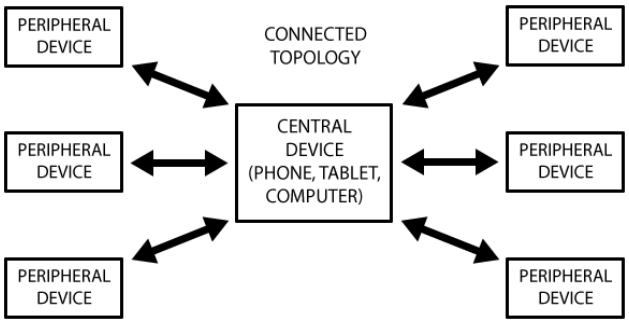
Nokia begann im Jahr 2001 die Entwicklung an einer neuen, drahtlosen Technologie, welche den Bluetooth Standard adaptierte und 2006 unter dem Namen Wibree lanciert wurde. Im Jahr 2009 wurde diese Technologie dann in Bluetooth 4.0 integriert und den Namen Bluetooth Low Energy (BLE) gegeben. Bluetooth LE wurde konzipiert, um einen geringeren Stromverbrauch zu haben und weniger kosten sollte als das «klassische» Bluetooth, dabei aber über einen ähnlichen Kommunikationsbereich verfügen sollte. Dadurch eignet sich Bluetooth LE hervorragend in Einsatzgebieten wie Gesundheitswesen, Fitness, Security und auch im Internet of Things. Erleichtert wird das auch durch die native Unterstützung von Bluetooth in vielen modernen Betriebssystemen wie Windows, MacOS, Linux, iOS und Android. Bluetooth LE und das «klassische» Bluetooth mögen zwar unterschiedliche Protokolle sein, können aber unter Bluetooth 4.0 und neueren Versionen, auf demselben Gerät verwendet werden.

## 3.2. GATT

GATT steht für «Generic Attribute Profile» und definiert, wie zwei Geräte, die Bluetooth LE verwenden, Daten untereinander versenden. Wichtige Konzepte in GATT sind Profile, die Services und die Characteristics. Wichtig ist auch, dass bei GATT eine Server/Client Beziehung geführt wird.

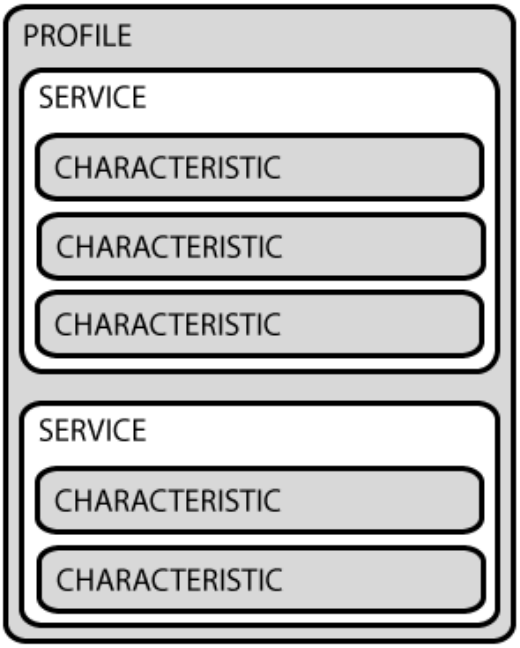
## 3.3. Server/Client Beziehung

Bei einer Verbindung zwischen zwei BLE-Geräten gibt es eine Server/Client Beziehung, wo einer der Client und der andere der Server ist. Im Gegensatz zu anderen Modellen, wo der Server das zentrale Stück ist, ist hier der Client das zentrale Stück in dieser Beziehung. Dabei sendet der Client alle Anfragen an den Server und wartet auf dessen Antwort. Der Server empfängt in dieser Beziehung die Anfragen des Clients und sendet entsprechende Antworten zurück. Diese Beziehung ist in der untenstehenden Illustration gut sichtbar.




## 3.4. Profil, Service und Characteristics

Alle Anfragen, GATT Transactions, sind auf Profil, Services und Characteristics aufgebaut, wie man in der untenstehenden Abbildung gut sehen kann.




Profile existieren nicht auf den BLE-Geräten selbst, sondern sind eine vordefinierte Ansammlung von Services, die vom Standardisierungsinstitut Bluetooth SIG oder den Herstellern selbst definiert wurden.
Wie man in der oberen Abbildung sehr gut sehen kann, kann jedes Profil mehrere Services besitzen, aber jeder Service kann sich nur in einem Profile befinden.

Services werden verwendet, um die Daten in noch kleinere logische Einheiten aufzuteilen, welche dann Characteristics genannt werden. Dabei kann ein Service wieder über mehrere Characteristics verfügen, ein Characteristics kann jedoch nur einem Service angehören. Die Services unterscheiden sich untereinander mit der Hilfe einer sogenannten UUID (Universally unique identifier). Dabei kann die UUID entweder aus 16-Bit, für offiziell aufgenommene BLE Services, oder 128-bit, für massgeschneiderte Services.

Eine Characteristic ist die kleinste logische Einheit in einer GATT Transaktion und enthalten die entsprechenden Daten der Übermittlung. Wie auch bei den Services unterscheiden sich die Characteristics voneinander mit einer UUID, die entweder 16- oder 128-bit sein kann. Man kann jene Characteristics verwenden, welche von Bluetooth SIG standardisiert wurden oder seine eigene massgeschneiderten verwenden.

## 3.5. Advertising und Discovery

Damit ein BLE-Client herausfinden kann, ob es BLE-Server gibt, mit denen es sich verbinden könnte, muss ein Discovery und Advertising durchgeführt werden. Dabei führt der Client das Discovery durch und der Server sendet ein Advertising-Paket. Der Server sendet seine Advertising-Pakete per Broadcast in einem fixen Intervall ab, um mögliche Clients finden zu können. Wenn der Client ein Discovery durchführt und auf ein Advertising-Paket eines Servers stösst, stellt der Client eine Verbindung her. Sobald der Server eine Verbindung mit einem Client hat, stoppt es das Advertising, da ein Server nur mit einem Client verbunden sein kann.

# 4. NodeJS

## 4.1. Definition

NodeJS ist eine Open-Source-JavaScript-Laufzeitumgebung, welches einem erlaubt JavaScript ausserhalb eines Webbrowsers auszuführen und auf Chromes V8 JavaScript Engine aufbaut. Dabei verwendet NodeJS ein Eventbasiertes, Non-Blocking I/O Modell, welches es sehr lightweight und effizient macht.

## 4.2. Node Module

Node Module sind wiederverwendbare Codeblöcke, die dabei so aufgebaut sind, dass sie keinen versehentlichen Einfluss auf anderen Code haben können. NodeJS bietet eine Vielzahl an Node Modulen an, die bereits eingebaut sind und ohne weitere Installation verwendet werden können. Man kann auch eigenen Node Module erstellen und diese in seinen Applikationen verwenden.

## 4.3. Noble

Noble ist ein Node Modul, welches die Kommunikation mit BLE-fähigen Geräten ermöglicht. NodeJS Apps, welche dieses Node Modul verwenden, agieren hierbei als Bluetooth LE Client. Für das Implementieren eines Bluetooth LE Servers gibt es ein anderes Node Modul namens Bleno. Seit dem 2ten Februar 2018 gab es keinen neuen offiziellen Release von Noble mehr. Daher ist es zu Empfehlen die Weiterführung von Noble zu verwenden, welche den Namen «abandonware/noble» hat und noch immer von einer Community gepflegt, welche auch immer neue Releases veröffentlichen.

# 5. Anki Overdrive

## 5.1. Übersicht

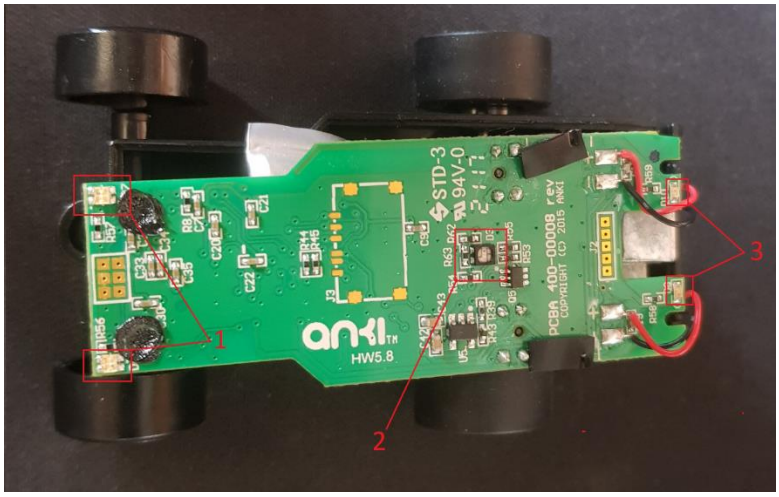
Anki Overdrive ist ein Produkt, welches aus ferngesteuerten Fahrzeugen und einem modularen Streckensystem besteht. Die Fahrzeuge gibt es dabei in unterschiedlichen Ausführungen, wie Rennwagen, Trucks oder Lastwagen ausserdem gibt es unterschiedliche Streckenteile, wie Geraden, Zielgeraden, Kurven und Kreuzungen. Die Streckenstücke können dabei in beliebigster Anordnung zusammengesteckt werden und somit eine Rennstrecke bilden. Die Fahrzeuge verfügen über eine Bluetooth LE Schnittstelle, über welche eine Verbindung aufgebaut werden kann und über welche sie dann gesteuert werden können.

## 5.2. Fahrzeuge

Ein Fahrzeug in Anki Overdrive verfügt über eine Bluetooth LE Schnittstelle, über welche es Befehle von einem BLE-Client erhalten kann. Ein paar Beispiele für Befehle sind das

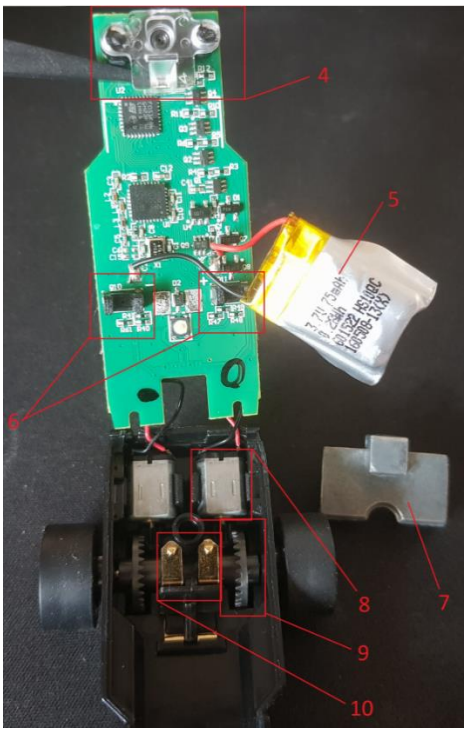
Anpassen der Geschwindigkeit auf der Strecke, das Wechseln der Spur, das Senden eines Pings mit einer entsprechenden Antwort und noch viele weitere Befehle.

Unter der Haube verfügen die Fahrzeuge über entsprechende Komponenten, welche es ihnen erlauben auf der Strecke zu bleiben, zu beschleunigen und mit BLE-Clients zu kommunizieren. Auf der Abbildung unten kann man das Innenleben eines entsprechenden Anki-Overdrive-Fahrzeuges sehen.




Wie auf dem oberen Bild, an den Markierungen sehen kann, gibt es verschiedene LEDs auf der oberen Seite des Fahrzeuges. Die LEDs bei (1) stehen für die Lichter auf der Motorhaube, die LEDs bei (3) stehen für die Hecklichter und das LED bei (2) ist ein Statuslicht, dass den Ladestatus der Batterie anzeigt.

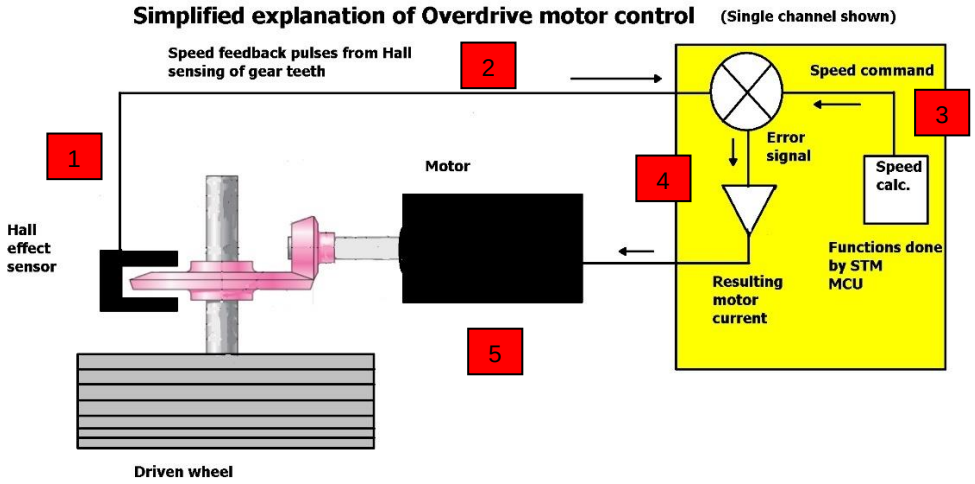
In dem untenstehenden Bild sieht man untere Seite des Fahrzeuges.




Auf der oberen Abbildung sieht man die nummerierten Komponenten auf der Unterseite des Fahrzeugs. Die Komponenten lauten wie folgt, Optischer Sensor (4), Lithium-Ionen-Batterie (5), Halleffekt Sensoren (6), Ballastgewicht (7), Motor (8) und das Getriebe (9).

### 5.2.1. Motorsteuerung

Wie man vielleicht bereits, aus den oben genannten Komponenten, schliessen kann, gibt es keinen herkömmlichen Mechanismus, um die Richtung zu beeinflussen. Stattdessen werden zwei kleine Elektromotoren (8) verwendet, die, über ein Getriebe (9) je eines der Hinterräder antreiben, um die Richtung des Gefährtes zu ändern. Dabei wird, bei einem der Räder, jeweils die Geschwindigkeit minimal beschleunigt oder verringert, um das Fahrzeug in eine gewünschte Richtung zu steuern. Damit die Motoren genau die gewünschte Drehzahl und somit Geschwindigkeit erreichen und beibehalten können, benötigt das Fahrzeug ein Regelungssystem. Ein wichtiger Bestandteil sind, wie oben genannt, die Halleffekt Sensoren (6), welchen die wichtigen Informationen zur Regelung der Motoren liefern.




Anhand der oberen Abbildung kann man sehen wie die Motorregelung funktioniert:

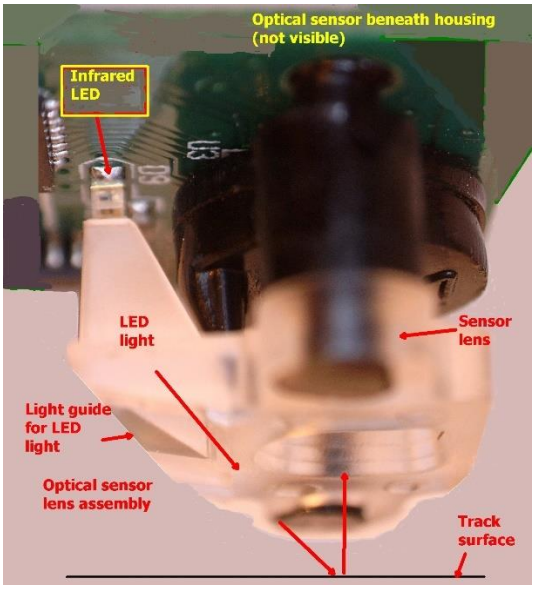
1. Jedes Mal, wenn ein Zahnrädchen des Getriebes beim Halleffekt Sensor vorbeigeht, wird ein Impuls generiert, das vom Sensor erkennt wird. Diese Information sagt aus, was die eigentliche Geschwindigkeit des Motors ist.
2. Diese Information wird dann zurück an den STM Motor Controller gesendet.
3. Der Motor Controller vergleicht die erhaltene Geschwindigkeit mit der vorgegebenen Geschwindigkeit und errechnet daraus eine Differenz, was ein Error Signal ergibt.
4. Basierend auf diesem Error Signal weiss der Motor Controller, ob es mehr Strom benötigt oder weniger, ob es den Motor beschleunigen oder abbremsen muss.
5. Der Motor passt seine Geschwindigkeit, anhand der neuen Befehle des Motor Controllers, an und über das Getriebe wird das Rad entweder beschleunigt oder gebremst. Gleichzeitig misst der Halleffekt Sensor wieder Impulse des Getriebes und sendet diese Information zurück zum Motor Controller, womit der ganze Ablauf wieder von vorne beginnt.

### 5.2.2. Optischer Sensor

Eine weitere interessante Komponente ist der optische Sensor, über welche jedes Fahrzeug verfügt. Dabei besteht der Sensor aus drei weiteren Komponenten, einem

Infrarot LED, einem optischen Sensor Chip und einer Ansammlung von Linsen. Die Linsen haben dabei zwei verschiedene Aufgaben, erstens lenken sie das infrarote Licht der LED auf die Strecke und zweitens fokussieren sie den Sensor Chip, damit es den beleuchteten Streckenteil analysieren kann.

Unterhalb sieht man noch eine detaillierte Nahaufnahme des optischen Sensors, mit entsprechenden Erklärungen.




## 5.3. Kommunikation

### 5.3.1. Bluetooth LE Advertising Pakete

Um mit einem möglichen Bluetooth LE Client zu kommunizieren verwenden die Anki-Fahrzeuge einen BLE Chip von Nordic Semiconductor.
Um sich mit einem potenziellen BLE-Client verbinden zu können, senden die Fahrzeuge Advertising Pakete aus, welche folgendes Format haben.

```
typedef struct anki_vehicle_adv {
    uint8_t                 flags;
    uint8_t                 tx_power;
    anki_vehicle_adv_mfg_t  mfg_data;
    anki_vehicle_adv_info_t local_name;
    uuid128_t               service_id;
} anki_vehicle_adv_t;
```

Dieses Bild wurde der offiziellen Dokumentation der Anki-Overdrive-SDK entnommen. Hierbei kann man sehen, dass das Advertising Paket ausfolgenden Attributen besteht.

- Flags: EIR Flag

- Tx_power: transmission power
- Mfg_data: Die Daten des MANUFACTURER_DATA Teils des Advertising Pakets, welche ein unique identifier des Fahrzeugs enthält
- Local_name: Die Daten des LOCAL_NAME Teils des Advertising Pakets, welche Informationen über Status, Version und Name des Fahrzeugs enthalten
- Service_id: Die Service UUID des Anki-Overdrive-Fahrzeugs

Der Manufacturer_Data Teil ist ein 64 Bit grosser Wert, der jedes Anki-Fahrzeug eindeutig identifiziert und aus dem man folgende Eigenschaften herauslesen kann.

- Identifier: Ein 32 Bit grosser Wert, der für jedes Fahrzeug einzigartig ist
- Model_id: Ein 8 Bit grosser Wert, der ein Fahrzeug zu einem bestimmten Anki-Fahrzeug-Modell zuweist
- _reserved: Ein 8 Bit grosser Wert, der reserviert ist
- Product_id: Ein 16 Bit grosser Wert, welcher ein Fahrzeug als Anki Drive Hardware identifiziert

Laut den Bluetooth Spezifikationen kann der LOCAL_NAME Wert bis zu 248 Bytes lang sein. Bei Bluetooth LE dürfen jedoch nur Werte, die bis zu 20 Bytes lang sind, im Advertising Paket mitgesendet werden und daher ist dieser Wert bei Anki Overdrive auch nur bis zu 20 Bytes gross. Aus dem LOCAL_NAME Wert können folgende Informationen herausgelesen werden.

- State: Der aktuelle Status des Fahrzeugs, der wiederum folgende Werte haben kann
  - Full_battery: Die Batterie des Fahrzeugs ist voll aufgeladen
  - Low_battery: Die Batterie des Fahrzeugs hat wenig Ladung übrig und wird nächstens keine Energie mehr haben
  - On_charger: Das Fahrzeug befindet sich auf dem Ladegerät und ist gerade am Aufladen
- Version: Die aktuelle Firmware Version, die auf dem Fahrzeug gerade am Laufen ist
- Name: Name, welcher vom Benutzer festgelegt wurde

Bei der 128 Bit grossen Service UUID ist sehr interessant, dass es im GATT Profil definiert ist und somit jedes Fahrzeug von Anki Overdrive über die gleiche Service UUID verfügt. Dieser Fakt wird in der offiziellen, wie auch in vielen anderen, SDKs verwendet, um die Fahrzeuge von anderen BLE-Peripherals zu unterscheiden.

### 5.3.2. Bluetooth LE Message Pakete

Jede Nachricht, die an ein oder von einem Anki-Fahrzeug gesendet wird, muss eine vorgegebene Struktur einhalten, die wie folgt aussieht.

| size_of(msg_id + payload) | msg_id | payload |
|---|---|---|
| 1 byte | 1 byte | 0-18 bytes |

Da dieses Nachrichtenformat für Bluetooth LE entwickelt worden ist, ist die maximale Grösse auf 20 Bytes beschränkt. Dabei müssen die ersten beiden

Bytes aus der Grösse der Nachricht und einer Message ID bestehen. Die restlichen 18 Bytes sind für die Daten reserviert, welche an das Fahrzeug oder von dem Fahrzeug gesendet werden können. Bei Anki Overdrive identifiziert die Message ID eine bestimmte Anfrage an das Fahrzeug oder eine bestimmte Antwort eines Fahrzeuges.

In den beiden untenstehenden Tabellen sind die bisher bekannten Anfragen -und Antworten typen aufgelistet

| Nachrichtentyp Anfrage | ID |
|---|---|
| ANKI_VEHICLE_MSG_C2V_DISCONNECT | 0x0d |
| ANKI_VEHICLE_MSG_C2V_PING_REQUEST | 0x16 |
| ANKI_VEHICLE_MSG_C2V_VERSION_REQUEST | 0x18 |
| ANKI_VEHICLE_MSG_C2V_BATTERY_LEVEL_REQUEST | 0x1a |
| ANKI_VEHICLE_MSG_C2V_SET_LIGHT | 0x1d |
| ANKI_VEHICLE_MSG_C2V_SET_SPEED | 0x24 |
| ANKI_VEHICLE_MSG_C2V_CHANGE_LANE | 0x25 |
| ANKI_VEHICLE_MSG_C2V_CANCEL_LANE_CHANGE | 0x26 |
| ANKI_VEHICLE_MSG_C2V_SET_OFFSET_FROM_ROAD_CENTER | 0x2c |
| ANKI_VEHICLE_MSG_C2V_TURN | 0x32 |
| ANKI_VEHICLE_MSG_C2V_LIGHTS_PATTERN | 0x33 |
| ANKI_VEHICLE_MSG_C2V_SET_CONFIG_PARAMS | 0x45 |
| ANKI_VEHICLE_MSG_C2V_SDK_MOD | 0x90 |

| Nachrichtentyp Antwort | ID |
|---|---|
| ANKI_VEHICLE_MSG_V2C_PING_RESPONSE | 0x17 |
| ANKI_VEHICLE_MSG_V2C_VERSION_RESPONSE | 0x19 |
| ANKI_VEHICLE_MSG_V2C_BATTERY_LEVEL_RESPONSE | 0x1b |
| ANKI_VEHICLE_MSG_V2C_LOCALIZATION_POSITION_UPDATE | 0x27 |
| ANKI_VEHICLE_MSG_V2C_LOCALIZATION_TRANSITION_UPDATE | 0x29 |
| ANKI_VEHICLE_MSG_V2C_LOCALIZATION_INTERSECTION_UPDATE | 0x2a |
| ANKI_VEHICLE_MSG_V2C_VEHICLE_DELOCALIZED | 0x2b |
| ANKI_VEHICLE_MSG_V2C_OFFSET_FROM_ROAD_CENTER_UPDATE | 0x2d |
| ANKI_VEHICLE_MSG_V2C_CAR_INFO | 0x3f |

### 5.3.3. Verbindung zum Fahrzeug herstellen

Um eine Verbindung zu einem Anki-Fahrzeug herstellen zu können, muss die Adresse des Gerätes oder ein einzigartiger Hardware identifier bekannt sein. Diese Information kann, während dem Scannen nach Geräten erworben und später für das Verbinden verwendet werden. Der genaue Verbindungsvorgang sieht wie folgt aus:

- Eine Verbindung, von einem BLE-Client, zu einem Anki-Fahrzeug, via einer Hardware Adresse, aufbauen.
- Auf die GATT Services des Fahrzeuges zugreifen, welche wiederum die Characteristics beinhalten, mit denen Daten gelesen und gesendet werden können.
- Sich bei den Gefundenen Characteristics registrieren, um Benachrichtigungen, bei erhaltenen Daten zu erhalten und Daten senden zu können.

Nach dem erfolgreichen Durchführen der oberen Schritte können Daten an das Fahrzeug gesendet und Daten empfangen werden, die vom Fahrzeug gesendet wurden.

## 5.3.4. Befehle an das Fahrzeug

In den nachfolgenden Abschnitten werden die einzelnen Befehle, die an ein Fahrzeug gesendet werden können, genauer erläutert. Um die Veranschaulichung zu vereinfachen, wurden die Beispiele mit der Programmiersprache NodeJS geschrieben.

### 5.3.4.1. SDK Mode an/ausschalten

Der SDK-Modus ermöglicht das Senden von Nachrichten, ohne vorher zusätzliche Daten oder Overhead senden zu müssen, wie es bei der Anki-App bei iOS und Android der Fall ist. Unterhalb ist noch ein Beispiel einer SDK Nachricht zu sehen.

```
message = new Buffer(4);
message.writeUInt8(0x03, 0);
message.writeUInt8(0x90, 1);
message.writeUInt8(0x01, 2);
message.writeUInt8(0x01, 3);
vehicle.writer.write(message, true);
```

Wie man bei diesem Beispiel sehen kann, wird ein neuer Buffer erstellt, dem dann eine entsprechende Bytesequenz zugewiesen wird. Wie im oberen Abschnitt erklärt erhalten die ersten beiden Bytes die Grösse der Nachricht und die Message ID, welche die Nachricht identifiziert. Die nachfolgenden Bytes beinhalten den eigentlichen Payload. In dieser Nachricht wäre der Payload ein Integer mit dem Wert 1 oder 0, was respektiv für «true» oder «false» steht, je nachdem, ob man den SDK-Modus an -oder ausschalten will.

### 5.3.4.2. Ping an Fahrzeug senden

Man kann mit diesem Befehl einen Ping an ein Fahrzeug senden. Falls noch eine Verbindung zu diesem Fahrzeug besteht, sendet dieses eine entsprechende Antwort zurück. Die Anfrage und die Antwort auf den Ping verfügen über die Message IDs 0x16 und 0x17. Unterhalb kann man noch ein Beispiel für eine Ping Anfrage sehen.

```
message = new Buffer(2);
message.writeUInt8(0x01, 0);
message.writeUInt8(0x16, 1); // ANKI_VEHICLE_MSG_C2V_PING_REQUEST
```

Bei dieser Nachricht sind die einzigen Informationen, die übermittelt werden, die Grösse der Nachricht und um welche Message ID es sich hierbei handelt. Bei diesem Beispiel muss kein Payload übergeben werden, da es sich um eine Ping Anfrage handelt und allein aus der Message ID herausgelesen werden kann, um was für eine Art von Nachricht es sich hierbei handelt.

#### 5.3.4.3. Geschwindigkeit festlegen

Mit dieser Nachricht kann einem Fahrzeug eine gewünschte Geschwindigkeit und Beschleunigung übermittelt werden, welches das Fahrzeug dann versucht einzuhalten. Die Message ID für diese Nachricht ist 0x24. Unterhalb ist noch ein Beispiel, welches eine solche Nachricht veranschaulichen sollte.

```
message = new Buffer(7);
message.writeUInt8(0x06, 0);
message.writeUInt8(0x24, 1);
message.writeInt16LE(speed, 2);
message.writeInt16LE(accel, 4);
vehicles[device_id]['writer'].write(message);
```

Wie bei allen Nachrichten werden in den ersten zwei Bytes, die Grösse der Nachricht und die Message ID übermittelt. Anschliessend folgt in den restlichen Bytes noch der Payload, wobei es sich hier um die gewünschte Geschwindigkeit und Beschleunigung handelt.

#### 5.3.4.4. Spurenwechsel festlegen

Mit dieser Nachricht wird einem Fahrzeug mitgeteilt, dass es die Spur wechseln soll. Wie weit nach rechts oder links sich die Spur verändert, auf welcher das Fahrzeug fahren soll, hängt von einem Offset ab, der mitgesendet wird. Der Offsetwert repräsentiert dabei die Distanz, in Millimeter, welche die gewünschte Spur von der Mittellinie hat und dieser Wert muss dabei zwischen -70 und +70 liegen. Dieser Werte sind so gesetzt, da die Strasse eine fixe Distanz zum Rand hat und jeder Wert über 70 dazu führen würde, dass das Fahrzeug über die Strasse hinausfährt.
Die Message ID ist hierbei 0x25 und untenstehend ist noch ein Beispiel, wie eine solche Nachricht gebildet wird.

```
message = new Buffer(12);
message.writeUInt8(11,0);
message.writeUInt8(0x25,1);
message.writeInt16LE(250,2);
message.writeInt16LE(1000, 4);
message.writeFloatLE(offset, 6);
vehicles[device_id]['writer'].write(message);
```

Wie immer sind die ersten beiden Bytes die Grösse der Nachricht und die Message ID. Anschliessend folgt der Payload der Nachricht, welcher ausfolgenden Attributen besteht.

- Speed: Mit welcher Geschwindigkeit die Spur gewechselt werden soll
- Acceleration: Mit welcher Beschleunigung die Spur gewechselt werden soll
- Offset: Wie viele Millimeter die Spur von der Mittellinie entfernt sein soll

Der Offset kann dabei einen Wert zwischen -70 und +70 haben, was

### 5.3.4.5. Offset festlegen

Mit dieser Nachricht kann der Offset überschrieben werden, welcher ein Fahrzeug gerade verwendet hat. Das heisst, falls es zu einem Fehler gekommen ist und ein Fahrzeug auf der Mittelinie fährt, aber den Offset -50 anzeigt, kann das entsprechend korrigiert werden. Die Message ID für diese Art von Nachricht ist 0x2c. Untenstehend noch ein Beispiel, wie eine solche Nachricht gebildet werden kann.

```
message = Buffer.alloc(4);
message.writeUInt8(3, 0);
message.writeUInt8(0x2c, 1);
message.writeFloatLE(offset, 2);
```

Wie bei allen Nachrichten sind die ersten beiden Bytes die Grösse der Nachricht und die Message ID. Des Weiteren wird im Payload der Offset gesendet, welchen das Fahrzeug übernehmen soll.

### 5.3.4.6. Turn anfordern

Bei dieser Nachricht wird dem Fahrzeug vermittelt, dass es, anhand der Parameter in der Nachricht, eine gewisse Wendung durchführen soll.
Die Message ID ist hierbei 0x32. Untenstehend noch ein Beispiel, wie eine solche Nachricht aussehen könnte.

```
message = Buffer.alloc(4);
message.writeUInt8(0x03, 0);
message.writeUInt8(0x32, 1); // ANKI_VEHICLE_MSG_C2V_TURN_180
message.writeUInt8(type, 2);
message.writeUInt8(time, 3);
```

Die ersten beiden Bytes sagen aus, wie gross die Nachricht ist und was die Message ID ist. Die beiden Informationen, die im Payload mitgeteilt werden, sind der Typ der Wendung und zu welcher Zeit es durchgeführt werden soll.
Es gibt dabei folgende Arten von Wendungen, welche durch eine Zahl repräsentiert werden:

- 0: Keine Wendung
- 1: Eine Wendung um 90° nach links
- 2: Eine Wendung um 90° nach rechts
- 3: Einen U-Turn um 180°

Es können auch zwei verschiedene Zeitpunkte ausgewählt werden, wann die Wendung vollzogen werden soll:

- 0: Die Wendung soll sofort durchgeführt werden
- 1: Die Wendung wird bei der nächsten Kreuzung durchgeführt

### 5.3.4.7. Batterieladestatus anfordern

Mit dieser Nachricht kann dem Fahrzeug eine Anfrage gesendet werden, um den aktuellen Status der Batterie zu erhalten. Die Message ID für die Anfrage ist 0x1a und für die 0x1b. Untenstehend noch ein Beispiel, wie die Anfrage gebildet wird.

```
message = new Buffer(2);
message.writeUInt8(0x01, 0);
message.writeUInt8(0x1a, 1); // ANKI_VEHICLE_MSG_C2V_BATTERY_LEVEL_REQUEST
```

Die beiden Bytes, die gesendet werden, geben die Grösse der Message und die Message ID an. Es muss hierbei kein weiterer Payload gesendet werden, da der Zweck der Anfrage aus der Message ID herausgelesen werden kann. Falls das Fahrzeug noch erreichbar ist und die Anfrage erhält, sendet es anschliessend eine Antwort mit seinem Batterien Status zurück.

#### 5.3.4.8. Version abfragen

Mit dieser Nachricht kann die aktuelle Version der Firmware eines Fahrzeuges abgefragt werden. Die Message ID der Abfrage ist dabei 0x18 und die der Antwort 0x19. Untenstehend noch ein Beispiel, wie eine solche Nachricht gebildet werden kann.

```
message = new Buffer(2);
message.writeUInt8(0x01, 0);
message.writeUInt8(0x18,1);
```

Die beiden Bytes, die gesendet werden, geben die Grösse der Message und die Message ID an. Es muss hierbei kein weiterer Payload gesendet werden, da der Zweck der Anfrage aus der Message ID herausgelesen werden kann.
Falls das Fahrzeug noch verbunden ist, sendet es eine Antwort, mit der aktuellen Firmware Information, zurück.

### 5.3.5. Antworten von einem Fahrzeug

Im nachfolgenden Abschnitt werden die Antworten, welche von einem Fahrzeug zurückgesendet werden, genauer erläutert. Um den Inhalt der Antworten besser veranschaulichen zu können, wurden die nachfolgenden Definitionen grösstenteils aus der offiziellen Anki-SDK entnommen. Diese Definitionen wurden in der Programmiersprache C und auch NodeJS geschrieben

#### 5.3.5.1. Antwort auf Version Request

Diese Antwort wird vom Fahrzeug zurückgesendet, wenn es eine Versionsanfrage erhalten hat. Diese Nachricht hat dabei die Message ID 0x19.

Untenstehend noch eine Abbildung wie die Struktur der Antwort aufgebaut ist.

```
typedef struct anki_vehicle_msg_battery_level_response {
    uint8_t     size;
    uint8_t     msg_id;
    uint16_t    battery_level;
} ATTRIBUTE_PACKED anki_vehicle_msg_battery_level_response_t;
```

Wie bei den Befehlen, die man an das Fahrzeug sendet, beinhalten die ersten zwei Bytes die Grösse der Nachricht und welche Message ID es hat. Im Payload befindet sich dann die Information, über welche Version das Fahrzeug verfügt.

#### 5.3.5.2. Antwort auf Battery Level Request

Diese Antwort wird zurückgesendet, wenn eine Anfrage zum Status der Batterie im Fahrzeug gesendet wurde. Diese Antwort hat dabei die Message ID 0x1b.

Untenstehend noch eine Abbildung wie die Struktur der Antwort aufgebaut ist.

```
typedef struct anki_vehicle_msg_battery_level_response {
    uint8_t     size;
    uint8_t     msg_id;
    uint16_t    battery_level;
} ATTRIBUTE_PACKED anki_vehicle_msg_battery_level_response_t;
```

Die ersten beiden Bytes beinhalten hierbei die Grösse der Nachricht und die Message ID der Antwort. Im nachfolgenden Payload befindet sich noch die Information, welches Level die Batterie im Fahrzeug hat.

#### 5.3.5.3. Lokalisation Position Update

Diese Antwort wird zurückgesendet, wenn ein Fahrzeug einen Positionierungscode fertiggelesen hat. Diese Antwort hat die Message ID 0x27. In der nachfolgenden Abbildung kann man die Struktur der Antwort sehen.

```
typedef struct anki_vehicle_msg_localization_position_update {
    uint8_t     size;
    uint8_t     msg_id;
    uint8_t     location_id;
    uint8_t     road_piece_id;
    float       offset_from_road_center_mm;
    uint16_t    speed_mm_per_sec;
    uint8_t     parsing_flags;

    /* ACK commands received */
    uint8_t     last_recv_lane_change_cmd_id;
    uint8_t     last_exec_lane_change_cmd_id;
    uint16_t    last_desired_lane_change_speed_mm_per_sec;
    uint16_t    last_desired_speed_mm_per_sec;
} ATTRIBUTE_PACKED anki_vehicle_msg_localization_position_update_t;
```

Hierbei kann man sehen, dass die ersten beiden Bytes die Grösse der Nachricht und die Message ID übermitteln. Der Payload beinhaltet folgende Informationen.

- Location ID: Beinhaltet die ID des Position Codes, dass das Fahrzeug gescannt hat.
- Piece ID: Beinhaltet die ID des Position Codes, dass das Fahrzeug gescannt hat.
- Offset_from_road_center_mm: Beinhaltet die Entfernung, welches das Fahrzeug vom Zentrum des Streckenstückes hat.
- Parsing_flags: Nicht bekannt.

- last_recv_lane_change_cmd_id: Nicht bekannt
- last_exec_lane_change_cmd_id: Nicht bekannt
- last_desired_lane_change_speed_mm_per_sec: Hierbei wird gewünschte Geschwindigkeit angezeigt, mit der das Fahrzeug die Spur wechseln soll.
- last_desired_speed_mm_per_sec: Hierbei wird die letzte, gewünschte Geschwindigkeit angezeigt, mit der sich das Fahrzeug fortbewegen soll.

#### 5.3.5.4. Lokalisation Transition Update

Diese Antwort wird immer dann gesendet, wenn ein Fahrzeug ein Transition Code Stück gescannt hat. Diese Nachricht hat dabei die Message ID 0x29. In der nachfolgenden Abbildung kann man die Struktur der Antwort sehen.

```
typedef struct anki_vehicle_msg_localization_transition_update {
    uint8_t     size;
    uint8_t     msg_id;
    int8_t      road_piece_idx;
    int8_t      road_piece_idx_prev;
    float       offset_from_road_center_mm;

    /* ACK commands received */
    uint8_t     last_recv_lane_change_id;
    uint8_t     last_exec_lane_change_id;
    uint16_t    last_desired_lane_change_speed_mm_per_sec;
    int8_t      ave_follow_line_drift_pixels;
    uint8_t     had_lane_change_activity;

    /* track grade detection */
    uint8_t     uphill_counter;
    uint8_t     downhill_counter;

    /* wheel displacement (cm) since last transition bar */
    uint8_t     left_wheel_dist_cm;
    uint8_t     right_wheel_dist_cm;
} ATTRIBUTE_PACKED anki_vehicle_msg_localization_transition_update_t;
```

Hierbei kann man sehen, dass die ersten beiden Bytes die Grösse der Nachricht und die Message ID übermitteln. Der Payload beinhaltet folgende Informationen.

- road_piece_idx: Nicht bekannt, gibt immer den Wert 0 zurück
- road_piece_idx_prev: Nicht bekannt, gibt immer den Wert 0 zurück
- Offset_from_road_center_mm: Beinhaltet die Entfernung, welches das Fahrzeug vom Zentrum des Streckenstückes hat.
- last_recv_lane_change_cmd_id: Nicht bekannt
- last_exec_lane_change_cmd_id: Nicht bekannt
- last_desired_lane_change_speed_mm_per_sec: Hierbei wird gewünschte Geschwindigkeit angezeigt, mit der das Fahrzeug die Spur wechseln soll.

- ave_follow_line_drift_pixels: Genaue Funktion ist noch nicht bekannt, aber die Annahme ist, dass es anzeigt wie weit das Fahrzeug von der Linie, der es folgt, abdriftet.
- had_lane_change_activity: Dieser Wert meldet, ob das Fahrzeug die Spur jemals gewechselt hat oder nicht.
- uphill_counter: Durch diesen Wert kann errechnet werden, ob es Steigungen auf der abgefahrenen Strecke gibt.
- downhill_counter: Durch diesen Wert kann errechnet werden, ob es Senkungen auf der abgefahrenen Strecke gibt.
- left_wheel_dist_cm: Dieser Wert gibt an, welche Strecke das linke Rad, seitdem letzten Transition Code Stück, zurückgelegt hat.
- right_wheel_dist_cm: Dieser Wert gibt an, welche Strecke das rechte Rad, seitdem letzten Transition Code Stück, zurückgelegt hat.

#### 5.3.5.5. Lokalisation Intersection Update

Diese Antwort wird immer dann ausgesendet, wenn ein Fahrzeug über eine Kreuzungseinfahrt -oder ausfahrt gefahren ist. Die Message ID ist dabei 0x2a. In der nachfolgenden Abbildung kann man die Struktur der Antwort sehen.

```
typedef struct anki_vehicle_msg_localization_intersection_update {
    uint8_t     size;
    uint8_t     msg_id;
    int8_t      road_piece_idx;
    float       offset_from_road_center_mm;

    uint8_t     intersection_code;
    uint8_t     is_exiting;
    uint16_t    mm_since_last_transition_bar;
    uint16_t    mm_since_last_intersection_code;
} ATTRIBUTE_PACKED anki_vehicle_msg_localization_intersection_update_t;
```

Hierbei kann man sehen, dass die ersten beiden Bytes die Grösse der Nachricht und die Message ID übermitteln. Der Payload beinhaltet folgende Informationen.

- road_piece_idx: Nicht bekannt, gibt immer den Wert 0 zurück
- Offset_from_road_center_mm: Beinhaltet die Entfernung, welches das Fahrzeug vom Zentrum des Streckenstückes hat.
- Intersection_code: Durch diesen Wert wird ausgesagt, wo auf der Kreuzung ein Fahrzeug durchgefahren ist.
- Is_existing: Ein Boolean Wert der sagt, ob das Fahrzeug die Kreuzung verlässt oder nicht.
- mm_since_last_transition_bar: Wieviele Millimeter das Fahrzeug seitdem letzten Transition Code Stück zurückgelegt hat.
- mm_since_last_intersection_code: Wieviele Millimeter das Fahrzeug seitdem letzten Kreuzungs Code Stück zurückgelegt hat.

#### 5.3.5.6. Fahrzeug Delocalized Nachricht

Diese Nachricht wird von einem Fahrzeug gesendet, wenn es unplanmässig von der befahrenen Strecke entfernt wurde. Die Message ID ist hierbei 0x2b.

Bei dieser Nachricht wird kein Payload mitgesendet, die einzigen Informationen sind die ersten beiden Bytes, welche die Grösse und die Message ID übermitteln.

#### 5.3.5.7. Car Status Info

Diese Nachricht wird von einem Fahrzeug gesendet, wenn es auf die Ladestation gesetzt oder von dort weggenommen wird. Die Nachricht wird auch versendet, wenn sich das Fahrzeug auf einem Streckenstück fortbewegt oder von dort entfernt wird. Die Message ID ist hierbei 0x3f.

Die Struktur der Nachricht ist wie folgt aufgebaut:

- 1 Byte – Size
- 1 Byte – Message ID
- 1 Byte – IsOnTrack
- 1 Byte – IsCharging

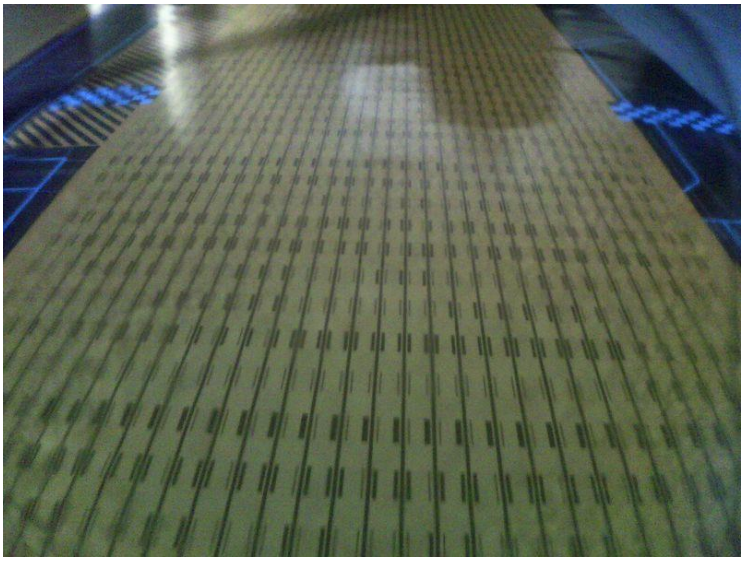
Die ersten beiden Bytes sind hierbei wieder die Grösse der Nachricht und die Message ID. Der Payload besteht bei dieser Nachricht aus den beiden Informationen, ob das Fahrzeug sich auf der Strecke befindet oder sich am Aufladen ist.

### 5.3.6. Streckenstücke

Anki Overdrive verfügt über verschiedene Arten von Streckenstücken, die alle, via Magneten, miteinander verbunden werden können. Die verfügbaren Streckenstücke wären:

- Geraden
- Zielgeraden
- Kurven
- Kreuzungen

Das spezielle bei den Streckenstücken ist, dass sie über einen transparenten Film verfügen, wo man mit dem blossen Auge nichts sehen kann, jedoch infrarote Strahlung durchlässt. Wenn man sich ein Streckenstück nun unter einer Kamera mit einem entsprechendem IR-Filter anschaut, erhält man folgendes Bild.



Wie auf der oberen Abbildung zu sehen ist, gibt es auf jedem Streckenstück mehrere Solcher Linien, mit weiteren Strichen daneben. Über den, im oberen Abschnitt beschriebenen, optischen Sensor, kann ein Anki-Fahrzeug nun diese Linien wahrnehmen und somit seine Position auf der Strecke bestimmen. Dabei repräsentiert jede Linie eine eigene Spur auf der Strecke, von denen es je nach Strecke bis zu 16 gibt.

Dadurch das die Fahrzeuge diese Spuren wahrnehmen und auch folgen können, ermöglicht das Befahren von Kurven und Kreuzungen und auch das parallele Fahren von mehreren Fahrzeugen gleichzeitig. Die Erklärung für die Striche neben den Spuren erfolgt, in einem späteren Abschnitt.

## 5.4. Node JS Anki Controller

Um die Anki Overdrive Fahrzeuge steuern zu können, benötigt es eine Art von Controller, welche über eine BLE Schnittstelle entsprechende Befehle sendet. Im Verlaufe dieses Projektes wurde für diesen Zweck ein entsprechender Controller entwickelt, der in den nachfolgenden Abschnitten genauer erklärt wird

### 5.4.1. MQTT Topic Struktur - Einleitung

Für den Controller wurde eine MQTT Topic Struktur implementiert, wobei über Events und Status Topics Informationen vermittelt werden und über Intents mit dem Controller interagiert werden kann. In den nachfolgenden Abschnitten werden die einzelnen Topics im Detail angeschaut.

## 5.4.2. MQTT Topic Struktur – Event

### 5.4.2.1. Actual Lane Offset

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, die den aktuellen Offset, des Fahrzeugs, zur Mitte mitteilt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/lane/offset/actual", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": offset_pos
}))
```

### 5.4.2.2. Desired Lane Change Speed

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welches die gewünschte Geschwindigkeit, beim Wechseln der Spur, mitteilt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/lane/desired_lane_change_speed", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": pieceLocation
}))
```

### 5.4.2.3. Speed

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welches die aktuelle Geschwindigkeit zurückmeldet. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/speed", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": speed
}))
```

### 5.4.2.4. Last desired Speed

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welches die zuletzt gewünschte Geschwindigkeit zurückmeldet. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/last_desired_speed", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": last_des_speed
}))
```

### 5.4.2.5. Track Piece ID

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche aussagt, dass ein Track Piece ID gescannt wurde. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/track_piece_id", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": pieceId
}))
```

### 5.4.2.6. Track Location ID

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche aussagt, dass eine Track Location ID gescannt wurde. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/track_location_id", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": pieceLocation
}))
```

### 5.4.2.7. Intersection Code

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche aussagt, dass ein Intersection Code gescannt wurde und das Fahrzeug somit sich auf einer Kreuzung befindet. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/intersection_code", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": intersection_code
}))
```

### 5.4.2.8. Average Follow Line Drift Pixels

Dieses Event wird gesendet, wenn eine entsprechende Nachricht vom Fahrzeug gesendet wird, welche diesen Wert beinhaltet. Die genaue Bedeutung dieses Wertes konnte leider noch nicht herausgefunden werden. Man könnte aber vermuten, dass es sich um eine Abweichung in Pixel beim Scannen der Linie handeln könnte. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/ave_follow_line_drift_pixels", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": ave_follow_line_drift_pixels
}))
```

### 5.4.2.9. Lane Change Activity

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche aussagt, ob das Fahrzeug die Spur bereits gewechselt hat oder nicht. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/had_lane_change_activity", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": had_lane_change_activity
}))
```

#### 5.4.2.10. Uphill Counter

Dieses Event wird gesendet, wenn eine entsprechende Nachricht vom Fahrzeug gesendet wird, welche diesen Wert beinhaltet. Die genaue Bedeutung dieses Wertes konnte leider noch nicht herausgefunden werden. Man könnte aber vermuten, dass es sich um eine Art von Zähler handelt, wenn das Fahrzeug nach oben fahrt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/uphill_counter", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": uphill_counter
}))
```

#### 5.4.2.11. Downhill Counter

Dieses Event wird gesendet, wenn eine entsprechende Nachricht vom Fahrzeug gesendet wird, welche diesen Wert beinhaltet. Die genaue Bedeutung dieses Wertes konnte leider noch nicht herausgefunden werden. Man könnte aber vermuten, dass es sich um eine Art von Zähler handelt, wenn das Fahrzeug nach unten fahrt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/downhill_counter", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": downhill_counter
}))
```

#### 5.4.2.12. Left wheel distance in cm

Dieses Event wird gesendet, wenn eine entsprechende Nachricht vom Fahrzeug gesendet wird, welche diesen Wert beinhaltet. Die genaue Bedeutung dieses Wertes konnte leider noch nicht herausgefunden werden. Man könnte aber vermuten, dass es sich um die Distanz handelt, die das linke Rad vom Rand der Strecke hat. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/left_wheel_dist_cm", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": left_wheel_dist_cm
}))
```

#### 5.4.2.13. Right wheel distance in cm

Dieses Event wird gesendet, wenn eine entsprechende Nachricht vom Fahrzeug gesendet wird, welche diesen Wert beinhaltet. Die genaue Bedeutung dieses Wertes konnte leider noch nicht herausgefunden werden. Man könnte aber vermuten, dass es sich um die Distanz handelt, die das rechte Rad vom Rand der Strecke hat. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/right_wheel_dist_cm", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": right_wheel_dist_cm
}))
```

#### 5.4.2.14. mm since last transition bar

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche die Distanz zur letzten Transition Bar, die gescannt wurde, in mm angibt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/mm_since_last_transition_bar", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": mm_since_last_transition_bar
}))
```

#### 5.4.2.15. mm since last intersection code

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche die Distanz zum letzten Intersection Code, der gescannt wurde, in mm angibt. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/mm_since_last_intersection_code", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": mm_since_last_intersection_code
}))
```

#### 5.4.2.16. Delocalized

Dieses Event wird gesendet, wenn eine Nachricht vom Fahrzeug eintrifft, welche aussagt, dass das Fahrzeug sich nicht mehr auf der Strecke befinet. Der Inhalt des Events sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/E/Delocalized", JSON.stringify( value: {
        "timestamp": Date.now()
    }
));
```

#### 5.4.2.17. Car Discovered

Dieses Event wird durch das Scannen nach Fahrzeugen ausgelöst und beinhaltet die Namen aller Fahrzeuge, die gescannt wurden. Der Inhalt des Events sieht wie folgt aus

```
client.publish( topic: "Anki/Host/" + hostID + "/E/CarDiscovered", JSON.stringify( value: {
    "timestamp": Date.now(),
    "Car": device.id
```

### 5.4.3. MQTT Topic Struktur – Intent

Es werden zwischen zwei verschiedenen Arten von Intents unterschiede, dem Connection Intent und den Command Intent. Beide Intents werden in den folgenden Abschnitten im Detail erklärt.

#### 5.4.3.1. Connection

Dieser Intent wird verwendet, um nach Fahrzeugen zu scannen und falls Fahrzeuge gefunden werden, eine Verbindung zu diesen aufzubauen. Der Payload besteht dabei aus einem einzigen Attribut namens «Connecting». Basierend darauf, ob man diesen Wert auf «true» oder «false» setzt, wird eine Verbindung aufgebaut oder alle Verbindungen geschlossen.

Der Inhalt des Intents sieht dabei wie folgt aus.

```
{
  "connecting": true
}
```

#### 5.4.3.2. Command

Mit diesem Intent kann entweder einem Fahrzeug spezifisch oder Global jedem Fahrzeug, ein Befehl gesendet werden. Dieser Intent wird unabhängig von den Befehlen, die übermittelt werden, an einen von zwei Topics gesendet, wobei man nur unterscheidet, ob der Befehl Global gilt oder nicht. Die möglichen Befehle, die in diesem Intent gesendet werden können, sind wie folgt:

- Speed: Hierbei wird die Geschwindigkeit des Fahrzeugs festgelegt, die Beschleunigung muss desweiteren auch noch mitgegeben werden
- Lane: Hierbei kann die Spur festgelegt werden, auf der das Fahrzeug sich fortbewegen soll
- Turn: Hierbei kann man dem Fahrzeug mitteilen, dass es eine bestimmte Wendung durchführen soll. Es müssen dabei die Art und der Zeitpunkt der Wendung mitgeteilt werden.
- Battery: Hierbei kann man anfordern, dass das Fahrzeug seinen aktuellen Batterien Status zurückmeldet
- Version: Hierbei kann man anfordern, dass das Fahrzeug seine aktuelle Firmware Version zurückmeldet
- Ping: Hierbei kann man anfordern, dass das Fahrzeug auf einen Ping antwortet.

### 5.4.4. MQTT Topic Struktur – Status

#### 5.4.4.1. Host Status

Diese Status Nachricht beinhaltet die Information, was der aktuelle Status des Hosts ist. Der Inhalt der Nachricht sieht wie folgt aus.

```
client.publish( topic: 'Anki/Host/' + hostID + '/S/HostStatus', JSON.stringify( value: {
    "value": true
}),  opts: {
    "retain": true,
    "qos": 1
})
```

Die Nachricht ist retained, damit andere Host den Status immer einsehen können, wenn sie dem Broker gerade erst beigetreten sind.
Wenn der Host seine Verbindung verlieren sollte, ohne von sich aus die Verbindung zu schliessen, wird, als letzter Wille, der Wert auf «false» gesetzt.

#### 5.4.4.2. Ping Response

Dieser Status wird veröffentlicht, wenn eine Bestätigung auf einen Ping Request, von einem Fahrzeug, erhalten wurde. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/PingResp", JSON.stringify( value: {
        "timestamp": Date.now
    }
), {
    "retain": true,
    "qos": 1
});
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.3. Version

Dieser Status wird veröffentlicht, wenn eine Bestätigung auf einen Versions Request, von einem Fahrzeug, erhalten wurde. Der Inhalt der Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/Version", JSON.stringify( value: {
        "timestamp": Date.now,
        "value": version
    }
), {
    "retain": true,
    "qos": 1
});
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.4. Battery Level

Dieser Status wird veröffentlicht, wenn eine Bestätigung auf einen Battery Level Request, von einem Fahrzeug, erhalten wurde. Der Inhalt der Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/BatteryLevel", JSON.stringify( value: {
        "timestamp": Date.now,
        "value": level
    }
), {
    "retain": true,
    "qos": 1
});
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.5. Desired Speed

Dieser Status wird veröffentlicht, wenn ein Intent an den Controller gesendet wurde, dass die Geschwindigkeit des Fahrzeugs verändert werden soll. Dabei wird diese Information im Status als gewünschte Geschwindigkeit angegeben. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + target + "/S/Speed/Desired", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": speed
}), {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.6. Desired Lane

Dieser Status wird veröffentlicht, wenn ein Intent an den Controller gesendet wurde, dass die Spur des Fahrzeugs verändert werden soll. Dabei wird diese Information im Status als der gewünschte Offset angegeben. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + target + "/S/Lane/Desired", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": offset
}), {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.7. Actual Lane

Dieser Status wird veröffentlicht, wenn eine Nachricht vom Fahrzeug erhalten wird, mit der Information, was die aktuelle Spur ist, auf der sich das Fahrzeug befindet. Dabei wird diese Information im Status als der aktuelle Offset angegeben. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/Lane/Actual", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": offset_pos
}), {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.8. Actual Speed

Dieser Status wird veröffentlicht, wenn eine Nachricht vom Fahrzeug erhalten wird, mit der Information, was die aktuelle Geschwindigkeit ist. Dabei wird diese Information im Status als die aktuelle Geschwindigkeit angegeben. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/Speed/Actual", JSON.stringify( value: {
    "timestamp": Date.now(),
    "value": speed
}), {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.9. CarStatus

Dieser Status wird veröffentlicht, wenn ein Fahrzeug seinen aktuellen Zustand in Form einer Nachricht zurücksendet. Diese Nachricht beinhaltet folgende Informationen:

- isCharging
- onTrack

Diese Informationen werden im Status entsprechend veröffentlich, zusätzlich mit der weiteren Information, dass das Fahrzeug online ist. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish("Anki/Car/" + vehicle.id + "/S/CarStatus", JSON.stringify( value: {
    "timestamp": Date.now(),
    "online": true,
    "charging": isCharging,
    "onTrack": isOnTrack
}), {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.10. Information

Dieser Status wird veröffentlicht, sobald ein Fahrzeug fertig gescannt wurde. Dabei werden in der Status Nachricht folgende Informationen vermittelt:

- Geräteadresse
- Identifier
- Model
- Model ID
- Product ID

Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish( topic: "Anki/Car/" + device.id + "/S/Information", JSON.stringify( value: {
    "address": device.address,
    "identifier": manufacturerData.readUInt32LE( offset: 0),
    "model": getModel(manufacturerData.readUInt8( offset: 4)),
    "modelId": manufacturerData.readUInt8( offset: 4),
    "productId": manufacturerData.readUInt16LE( offset: 6)
}), opts: {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.11. Cars

Bei diesem Status wird eine Liste mit allen verbundenen Fahrzeugen veröffentlicht. Die Status Message sieht wie folgt aus.

```
client.publish( topic: "Anki/Host/" + hostID + "/S/Cars", JSON.stringify(payload), opts: {
    "retain": true,
    "qos": 1
});
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

#### 5.4.4.12. Discovery Time

Dieser Status wird veröffentlicht, wenn ein Fahrzeug beim Scannen gefunden wurde. Die Information, die im Status veröffentlicht wird, ist der Zeitpunkt der Entdeckung. Der Inhalt der Status Nachricht sieht wie folgt aus.

```
client.publish( topic: "Anki/Car/" + device.id + "/S/DiscoveryTime", JSON.stringify( value: {
    "timestamp": Date.now(),
}), opts: {
    "retain": true,
    "qos": 1
})
```

Da es sich um eine Status Nachricht handelt, ist sie retained. Dadurch kann jeder Client, der sich gerade erst auf den Broker verbindet, diesen Status erhalten.

## 5.5. Erweitern von Anki Overdrive durch eigene Streckenstücke

Die Firma Anki, welche die Anki Overdrive Produkte entwickelte und verkaufte musste Mitte 2019 ihre Türen schliessen, da sie Konkurs gingen. Das bedeutete, dass die Produktion und der Verkauf der Anki Overdrive Produktlinie per sofort stoppte. Das war zwar nicht das Ende für Anki Overdrive, da die Lizenzrechte von der Firma Digital Dream Labs aufgekauft wurden. Nach dem Aufkaufen der Patente für Anki Overdrive, sagte Digital Dream Labs, dass sie die Entwicklung und die Produktion wieder aufnehmen würden, jedoch kann man bis heute nur Restbestände kaufen.

Das bedeutet, dass man nur umständlich an neue Streckenteile gelangt und dabei auch eine rechte Summe Geld dafür hinlegen muss. Da würde es sich lohnen die Frage zu beantworten, ob man nicht seine eigenen Streckenteile herstellen und sich somit seine eigenen Bahnen zusammenstellen kann.

### 5.5.1. Aufbau Allgemein

In diesem Abschnitt geht es darum, wie die Logik der Streckenstücke im Allgemeinen funktioniert. Wie man im oberen Abschnitt gesehen hat, besteht jedes Streckenstück, unterhalb der Folie, aus Linien, denen die Fahrzeuge folgen können. Diese Linien verfügen des Weiteren über dünnere und dickere Striche links und rechts der Linien. Diese Striche links oder rechts der Spur übermitteln den Fahrzeugen unterschiedliche Informationen. Bis jetzt wurden diese 3 unterschiedliche Arten von Strichen gefunden, wie man in der Abbildung unterhalb sehen kann.


1. Diese Art von Strich vermittelt einem Fahrzeug, dass entweder ein neues Streckenstück oder ein neues Teilstück auf der gleichen Strecke anfängt. Somit können die Fahrzeuge die unterschiedlichen Abschnitte auf einem Streckenstück voneinander unterscheiden.
2. Diese Art von Strich übermittelt einem Fahrzeug den binären Wert «1»
3. Diese Art von Strich übermittelt einem Fahrzeug den binären Wert «0»

Bei den Kurven sehen diese Striche leicht anders aus, da sie gebogen sind, aber das Prinzip dahinter bleibt das gleiche.

Grundsätzlich besteht ein Streckenabschnitt aus mehreren Teilbereichen, die je durch die Striche vom Typ 1 unterteilt werden. Diese Teilbereiche haben dabei je 7 dieser Striche vom Typ 2 und 3, die parallel zur Spur angelegt sind. Ein Beispiel dafür finden sie im Bild unten.


Bei der oberen Abbildung kann man sehen, wie ein Teilstück einer geraden aussehen kann. An den beiden Enden befindet sich dabei jeweils ein Strich des Typen 1, der den Anfang oder das Ende eines Teilstückes markiert. Dazwischen befinden sich 7 verschieden dieser Striche des Typen 2 und 3, jeweils Links und Rechts der Spur, welcher das Fahrzeug folgt. Soweit man weiss, benötigen die Fahrzeuge immer 7 dieser Striche, damit sie alle Informationen korrekt ablesen können.

Jetzt bleibt noch die Frage offen, was für Informationen diese Striche übermitteln. Das lässt sich wie folgt beantworten, da diese Striche des Typen 2 und 3 die binäre Werte 1 und 0 repräsentieren, kann man diese 7 Striche als 7 Bit anschauen. Das bedeutet, wenn ein Fahrzeug diese Striche scannt, hat es am Ende der Teilstrecke 7 Bit pro Seite. Basierend von welcher Seite das Fahrzeug auf die Teilstrecke, werden die 7 Bit inverse gescannt, damit man von jeder Seite her die gleichen Identifier hat. Diese Identifier haben pro Seite folgende Bedeutung.

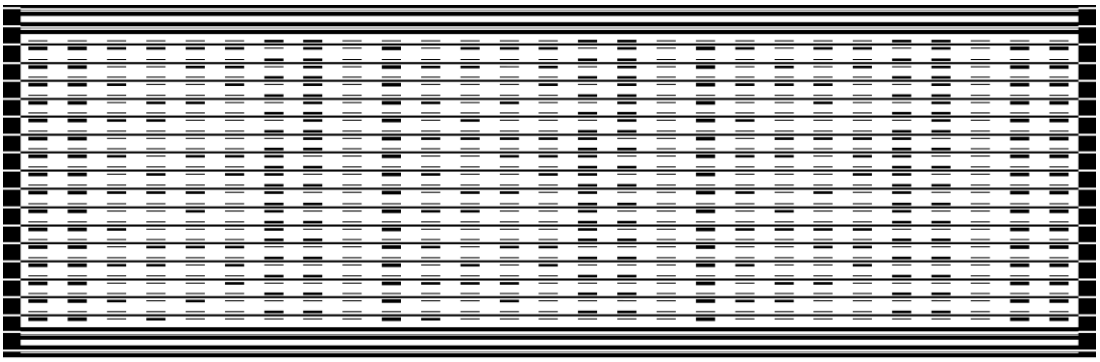
- Links: Piece ID
- Rechts: Location ID

Die 7 Bit auf der linken Seite identifizieren ein gesamtes Streckenstück, das heisst das überall auf dem Streckenstück die gleiche ID auf der linken Seite abgelesen wird. Diese Piece IDs können auch verwendet werden, um sagen zu können, um welche Art von Streckenstück es sich handelt, ob es eine Gerade ist oder eine Kurve.

Die 7 Bit auf der rechten Seite identifizieren, wo man sich auf dem Streckenstück befindet. Dabei fangen die IDs bei 0 an und inkrementieren für jede Spur um eine Zahl nach oben.

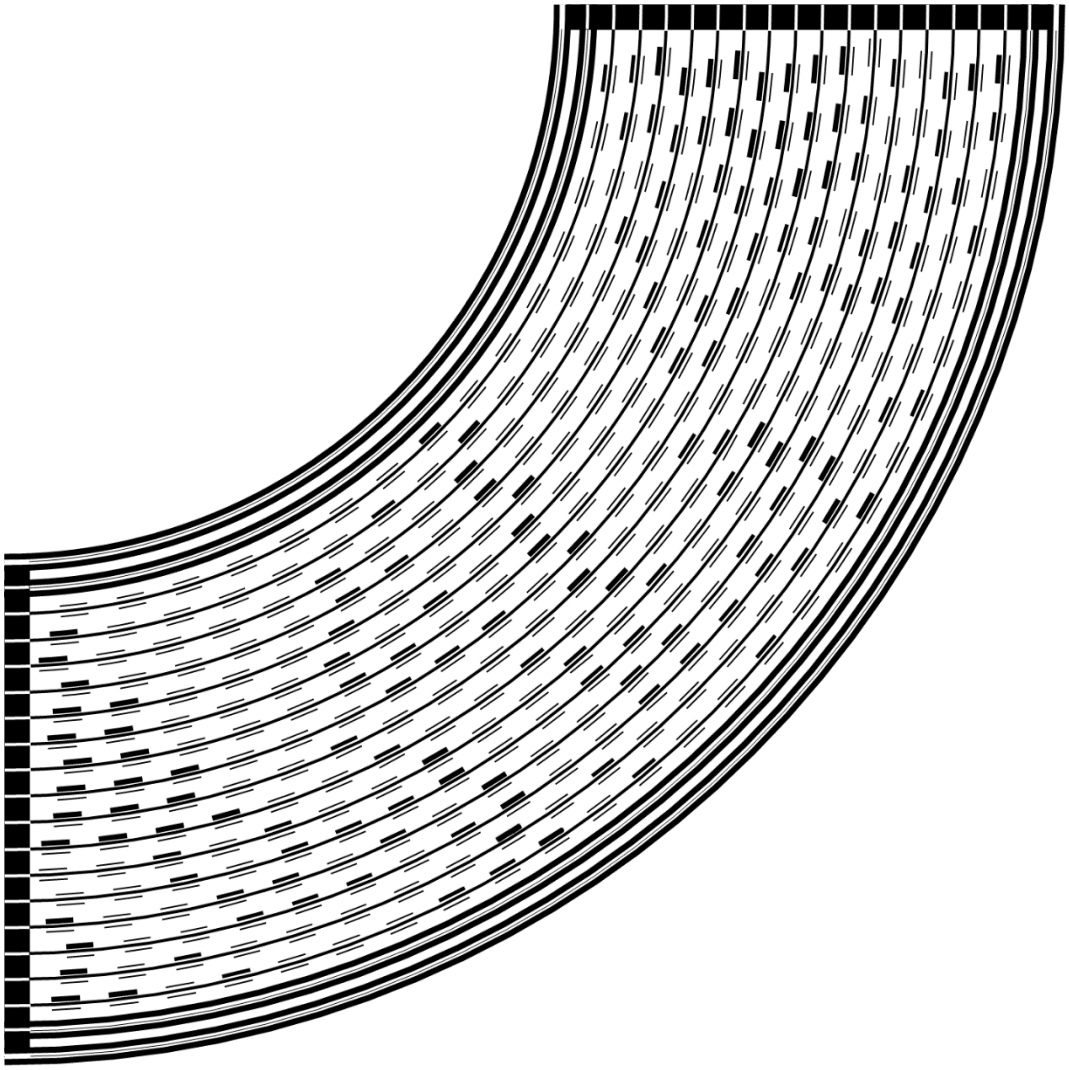
### 5.5.2. Streckenstücke selbst ausdrucken

Durch das Anwenden der oben erwähnten Logik kann man nun eigentliche eigene Streckenstücke herstellen und ausdrucken. In der unteren Abbildung kann man ein Beispiel für eine Gerade in Anki Overdrive sehen, die ausgedruckt wurde und erfolgreich in ein bestehendes System eingebaut wurde.



In der oberen Abbildung kann man sehen, wie eine Gerade von Anki Overdrive unter der Folie aussehen würde. Dieses Beispiel hier wurde jedoch anhand der oben genannten Logik erstellt und kann auch ausgedruckt werden.

Ein weiteres Beilspiel kann man in der zweiten Abbildung unterhalb sehen.



Die Abbildung oben zeigt ein Kurvenstück von Anki Overdrive, wie es unterhalb der Folie aussehen würde. Dieses Kurvenstück oberhalb wurde anhand der gleichen Logik, wie bei der Geraden oben, erstellt. Zwar sind hierbei die Striche gebogen und besteht die Spur ganz links nur aus 2 Teilstücken, jedoch besteht die Logik hier trotzdem. Diese Streckenstück wurde auch testweise ausgedruckt und erfolgreich in ein Anki Overdrive System implementiert.

## 5.6. Quelle

https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

https://en.wikipedia.org/wiki/Node.js

https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

https://github.com/abandonware/noble

https://libraries.io/npm/@abandonware%2Fnoble

https://libraries.io/npm/noble

https://www.microcontrollertips.com/teardown-inside-anki-overdrive-racecar-set/

http://anki.github.io/drive-sdk/docs/programming-guide

## 5.7. Abbildungsverzeichnis

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

https://www.microcontrollertips.com/teardown-inside-anki-overdrive-racecar-set/

http://anki.github.io/drive-sdk/docs/programming-guide