SS 2015

Autonomous RFID Car

Physical Computing

NAVIGATION IM RAUM MITTELS RFID-TAGS

CHRISTIAN BINKOWSKI, OLIVER HOFFMANN, KATRIN LUTTER

Inhalt

1. Einführung

In der Hochschule wurde im letzten Jahr ein kompletter Raum mit RFID-Tags im Boden ausgestattet. Diese Tags können mit einem RFID-Reader ausgelesen werden. Der Boden bietet eine gute Grundlage um eine Positionsbestimmung innerhalb des Raumes zu verwirklichen. Aus diesem Grund kam unsere Gruppe im Rahmen der Vorlesung „Physical Computing“ im Sommersemester 2015 zu der Idee, einen Roboter innerhalb des Raumes an bestimmte Positionen navigieren zu lassen. Nach kurzer Zeit wurde festgestellt, dass bereits ein Projekt, das sich mit der Orientierung im Raum mittels RFID-Tags beschäftigt hat, realisiert wurde

Das Vorgängerprojekt erstellte eine Karte des Raumes mittels der RFID-Tags. Sie versuchten sich dann anhand eines Vektors im Raum zu orientieren. Man gibt bei diesem System den Startpunkt und den Endpunkt an. Der Computer berechnet dann aus diesen zwei Punkten einen Vektor und den dazugehörigen Winkel.

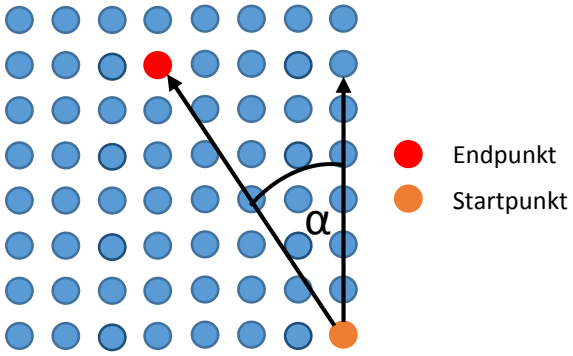

Abbildung 1: Beispiel altes Konzept

Der berechnete Winkel α gibt den Drehwinkel für den Roboter vor. Aufgrund von Winkelabweichungen bei der Drehung kann es passieren, dass der Roboter durch die Abweichung nicht ans Ziel gelangt. Dies tritt schon bei ein paar Grad Abweichung auf. Zudem wird, wenn ein RFID-Tag detektiert wird, ständig ein neuer Weg berechnet bis der Roboter nach mehreren Versuchen an der Zielposition ankommt. Dies führt zu einem nicht deterministischen Verhalten des Roboters. Es ist also nie vorhersagbar, wie und wann der Roboter welchen Weg befährt.

Um dies vorzubeugen wurde ein neues System entwickelt. Hierbei sollen nicht ein Vektor und ein Winkel bestimmt werden, dem der Roboter folgen soll. Stattdessen soll der Roboter fest definierten Wegpunkten folgen und somit von RFID zu RFID tag fahren.

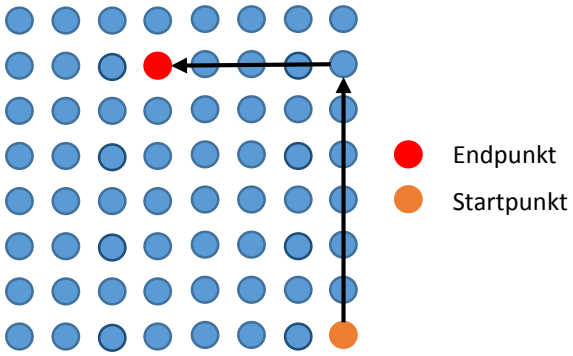

Abbildung 2: Beispiel neues Konzept

Wie auf dem obigen Bild zu sehen ist, wird nicht ein spezieller Winkel angesteuert, sondern von Punkt zu Punkt gefahren, bis der Roboter sein Ziel erreicht hat. Mit dem neuen Konzept sollen auch nur 90°-Drehungen nach links und nach rechts möglich sein, um eventuelle Abweichungen so klein wie möglich zu halten. Dies spielt vor allem bei der Hindernisumfahrung eine bedeutende Rolle, da auf diese weiße Hindernisse gezielt Umfahren werden können und ein neuer Weg leicht zu berechnen ist.

Der Roboter soll in diesem Projekt mit zusätzlichen Sensoren ausgestattet werden. Mit einem Ultraschallsensor soll es möglich sein Hindernisse zu erkennen und diese dann zu umfahren. Des Weiteren soll ein Kompass helfen die Drehungen zu präzisieren. Die RFID-Tags im Boden werden weiterhin mit dem RFID-Sensor ausgelesen. Durch die neu angebrachten Sensoren soll das ganze System intelligenter, autonomer und zuverlässiger werden.

Das Hardware-Grundgerüst, welches aus dem Arduino Board und dem Lego Mindstorm Roboter besteht, wurde von der Vorgruppe übernommen. Es wurde hierbei die Verdrahtung neu gestaltet um mehrere Sensoren am gleichen Bussystem anschließen zu können. Die Software für das Arduino Board sowie das Lego Mindstorm Programm wurde nach eigenen Anforderungen komplett neu gestaltet. Des Weiteren wurde auch bei der Datenverarbeitung am PC keinerlei Code der Vorgruppe berücksichtigt.

2. Funktionalitäten

2.1. Modellierung einer Karte und Wegberechnung

Als erstes soll der Raum in einheitlich große Bereiche eingeteilt werden. Damit soll ein Raummodell in Form einer Karte gebildet werden.

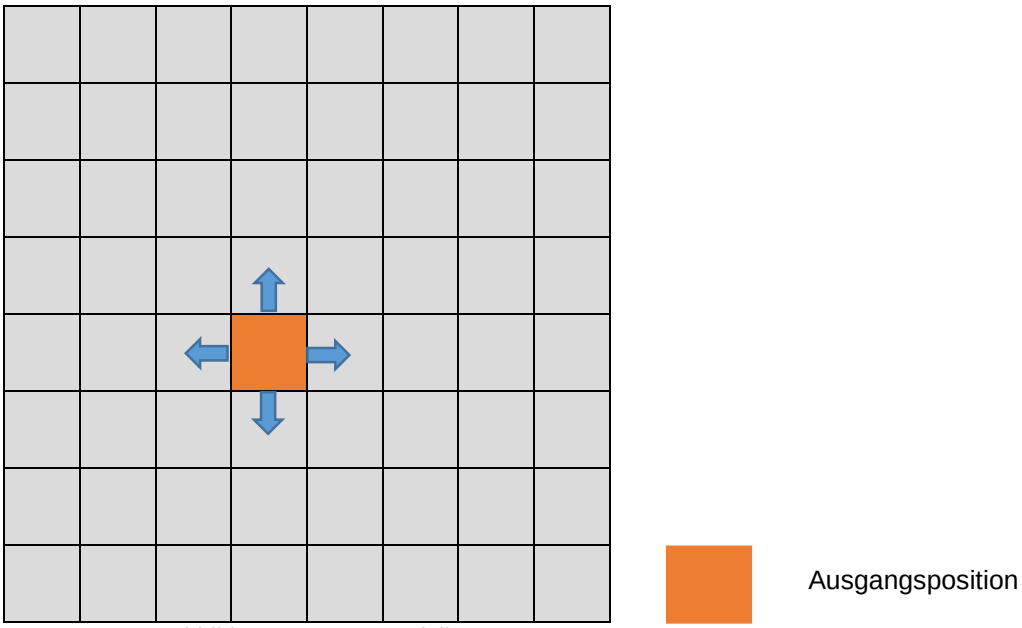

Abbildung 3: Kartenmodell

Jedes Element in diesem Modell entspricht einer Fläche von 25cm x 25cm im Raum, welche von einem RFID-Tag, der im Boden verlegt wurde, repräsentiert wird. Eine dieser Elemente entspricht somit einer fest definierten Fläche des Raumes, in der sich der Roboter befinden kann. Die Navigation erfolgt Schrittweiße von Element zu Element. Es ist dem Roboter nur möglich in eine der vier Himmelsrichtungen zu fahren. Eine diagonale Bewegung wird aufgrund der Ungenauigkeit und der erhöhten Komplexität der Wegberechnung und Hindernisumfahrung ausgeschlossen. Durch einen Algorithmus wird stets dafür gesorgt, dass der Roboter über die kürzeste Route zur Zieladresse fährt. Dabei spielt es keine Rolle, ob der RFID-Tag eines Elementes auf diesem Weg erkannt wird oder nicht. Mit Hilfe der Karte soll es dem Benutzer möglich sein, eine Zielposition festzulegen. Das Programm berechnet anschließend anhand der Ausgangsposition und der Zielposition die kürzeste Route und steuert den Roboter. Wie dies funktioniert, wird im Kapitel 5 genauer beschrieben.

2.2. Hinderniserkennung

Eine weitere wichtige Funktionalität ist die eingebaute Hinderniserkennung. Diese soll Kollisionen mit Hindernissen vermeiden. Wird ein Hindernis auf der Zielroute des Roboters erkannt wird das Element, das der Roboter als nächstes anfahren würde, als Hindernis markiert. Anschließend kommt es zu einer Neuberechnung des Weges unter Berücksichtigung des Hindernisses. Alle Hindernisinformationen werden zentral verwaltet und gespeichert. Dabei erkennt der Roboter ebenfalls, wenn ein gespeichertes Hindernis nicht mehr präsent ist und aktualisiert die virtuelle Karte. Die Hinderniserkennung kommt sowohl bei der Navigation mittels berechneten Weg, als auch bei der in Kapitel 2.3 beschriebenen Suchfunktion zum Einsatz.

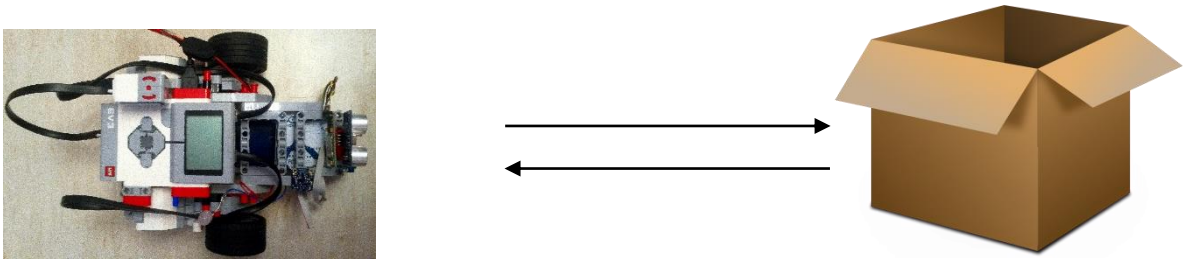

Abbildung 4: Hinderniserkennung

Aus Sicherheitsgründen wurde Folgendes festgelegt:

Es wird stets eine Reserve von 5cm bei der Abstandsüberwachung einkalkuliert. Diese dient dazu, dass der Roboter sich noch auf der Stelle drehen kann, ohne mit dem Hindernis in Berührung zu kommen.

2.3. Suchfunktion zur Positionsbestimmung

Es soll dem Benutzer möglich sein beim Start den Roboter beliebig im Raum zu platzieren, ohne dass sich der Roboter auf einem RFID-Tag befindet. Damit der Roboter möglichst schnell und genau einen RFID-Tag in der Nähe findet, soll er eine gewisse Bewegung durchführen und nach dem Fund eines Tags auf diesem stehen bleiben.

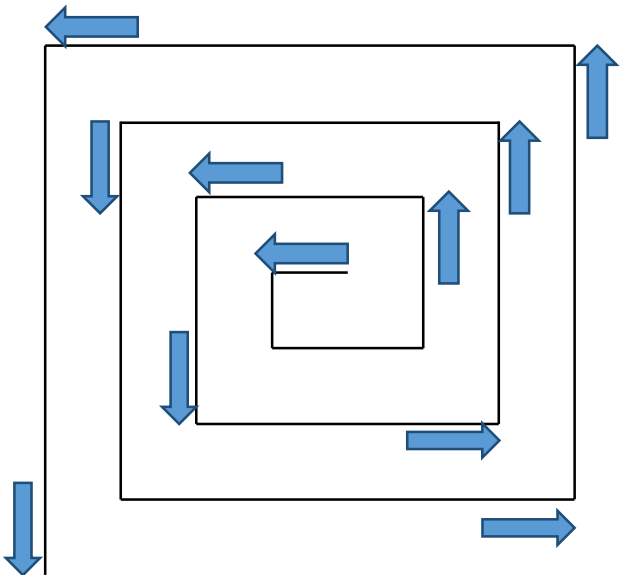

Abbildung 5: Such-Schema

Der Roboter bewegt sich hierbei auf einem „spiralförmigen“ Weg von der aktuellen Position weg. Dabei soll eine bestimmte Fläche komplett von dem RFID-Reader abgedeckt werden. Somit wird sichergestellt, dass der nächstliegende RFID-Tag gefunden wird.

Dieses Muster soll auch am Ende einer Fahrt abgefahren werden, falls der gesuchte Tag nicht gefunden wurde. Dies kann bei einer längeren Fahrt durchaus der Fall sein, da es durch viele Drehungen sowie „Spiel“ an den Rädern, keine exakt gerade Bewegung möglich ist.

Des Weiteren wird beim Suchvorgang auch die Hinderniserkennung eingesetzt. Der Roboter, soll in diesem Fall so nah wie möglich dann an das Hindernis heranfahren, dann eine Drehung vollführen und anschließend mit dem normalen Suchablauf weitermachen.

3. Komponenten und Kommunikation

3.1. Technologieschaubild

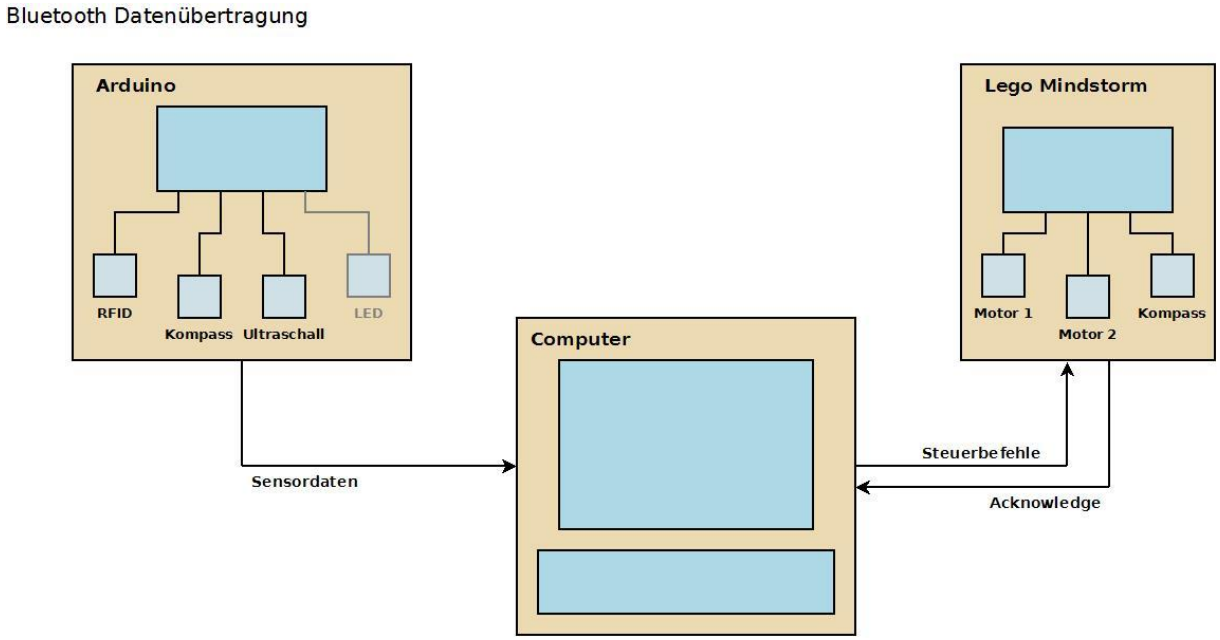

Abbildung 6: Aufbau und Kommunikation

Das Projekt besteht aus 3 Hauptbestandteilen:

- dem Arduino Board mit einem Bluetooth-Modul, RFID-Reader, Kompass-Modul (Gyroskop), Ultraschallsensor und einer LED
- dem Lego Mindstorm EV3-Baustein mit 2 DC-Motoren, einem Gyroskop und einem Bluetooth-Modul
- der PC-Komponente mit Bildschirm, Bluetooth-Transceiver, Tastatur und Maus.

Die Kommunikation aller Komponenten erfolgt über Bluetooth.

Zu Beginn baut die PC-Komponente die Verbindung zu den beiden anderen Teilnehmern auf. Anschließend sendet das Arduino Board im Abstand von ca. 250ms Telegramme mit den Sensordaten an die PC-Komponente. Das Arduino Board sendet zusätzlich immer dann ein Telegramm mit allen Daten, wenn ein RFID-Tag erkannt wird. Dies geschieht um unmittelbar auf das Erkennen eines RFID-Tags reagieren zu können.

Die PC-Komponente kommuniziert mit dem Benutzer über eine grafische Benutzeroberfläche, empfängt die Sensordaten und wertet diese aus. Aufgrund der Ergebnisse wird daraufhin der weitere Weg berechnet.

Außerdem gibt die PC-Komponente Bewegungsabläufe an den Lego Mindstorm EV3-Baustein weiter. Hierzu wurden verschiedene Befehle definiert, die angeben in welche Richtung sich der Roboter bewegen soll. Der Lego Mindstorm EV3-Baustein empfängt die Nachricht und steuert die Motoren an. Nachdem ein Bewegungsabschnitt ausgeführt wurde, sendet der Lego Mindstorm EV3-Baustein eine Quittierung zurück und teilt der PC-Komponente so mit, dass der zuletzt gesendete Befehl ausgeführt wurde.

3.2. Aufbau des Roboters

Der Roboter besteht aus Bauteilen der Lego Mindstorm-Reihe, einem Arduino Board und damit verbundenen Sensoren. Der grundlegende Aufbau wurde von dem bereits existierenden Projekt übernommen.

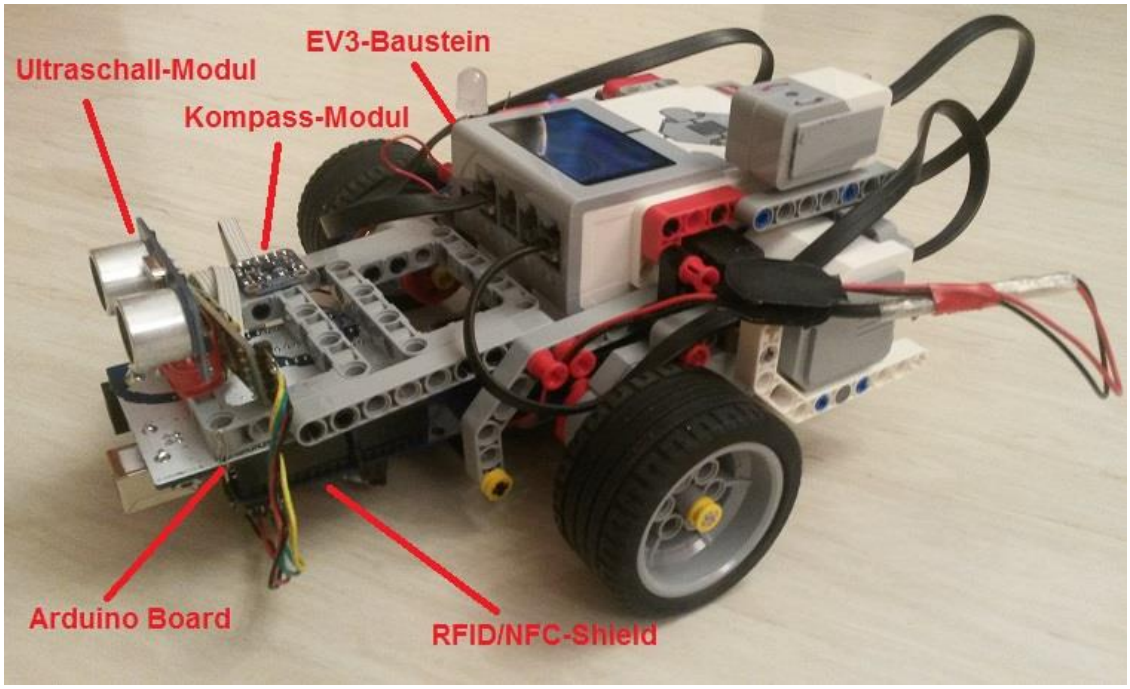

Abbildung 7: Der Roboter

Um die weiteren Sensoren wie das Kompass-Modul und das Ultraschall-Modul anbringen zu können, wurden einige Änderungen im vorderen Bereich des Lego Mindstorm Aufbaus vorgenommen.

Zudem wurde der vordere Teil der Konstruktion etwas verlängert, um das NFC-Shield möglichst mittig unter den Roboter-Aufbau anbringen zu können und so die Genauigkeit beim Anfahren der RFID-Tags noch weiter zu verbessern. Ist der RFID-Leser direkt zwischen den Rändern positioniert, werden die Drehungen, die auf der Stelle vollzogen werden, auch direkt über dem erkannten RFID-Tag ausgeführt und es kommt zu geringeren Abweichungen.

Die Leitungen am Arduino Board wurden ebenfalls neu angebracht. Die Verbindungen der Sensoren mit dem Arduino Board wurden von einer festen Verbindung abgeändert auf eine steckbare Verbindung und alle zusätzlichen Sensoren wurden auf gleiche Art und Weise angebracht. Jeder Sensor kann nun ganz einfach auch separat abgesteckt werden ohne, dass dies den restlichen Aufbau beeinflusst oder sonstige Änderungen an der Hardware vorgenommen werden müssen um ein funktionsfähiges System zu erhalten.

4. Arduino

Das Board Arduino Uno und verschiedenen Sensoren sind die Hauptbestandteile zur Orientierung im Raum. Die Sensordaten werden auf unterschiedliche Weise an das Arduino Board weitergegeben, anschließend verarbeitet und über Bluetooth an die PC-Komponente übermittelt.

4.1. Der Schaltplan

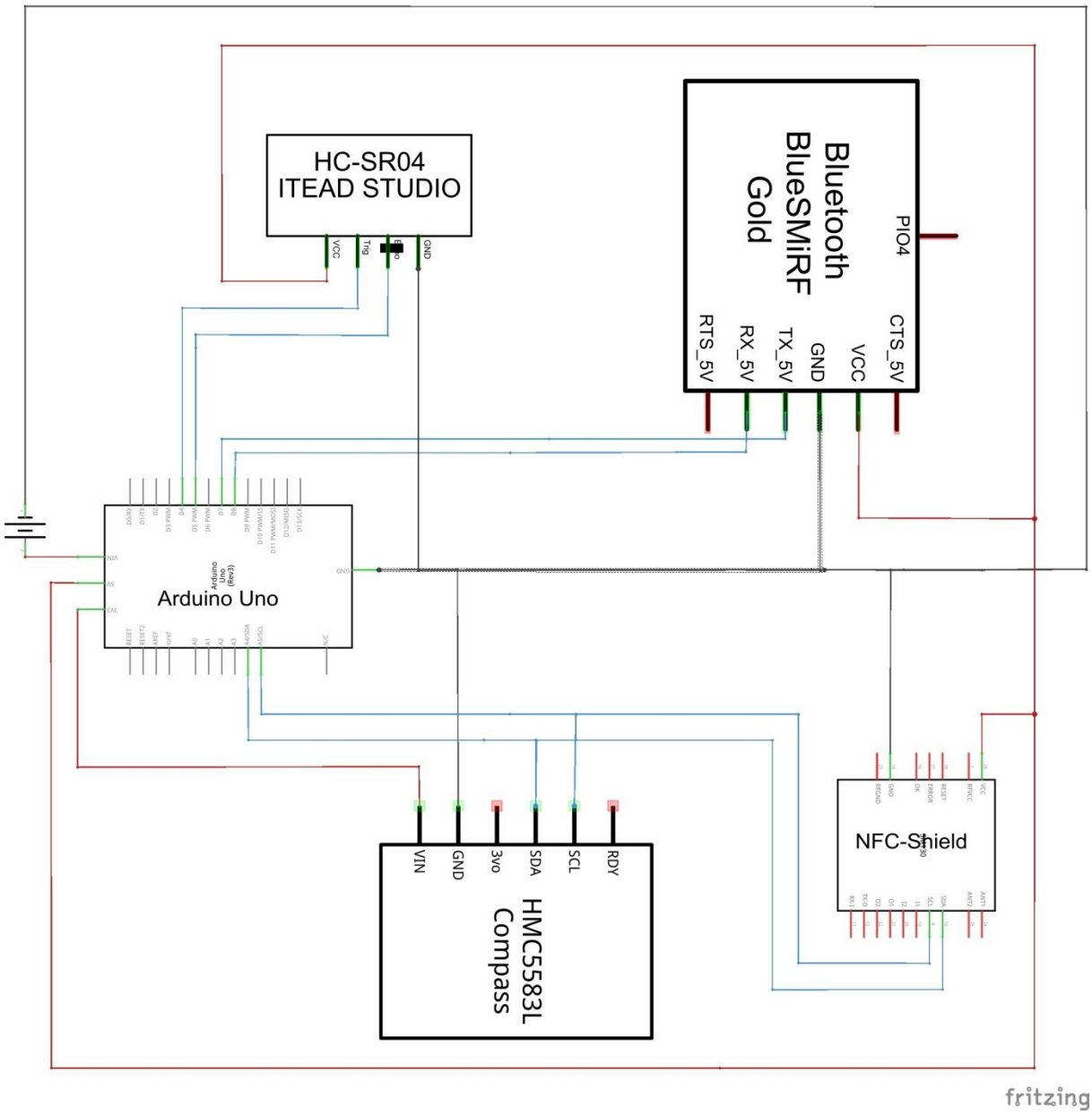

Abbildung 8: Schaltplan Arduino und Sensoren

Das Arduino Board wird über ein 9V-Blockbatterie Batterieanschlusskabel, welches mit dem USB-Anschluss des EV3-Bausteins verbunden ist, mit Spannung versorgt. Um die korrekte Funktionsweise aller Sensoren zu gewährleisten, werden diese, je nach Bedarf mit 5V oder 3V-Ausgangsspannung betrieben.

Die Pin-Belegung des Arduino Boards ist in Folgender Tabelle beschrieben:

Digitale Pin 2	NFC – Interrupt
Digitale Pin 3	NFC – Reset
Digitale Pin 4	Ultraschall Sensor Trigger
Digitale Pin 5	Ultraschall Sensor Echo
Digitale Pin 6	Bluetooth RX
Digitale Pin 7	Bluetooth TX
Digitale Pin 8	RGB – LED grün (Aktor)
Analog Input 4	I2C SDA
Analog Input 5	I2C SCL

Abbildung 9: Pinbelegung

Der Ultraschall-Sensor nutzt den Digitalen Pin 4 als Digitalen Ausgang und den Digitalen Pin 5 als Digitalen Eingang.

Für den Bluetooth-Baustein wird der Digitale Pin 6 zum Empfangen von Bluetooth-Signalen und der Digitale Pin 7 zum Versenden genutzt.

Die LED wird über den Digitalen Pin 8 angesteuert.

Der Kompass und das RFID/NFC-Shield nutzen den I2C-Bus zur Datenübertragung. Beim Arduino Uno werden die Analogen Pins 4 und 5 für den I2C als SDA und SCL-Leitung verwendet.

4.2. Die Sensoren

Die Programmteile der Sensoren wurden größtenteils aus den Beispielprojekten der Bibliotheken entnommen.

4.2.1. Ultraschall-Modul: HC SR04

Mit dem Ultraschall-Modul HC-SR04 können Entfernungen von 2 cm bis hin zu 3 Metern gemessen werden. Die Entfernung wird mit einer Genauigkeit von 3 mm bestimmt.

Das Ultraschall-Modul wird mit einer Versorgungsspannung von 5V betrieben.

Um die Entfernung zu Messen wird am Trigger-Eingang nach 10µs eine fallende Flanke benötigt. Daraufhin sendet das Modul ein 40kHz Signal, welches anschließend vom Echo-Pin erwartet wird. Wird ein Echo-Signal empfangen geht der Echo-Pin von einem High-Pegel auf einen Low-Pegel über. Sollte kein Echo-Signal empfangen werden, bleibt der Echo-Pin für 200ms auf High und zeigt so an, dass keine Messung erfolgen konnte. Erst 20ms nach der letzten Messung kann eine neue Messung gestartet werden.[1]

Der Programmcode:

```
digitalWrite(ULTRASONIC_TRIGGER, LOW);
delayMicroseconds(2);
digitalWrite(ULTRASONIC_TRIGGER, HIGH);
delayMicroseconds(10);
```

[1] KT-Elektronik, Ultraschall Messmodul HC-SR04, http://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf, 10.07.2015

```
  digitalWrite(ULTRASONIC_TRIGGER, LOW);

  duration = pulseIn(ULTRASONIC_ECHO, HIGH);
  distance_in_cm = duration * 343 / 20000;
  send_msg.us_distance = distance_in_cm;
```

Zu Beginn des Code-Segments wird der Trigger-Pin auf Low gesetzt um anschließend die exakten 10µs High-Pegel zu gewährleisten, auf welche die fallende Flanke folgen muss um den Messvorgang einzuleiten.

Über die pulseIn-Funktion wird gewartet bis am Digitalen PWM-Pin 5 ein High-Pegel angelegt wird. Ein Timer wird gestartet und stoppt die Zeit bis das Echo-Signal vom Modul empfangen wird und dadurch der Pin wieder einen Low-Pegel aufweist. Daraus ergibt sich die Zeit, die benötigt wurde vom Senden des Signals bis zum Empfangen (duration).

Anschließend erfolgt die Berechnung der Entfernung. Hierzu wird die benötigte Zeit in µs mit der Schallgeschwindigkeit multipliziert. Die Schallgeschwindigkeit bei 20°C beträgt in der Luft 343 m/s. Um die Einheiten auszugleichen und am Ende auf die Einheit cm zu kommen muss die Gleichung durch 10 000 geteilt werden. Da die Zeit aber für die Strecke zum Hindernis und zurück gemessen wurde muss das Ergebnis noch durch 2 geteilt werden um den Abstand zum Hindernis zu erhalten. Das bringt uns zu der im Code-Segment verwendeten Formel.

4.2.2. Kompass-Modul: HMC 5883L Magnetometer

Mit dem HMC 5883L kann die magnetische Flussdichte von Magnetfeldern in 3 Achsen gemessen werden, wodurch man ihn auch als Kompass verwenden kann. Das Kompass-Modul misst die Richtung mit einer Genauigkeit von ca. 2 Grad. Es wird mit einer Spannung von 3V versorgt und kommuniziert mit dem Arduino Board über den I2C-Bus.

Für die Nutzung des Kompass-Moduls wird von Adafruit eine Arduino-Bibliothek bereitgestellt. Um den Sensor auslesen zu können, werden 2 der Bibliotheken benötigt. Zum einen die „Adafruit_Sensor.h“ und zum anderen die „Adafruit_HMC5883_U.h“.

Um das Kompass-Modul auslesen zu können wird zuerst eine Instanz der Kompass-Klasse erstellt.

```
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
```

Anschließend müssen die Messungen über die begin-Methode gestartet werden. Jetzt kann das Kompass-Modul die Werte der gemessenen magnetischen Flussdichte zurückgeben.

Der Programmcode:

```
sensors_event_t event;
mag.getEvent(&event);
heading = atan2(event.magnetic.y, event.magnetic.x);
float declinationAngle = 0.0457;
heading += declinationAngle;

if(heading < 0)
  heading += 2*PI;

if(heading > 2*PI)
  heading -= 2*PI;

headingDegrees = heading * 180/M_PI;
```

```
send_msg.com_degrees = headingDegrees;
```

Um die Sensordaten verwenden zu können wird über die Event-Klasse eine Event-Instanz angelegt. In dieser werden nach dem Holen mit getEvent, die Sensordaten abgespeichert.

Zum Bestimmen der Richtung wird der Arkustangens von der y zur x-Achse berechnet und anschließend die Korrektur für die Position auf der Erde addiert, um die Ausrichtung nach den Polen exakt zu treffen.

Am Ende wird die Richtung die bisher in Bogenmaß vorliegt noch in das Gradmaß umgerechnet.

4.2.3. RFID/NFC-Shield: Adafruit PN532 NFC/RFID Controller Shield

Mit dem Adafruit PN532 NFC/RFID Controller Shield können RFID-Tags ausgelesen und beschrieben werden. Um es problemlos nutzen zu können, stellt Adafruit die „Adafruit_PN532"-Bibliothek zur Verfügung. Das RFID/NFC-Shield wird mit einer Versorgungspannung von 5V betrieben und kommuniziert ebenfalls über den I2C-Bus mit dem Arduino Board.

Beim Anlegen einer Instanz der „Adafruit_PN532"-Klasse wird die Art der Kommunikation gewählt.

```
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);
```

Anschließend wird die Hardware initialisiert und konfiguriert.

```
nfc.begin();
nfc.SAMConfig();
```

Mit der Funktion „readPassiveTargetID" wird gewartet bis ein RFID-Tag erkannt wird. Sobald dies geschieht gibt die Funktion eine 1 zurück und liefert in den als Parameter übergebenen Variablen die ID des erkannten Tags und die Länge der ID.

```
success_i2c = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A,
      uid_i2c, &uidLength_i2c, 100);
```

Um den Ablauf des Programms nicht zu stören wird ein Timeout von 100ms verwendet. Wurde nach dieser Zeit kein Tag erkannt, liefert die Funktion einen 0 zurück.

4.2.4. Bluetooth-Modem: Sparkfun BlueSMiRF

Das Sparkfun BlueSMiRF ist ein Bluetooth-Modem, das sowohl Daten senden als auch empfangen kann. Die Reichweiter des Bluetooth-Modems beträgt bis zu 10 Meter. Es kann mit Spannungen von 3.3V bis hin zu 6V betrieben werden. In diesem Fall ist es mit den 5V Ausgang des Arduino Boards verbunden.

Das Bluetooth-Modem wird über ein Serielles Interface angesprochen. Daten werden seriell an das Modem weitergegeben, dieses sendet die Daten über Bluetooth weiter zum Empfänger. Empfangene Daten werden wieder seriell an das Arduino Board übergeben. Für die Umsetzung der Seriellen Verbindung wird die „SoftwareSerial"-Bibliothek eingebunden.[2]

Zunächst wird eine Instanz der „SoftwareSerial"-Klasse angelegt.

[2] Sparkfun, https://learn.sparkfun.com/tutorials/using-the-bluesmirf, 10.07.2015

```
SoftwareSerial bluetooth(BLUETOOTH_TX, BLUETOOTH_RX);
```

Zur Vorbereitung der Kommunikation über Bluetooth muss das Bluetooth-Modem konfiguriert werden.

```
bluetooth.begin(115200);
bluetooth.print("$");
bluetooth.print("$");
bluetooth.print("$");
delay(100);
bluetooth.println("U,9600,N");
bluetooth.begin(9600);
```

Mit der begin-Funktion wird die Datenrate zunächst auf die die voreingestellte Datenrate des Bluetooth-Modems eingestellt. Anschließend wird dreimal das „$“-Zeichen gesendet um das Modem in den „command mode“ zu setzen. Jetzt können Einstellungen zur seriellen Datenübertragung vorgenommen werden. Bis zum nächsten Neustart des Bluetooth-Modems wird nun die Datenrate auf 9600 Baude gesetzt. Um die Kommunikation starten zu können wird nun die serielle Verbindung nochmals mit der zuvor festgelegten Baud-Rate gestartet.

Zum Senden einer Nachricht wird lediglich die send-Funktion benötigt. Dieser wird ein Datenarray und die Anzahl der zu sendenden Daten übergeben.

```
bluetooth.write(send_message, 16);
```

4.3. Datenübertragung

4.3.1. I2C

Die Übertragung der Sensordaten des RFID/NFC-Shield und des Kompass-Moduls wird von dem I2C-Bus übernommen.

Der I2C-Bus ist ein synchroner, serieller Zweidraht-Bus mit einer Daten und einer Taktleitung. I2C wird vor allem für die Kommunikation von Integrierten Schaltkreisen (integrated circuit = IC) verwendet. Das Arduino Board wird als Master am Bus verwendet. Zusätzlich können bis zu 127 Slaves am I2C angeschlossen werden. Jeder dieser Slaves muss eine individuelle Adresse besitzen.

Um eine Kommunikation zu starten übernimmt der Master den Bus und sendet die Adresse des Slaves, den er ansprechen möchte. Erkennt der Slave seine Adresse sendet dieser ein Acknowledge auf den Bus. Nach dem Erhalt des Acknowledge gibt der Master an, ob er Daten senden oder Empfangen möchte. Anschließend legt, je nach Aktion, entweder der Master Daten auf den Bus oder der Slave. Nach Abschluss des Senden oder Empfangens gibt der Master den Bus wieder frei. [3]

Zur Nutzung des I2C-Bus wird die „Wire“-Bibliothek eingebunden. Mit der begin-Funktion wird die Buskommunikation vorbereitet. Die eigentliche Datenübertragung über den I2C-Bus wird von den beiden Bibliotheken der Sensoren übernommen.

[3] Microcontroller.net, I2C, http://www.mikrocontroller.net/articles/I%C2%B2C, 10.07.2015

4.3.2. Bluetooth

Die Kommunikation mit der PC-Komponente wurde über eine Bluetooth-Verbindung realisiert. Wie im Kapitel zuvor beschrieben wird hierfür das Bluetooth-Modem BlueSMiRF von Sparkfun verwendet.

Gesendet werden die verarbeiteten Sensordaten die das Arduino Board von den verbundenen Sensoren empfängt. Hierzu gehören die Distanz zum nächsten Hindernis, ein Flag um anzuzeigen ob ein RFID-Tag erkannt wurde, die zugehörige ID des Tags und die Richtung des Roboters. Zeigt das Flag an, dass ein RFID-Tag erkannt wurde, wird die ID übertragen, wurde kein Tag erkannt wird das ID-Feld der Nachricht mit nullen befülllt.

Zum Speichern der Daten wurde global eine Struktur „message“ angelegt. Diese hält die Sensordaten bis zum senden einer Nachricht.

```
struct message
{
  int us_distance;
  byte rfid_tag_found;
  byte rfid_uid[7];
  int com_degrees;
}send_msg;
```

Um die Daten versenden zu können, werden die Daten der Struktur in ein 16 Byte großes Array geschrieben. Hierfür wurde die Funktion msg_to_bytes() implementiert.

Die gesamte Nachricht ist wie folgt aufgebaut:

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
RFID-Flag	ID des RFID-Tags (7 Byte)						
Byte 9	**Byte 10**	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**
Distanz zum Hindernis (4 Byte Integer)				Richtung in Grad (4 Byte Integer)			

Abbildung 10: Nachrichtenstruktur der Bluetoothübertragung von Arduino zur PC-Komponente

Wie zuvor beschrieben wird das Array anschließend seriell an das Bluetooth-Modul weitergegeben und von dort an die PC-Komponente versendet.

4.4. Programmablauf

Das Arduino-Programm wurde mit der von Arduino zur Verfügung gestellten Entwicklungsumgebung erstellt. Die Programme gliedern sich in eine setup-Funktion, die beim Starten des Arduino Boards einmalig ausgeführt wird und eine loop-Funktion, welche nach jedem Durchlauf erneut aufgerufen wird.

In der setup-Funktion werden die Initialisierungen der Seriellen-Schnittstellen und des I2C-Bus durchgeführt. Zudem werden alle genutzten Module initialisiert und die Nachrichten-Struktur mit Initialwerten gefüllt. Am Ende des Setup wird noch ein Timer initialisiert, der zum zyklischen Senden der Nachrichten benötigt wird.

```
int tickEvent = t.every(250, set_timer_var);
```

Nach 250ms wird durch das Ablaufen des Timers die Funktion set_timer_var() aufgerufen. In dieser wird ein Flag gesetzt, welches dem Programm den Beginn eines neuen Sendezyklus signalisiert.

Die loop-Funktion ist nach folgendem Flussdiagramm realisiert:

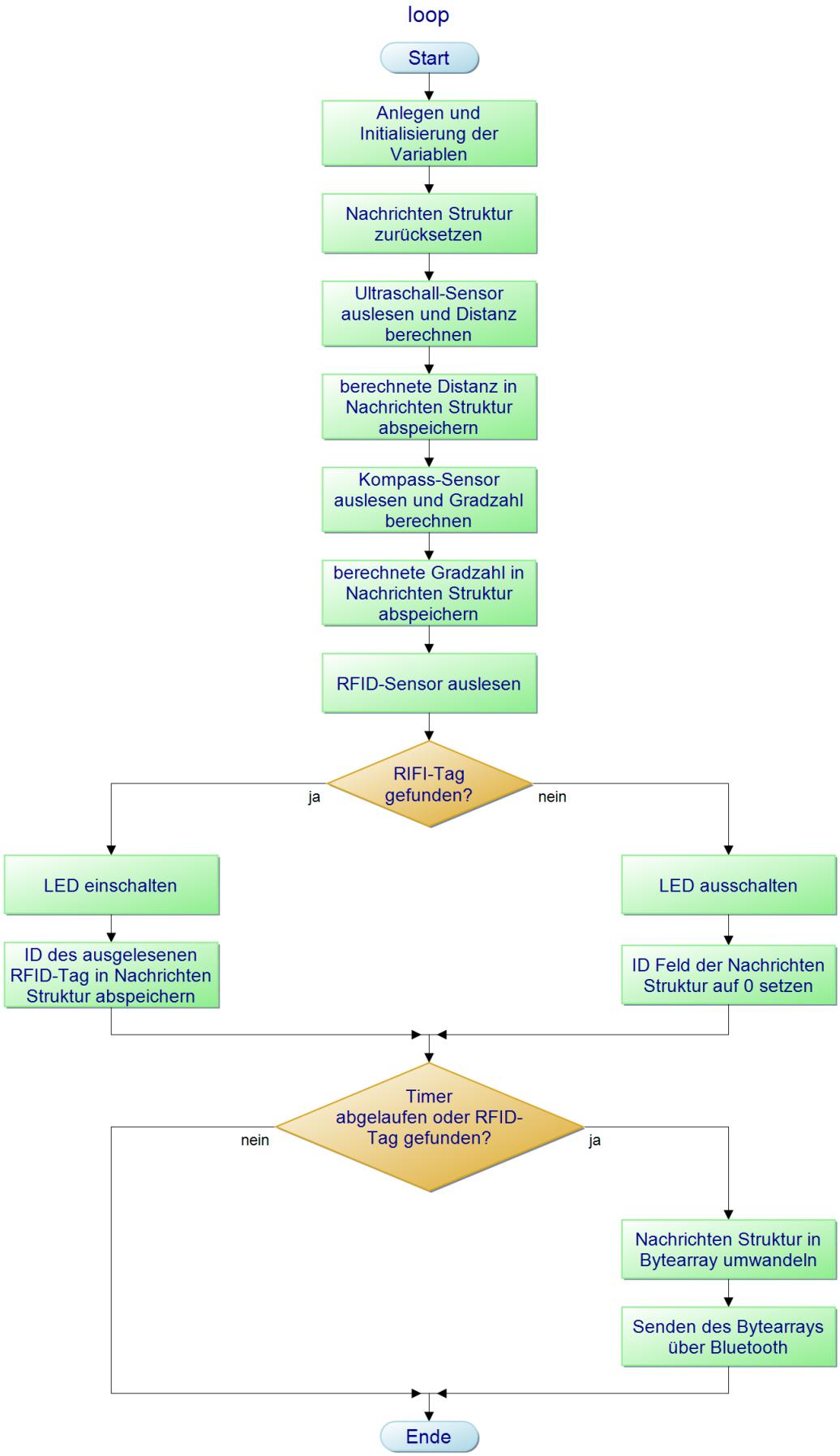

Abbildung 11: Flussdiagramm des Arduino-Programms

Nachdem alle lokalen Variablen, welche zur Verarbeitung der Sensordaten genutzt werden, initialisiert sind, wird die Nachrichten-Struktur auf ihre Initialwerte zurückgesetzt.

Anschließend werden alle Sensordaten abgefragt und verarbeitet. Und am Ende wird geprüft, ob ein neuer Sendezyklus angebrochen ist. Ist dies der Fall, werden die Daten aus der Nachrichten-Struktur in ein Bytearray übertragen und die Nachricht über Bluetooth versendet.

Wurde ein RFID-Tag am RFID/NFC-Shield erkannt wird die aktuelle Nachricht immer gesendet, auch wenn kein neuer Sendezyklus angebrochen ist.

5. PC –Komponente

Am PC findet die zentrale Datenverarbeitung statt. Es werden die Sensordaten vom Arduino Board empfangen, verarbeitet und Steuerbefehle an den Lego Mindstorm EV3-Baustein gesendet.

Die Softwareapplikation wurde mit der Entwicklungsumgebung Visual Studio 2010 in der Programmiersprache C# erstellt.

5.1. Grafische Benutzeroberfläche

Die Visualisierung wurde mithilfe der WinForms Technologie des .Net-Frameworks umgesetzt. Hierfür wurde eine grafische Oberfläche entwickelt wie in Abbildung 1 zu sehen ist. Die Oberfläche dient dem Benutzer als Schnittstelle um den Programmablauf zu steuern. Sie ist, wie nachfolgend beschrieben, logisch in drei Bereiche aufgeteilt.

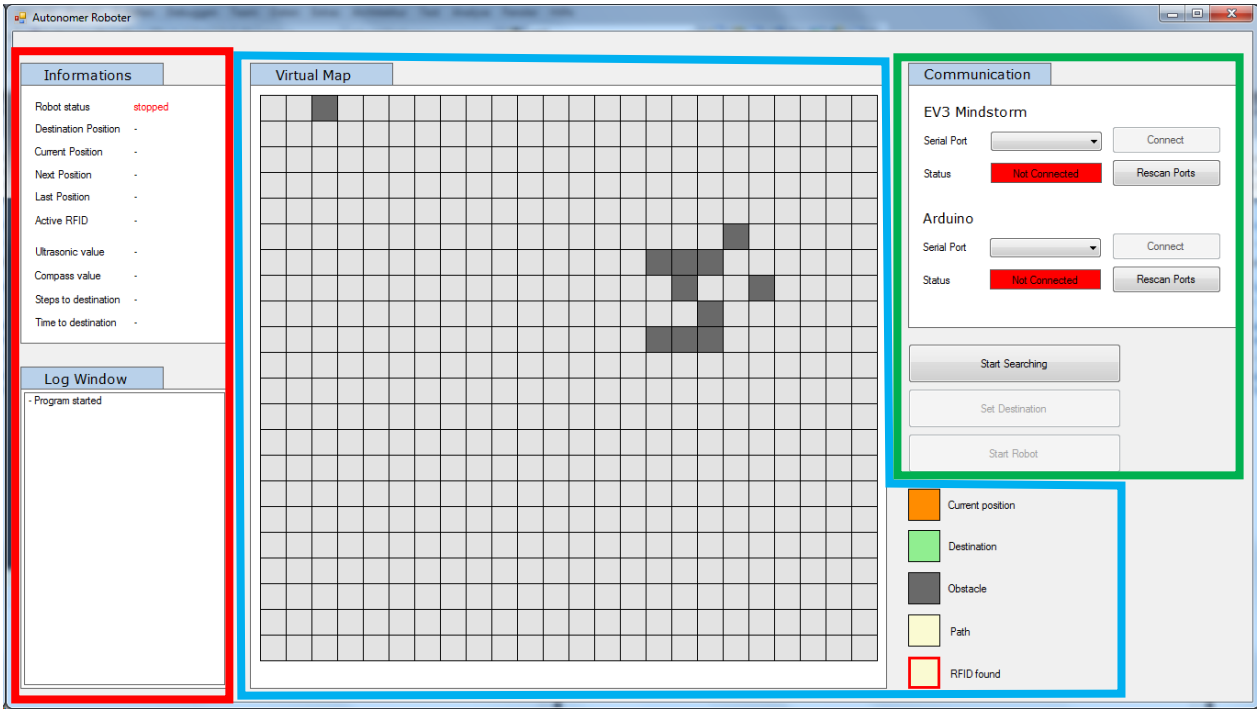

Abbildung 12: Grafische Benutzeroberfläche

Informationsbereich

Der im Bild rot markierte Bereich ist der Informationsbereich. Er Liefert Informationen zu Sensordaten, Programmausgaben, sowie zum Status des Roboters inklusive der aktuellen Position und der Zielposition.

Kartenbereich

Der Kartenbereich ist im Bild blau gekennzeichnet. Er visualisiert die Karte inklusive Legende zu den Elementzuständen (Aktuelle Position, Zielposition, Hindernis, Pfad etc.). Der Bereich stellt den aktuell kalkulierten Pfad sowie alle Hindernisse, die sich auf der Karte befinden dar.

Kontrollbereich

Die Grün markierte Fläche im Bild ist der Kontrollbereich. Er sorgt für die Interaktion des Benutzers mit dem Programm, wie zum Beispiel den Aufbau von Verbindungen mit den Komponenten und das Starten von Abläufen im Programm.

5.2. Algorithmus zur Wegberechnung

Die Berechnung des kürzesten Weges zur Zieladresse wurde mit dem sogenannten Wavefront Algorithmus implementiert. Er ist eine spezialisierte Form des Dijkstra-Algorithmus. Dieser sorgt zunächst für eine Gewichtung der Elemente in einer 2-Dimensionalen Matrix, je nach deren Bezug zu Ziel- und Ausgangsposition. Anschließend wird anhand dieser Gewichtung der Weg, der die wenigsten Elemente beinhält, gefunden.

5.2.1. Grundalgorithmus

Nachfolgend ist die mathematische Beschreibung sowie ein Pseudocode des Wavefront Algorithmus dargestellt.

1. Initialize $W_0 = X_G$; $i = 0$.
2. Initialize $W_{i+1} = \emptyset$.
3. For every $x \in W_i$, assign $\phi(x) = i$ and insert all unexplored neighbors of x into W_{i+1}.
4. If $W_{i+1} = \emptyset$, then terminate; otherwise, let $i := i + 1$ and go to Step 2.

Abbildung 13: math. Beschreibung des Wavefront Algorithmus[4]

```
check node A at [0][0]

now look north, south, east, and west of this node
      (boundary nodes)

if (boundary node is a wall)
      ignore this node, go to next node B

else if (boundary node is robot location && has a number in it)
      path found!
      find the boundary node with the smallest number
      return that direction to robot controller
      robot moves to that new node

else if (boundary node has a goal)
      mark node A with the number 3

else if (boundary node is marked with a number)
      find the boundary node with the smallest number
      mark node A with (smallest number + 1)
```

[4] S. M. LaValle, Planning Algorithms, Cambridge, NY: Cambridge University Press, 2006 Page 312

```
if (no path found)
      go to next node B at [0][1]
      (sort through entire matrix in order)

if (no path still found after full scan)
      go to node A at [0][0]
            (start over, but do not clear map)
      (sort through entire matrix in order)
      repeat until path found

if (no path still found && matrix is full)
      this means there is no solution
      clear entire matrix of obstacles and start over
      this accounts for moving objects! adaptivity[5]
```

Dieser Algorithmus wurde mithilfe des Pseudocodes in C# umgesetzt. Des Weiteren wurde der Algorithmus auf unsere Anforderungen optimiert. Die Optimierung erfolgte indem der Algorithmus mit zwei Erweiterungen ausgestattet wurde.

5.2.2. Erweiterung: Berücksichtigung der Anzahl der Richtungsänderungen

Neben der Auswahl des kürzesten Weges ist es außerdem wichtig, dass es auf diesen Weg zu möglichst wenigen Drehungen des Roboters kommt. Dies hat zwei Gründe. Zum einen nimmt die Genauigkeit der Position des Roboters bei jeder Drehung ab. Zum anderen erhöht sich die Zeit, die bis Erreichen des Ziels benötigt wird mit der Anzahl der Drehungen.

Deswegen wurde der Algorithmus so erweitert, dass er bei der Auswahl des nächsten Elements für den Pfad, bevorzugt das Element auswählt, das in gleicher Richtung wie das Element der vorherigen Auswahl liegt. Somit kommt es automatisch nur zu einem Richtungswechsel, wenn dies zwingend notwendig ist.

Die folgenden Bilder zeigen die Pfadauswahl vor und nach der genannten Erweiterung.

[5] Societyofrobots, http://www.societyofrobots.com/programming_wavefront.shtml, 18.05.2015

8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	5	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
9	8	7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
10	9	8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
11	10	9	8	7	6		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
12	11	10	9	8	7			10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
13	12	11	10	9	8		12		12	25	14	15	16	17	18	19	20	21	22	23	24	25	26
14	13	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
15	14	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
16	15	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
17	16	15	14	13	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Abbildung 14: Pfad vor Berücksichtigung der Anzahl der Richtungsänderungen

8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	5	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
9	8	7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
10	9	8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
11	10	9	8	7	6		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
12	11	10	9	8	7			10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
13	12	11	10	9	8		12		12	25	14	15	16	17	18	19	20	21	22	23	24	25	26
14	13	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
15	14	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
16	15	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
17	16	15	14	13	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Abbildung 15: Pfad nach Berücksichtigung der Anzahl der Richtungsänderungen

Wie in dem Beispiel zu sehen ist, wurde durch die Erweiterung die Anzahl der Drehungen von sieben auf eine Drehung minimiert.

5.2.3. Erweiterung: Festlegung der Startrichtung

Eine andere Erweiterung ist eine Festlegung in Welche Richtung der Algorithmus mit der Auswahl der Elemente zur Pfaderstellung beginnt. Auch dieser Algorithmus sorgt für eine Minimierung der Richtungsänderungen. Bevor der Algorithmus das erste Element auswählt werden die zwei Richtungen, die zur Zielposition zeigen, überprüft. Es wird die Richtung ausgewählt, die kein oder das weiter entfernte Hindernis besitzt. Somit wird garantiert, dass der Algorithmus schon vor Beginn erkennt, dass in welcher Richtung ein Hindernis und somit eine Richtungsänderung auftreten wird und dadurch die bessere Richtung auswählt.

Die folgenden Bilder zeigen die Pfadauswahl vor und nach der genannten Erweiterung.

8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	5	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
9	8	7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
10	9	8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
11	10	9	8	7	6		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
12	11	10	9	8	7			10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
13	12	11	10	9	8		12		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
14	13	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
15	14	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
16	15	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
17	16	15	14	13	12	13	25	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Abbildung 16: Pfad vor Festlegung der Startrichtung

8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	5	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
9	8	7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
10	9	8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
11	10	9	8	7	6		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
12	11	10	9	8	7			10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
13	12	11	10	9	8		12		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
14	13	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
15	14	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
16	15	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
17	16	15	14	13	12	13	25	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Abbildung 17: Pfad nach Festlegung der Startrichtung

In diesem Beispiel kommt es zu einer Minimierung der Richtungsänderung um eins.

5.3. Bluetooth Kommunikation

Eine weitere wichtige Aufgabe ist die Kommunikation zum Arduino Board zur Sensordatenerfassung und zum Lego Mindstorm EV3 Baustein zur Bewegungssteuerung. Die Kommunikation wurde mithilfe der von Windows zur Verfügung gestellten seriellen COM Ports implementiert. Diese bilden eine serielle Schnittstelle zur Kommunikation zu Peripheriegeräten wie beispielsweiße einem Bluetooth-Modul. Im .Net-Framework werden hierfür Klassen definiert, die den Zugriff auf diese Schnittstellen erleichtern.

Datenaustausch mit dem Arduino Board

Die Sensordaten werden vom Arduino Board via Bluetooth zum PC gesendet. Diese Daten werden zyklisch vom Programm empfangen und in eine interne Datenstruktur gespeichert. Die gesendeten Telegramme sind sehr simpel gehalten wie in Kapitel 3.1. beschrieben und die entsprechenden Sensorwerte können somit einfach den internen Datenstrukturen zugewiesen werden.

Datenaustausch mit dem Lego Mindstorm EV3-Baustein

Die Kommunikation mit dem Lego Mindstorm EV3-Baustein ist sehr komplex. Lego bietet hierfür keine offizielle Bibliothek die das Protokoll zur Kommunikation implementiert. Nach einiger Recherche wurde eine Open-Source-Bibliothek eines Drittanbieters gefunden, die Methoden zur Kommunikation mit dem Lego Mindstorm EV3-Baustein zur Verfügung stellt. Diese Bibliothek bietet sowohl die Möglichkeit zum Senden als auch zum Empfangen von Daten. Zudem sind Methoden implementiert, die eine Erstellung der Kommunikationsverbindung über einen COM-Port erlauben. Wie ein Telegramm aufgebaut ist wird im Kapitel 3.3 beschrieben.

5.4. Programmstruktur

Das PC-Programm umfasst ca. 2000 Zeilen Programmcode, wobei das Hauptprogram inklusive Oberfläche 1444 Zeilen Code enthält. Aufgrund des Umfangs und der Komplexität wurde das Programm in eine Vielzahl von Klassen für verschiedene Aufgaben unterteilt. Diese Klassen sind in verschiedenen Modulen implementiert. Nachfolgend ist die Klassenstruktur in Form eines UML-Diagramms dargestellt.

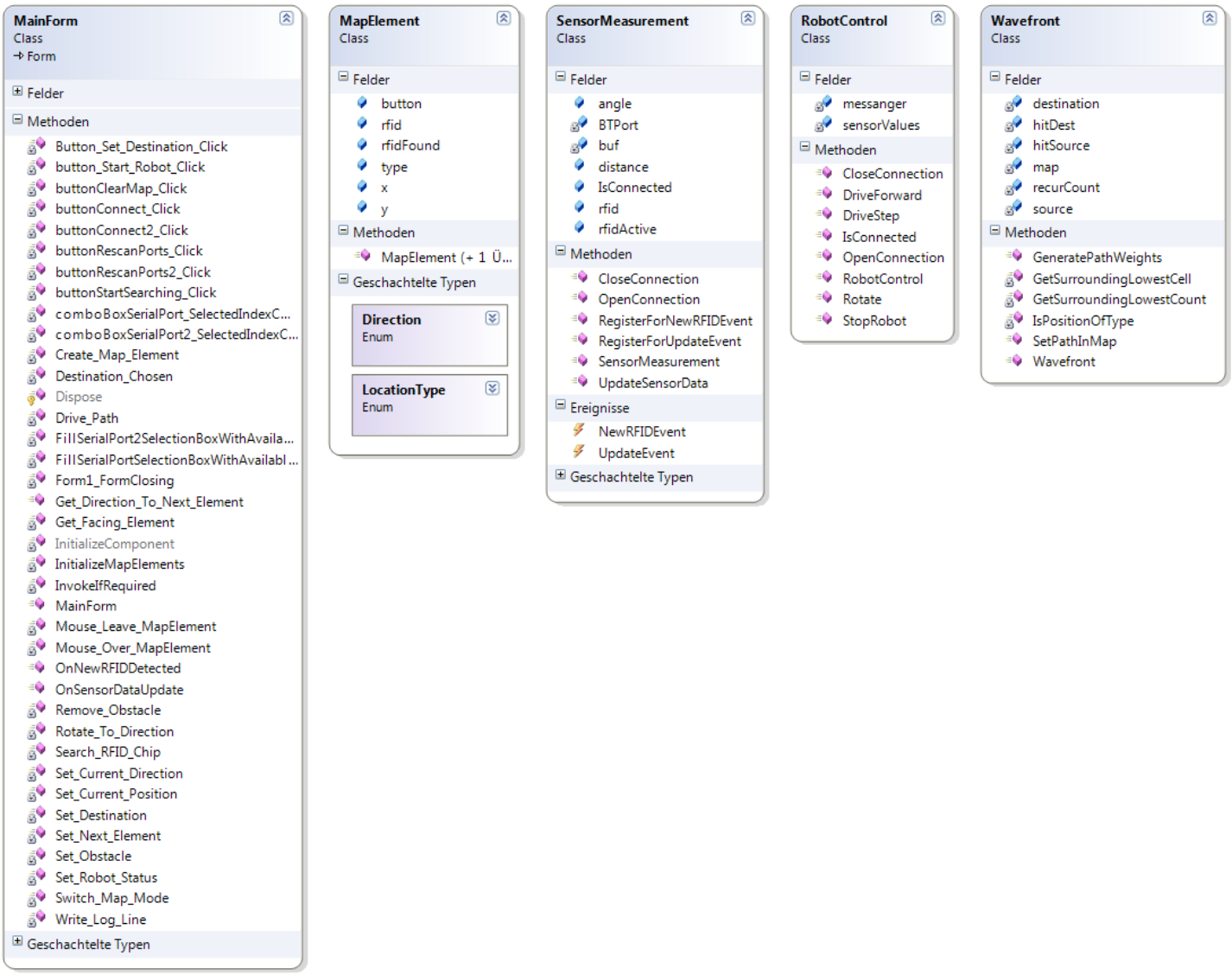

Abbildung 18 Klassendiagramm

Klasse MainForm

Diese Klasse ist für die Realisierung der grafischen Benutzeroberfläche sowie für die Auswertung der Benutzereingaben zuständig. Sie steuert den Ablauf des kompletten Programms.

Klasse MapElement

Diese Klasse repräsentiert ein logisches Element der virtuellen Karte.

Klasse SensorMeasurement

Diese Klasse sorgt für die Bereitstellung der Sensordaten. Sie implementiert die Kommunikation mit dem Arduino Board und speichert die Werte der Sensoren intern.

Klasse RobotControl

Diese Klasse ist für die Steuerung des Roboters zuständig. Es sind verschiedene Methoden implementiert, die einen bestimmten Vorgang des Roboters startet indem Befehle an diesen gesendet werden.

Klasse Wavefront

Die Klasse Wavefront ist zur Implementierung des Wavefront-Algorithmus zur Wegberechnung zuständig.

Bibliothek EV3Messanger

Die Bibliothek implementiert die Kommunikation zum Lego Mindstorm EV3-Baustein und wurde mit einer Open-Source-Lizenz in unser Programm übernommen. Quelle des Source-Codes ist folgende:

https://ev3messenger.codeplex.com/

5.5. Programmablauf

Der Programmablauf ist sehr komplex, da es viele verschiedene Benutzereingaben gibt und es somit zu zahlreichen internen Zuständen und Arbeitsabläufen kommt. Aufgrund der verschiedenen Aufgaben, wie beispielsweise die Kommunikation zu den Komponenten oder das Abfragen von Benutzereingaben entstehen Programmabläufe, die teilweise parallel, also gleichzeitig ablaufen müssen. Diese Aufgaben wurden mithilfe verschiedener Threads implementiert. So werden prinzipiell vier verschiedene Threads unterschieden, die je nach Ablauf aktiv sind. Jeder der Threads ist für eine spezielle Aufgabe verantwortlich und sorgt für einen speziellen Programmablauf. Nachfolgend wird mithilfe von Programmablaufplänen der Ablauf der verschiedenen Threads vereinfacht dargestellt.

UpdateDataThread

Dieser Thread wird aktiv, sobald es zu einer erfolgreichen Verbindung zum Arduino Board kommt. Er liest zyklisch die Sensordaten von dem COM-Port, speichert diese und reagiert mit dem Starten eines oder zweier Events, die wiederum für eine weitere Reaktion sorgen.

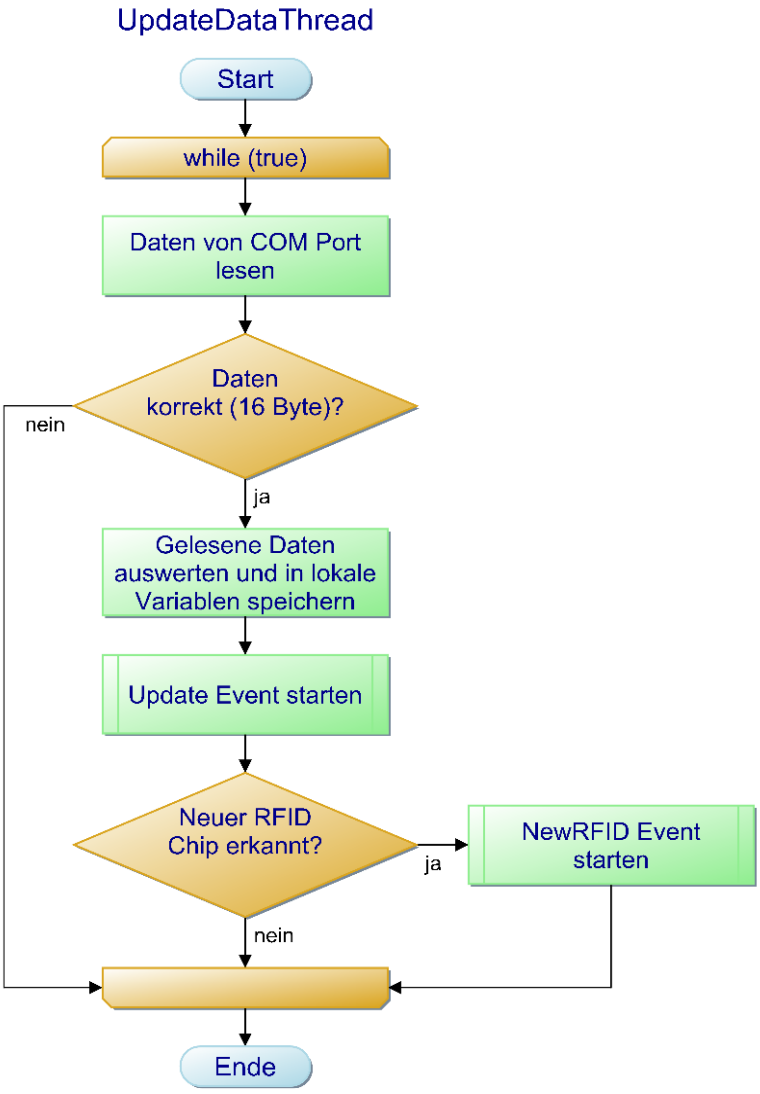

Abbildung19:UpdateData Thread

DrivePathThread

Dieser Thread ist für die Navigation des Roboters zu einer ausgewählten Zieladresse zuständig. Er wird gestartet sobald der Benutzer den Startvorgang des Roboters betätigt. Der Thread sorgt dafür, dass der Roboter über den zuvor kalkulierten Weg an der Zieladresse ankommt.

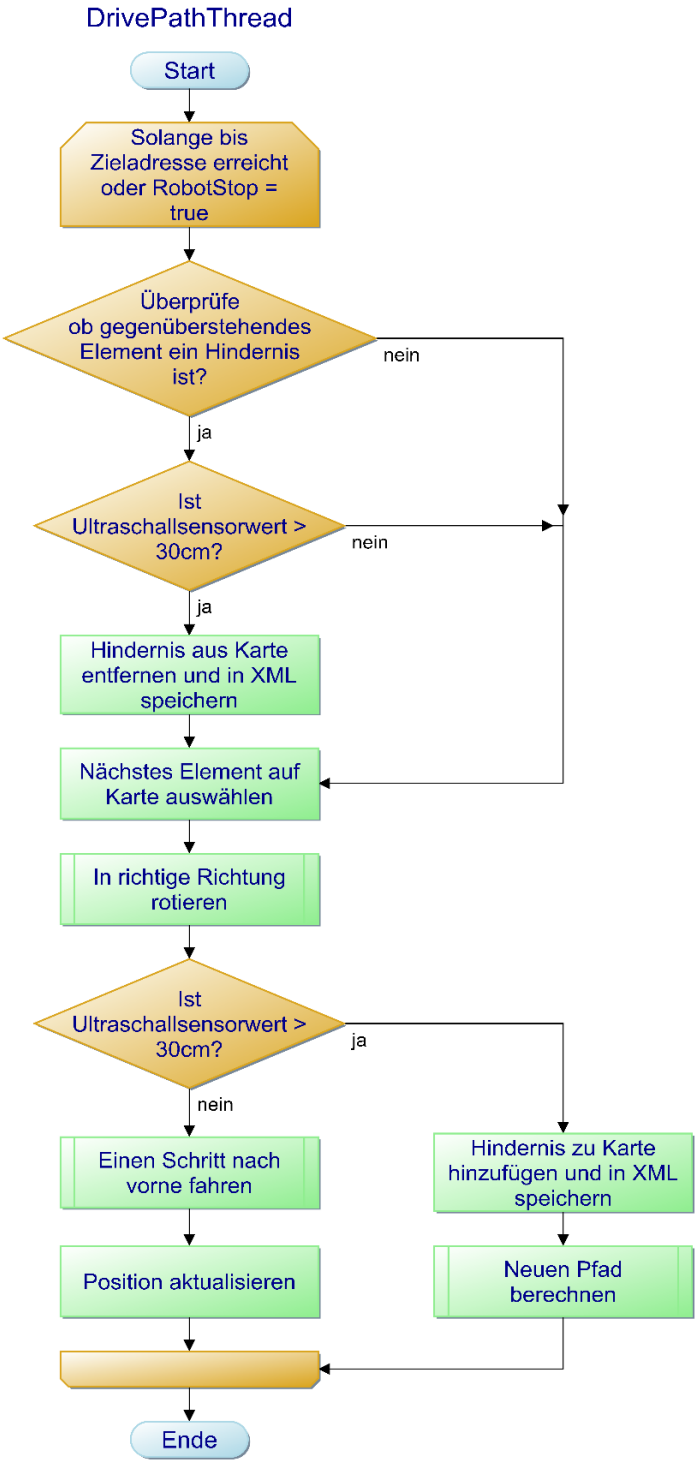

Abbildung 20: DrivePath Thread

SearchRFIDThread

Dieser Thread sorgt für das Abarbeiten des Suchalgorithmus zum Finden eines RFID-Tags. Er steuert den Roboter so an, dass dieser die in Kapitel 2.3. beschriebene Form fährt. Wird ein RFID-Tag detektiert wird dieser Thread automatisch von einen anderen Thread beendet.

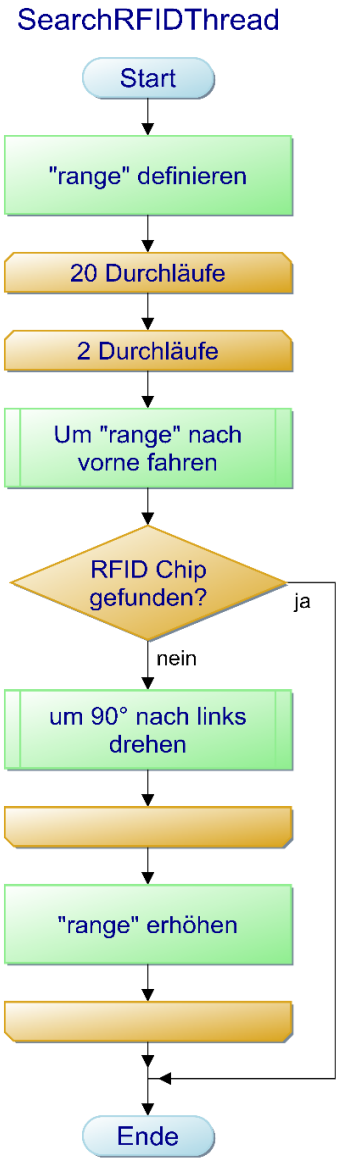

Abbildung 21:SearchRFID Thread

Main Thread

Dieser Thread Hauptthread, ist also während der kompletten Programmausführung aktiv. Er startet die Oberfläche, reagiert auf Benutzereingaben und steuert die anderen Threads.

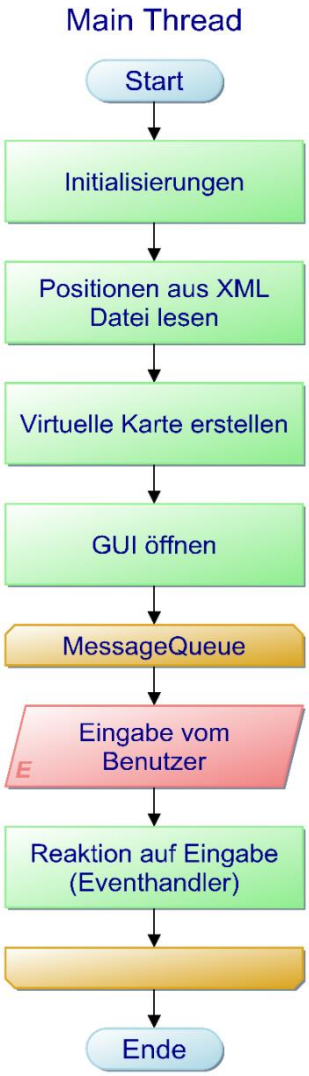

Abbildung 22: Main Thread

6. Mindstorm EV3 Programm

6.1. Das Hauptprogramm

Auf dem Roboter läuft ein Programm, das die Steuerbefehle, die via Bluetooth vom PC gesendet werden, auswertet und damit die Motoren ansteuert. Es ist zu ergänzen, dass der EV3 Stein nicht dazu gedacht ist, Nachrichten per PC zu empfangen. Der Mindstorms EV3-Stein wird sonst nur via Bluetooth per App angesteuert. Eine C#-Library aus dem Internet baut das Protokoll nach und ermöglicht so sämtliche Befehle an den Roboter senden zu können.

Das Hauptprogramm folgt folgendem Ablaufplan:

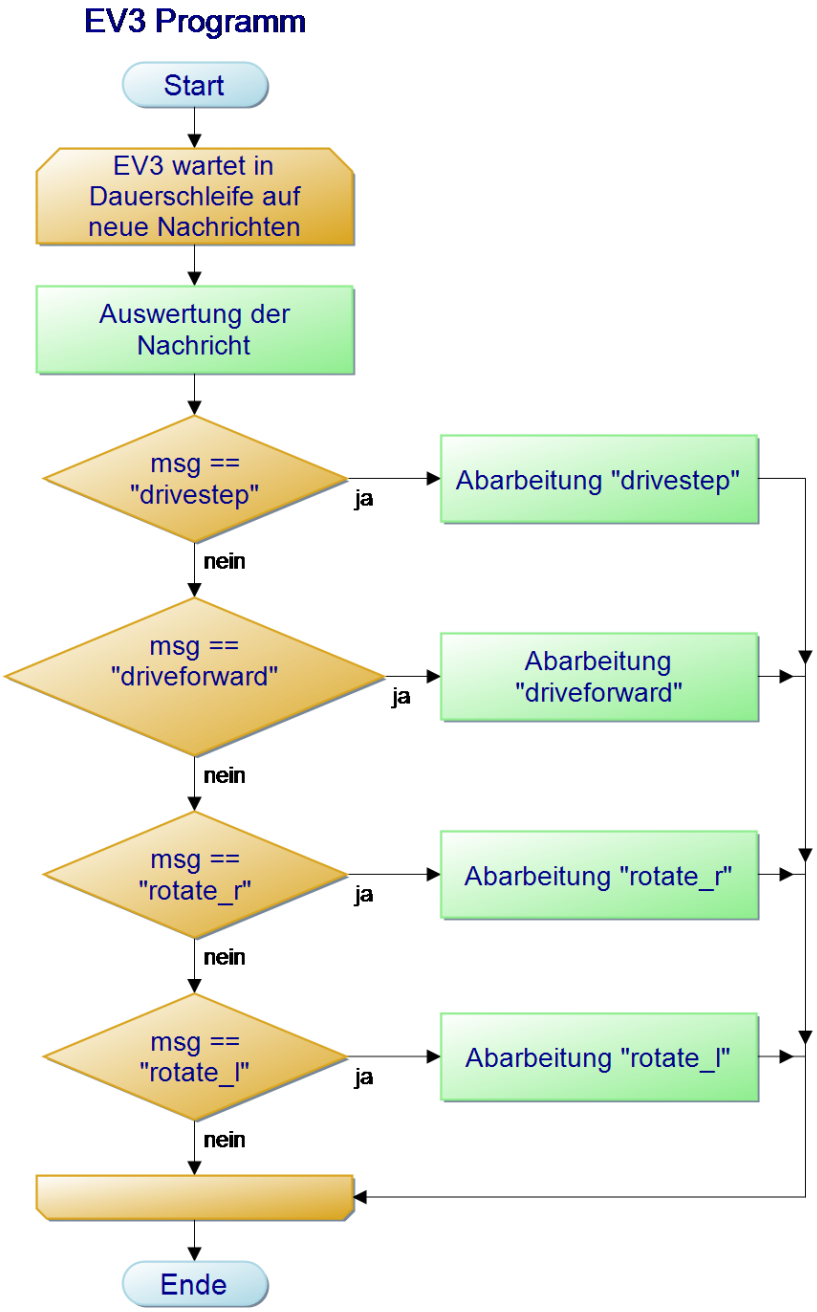

Abbildung 23: Ablaufplan Hauptprogramm

Im „EV3-Programmcode“ ist der Ablauf folgendermaßen implementiert:

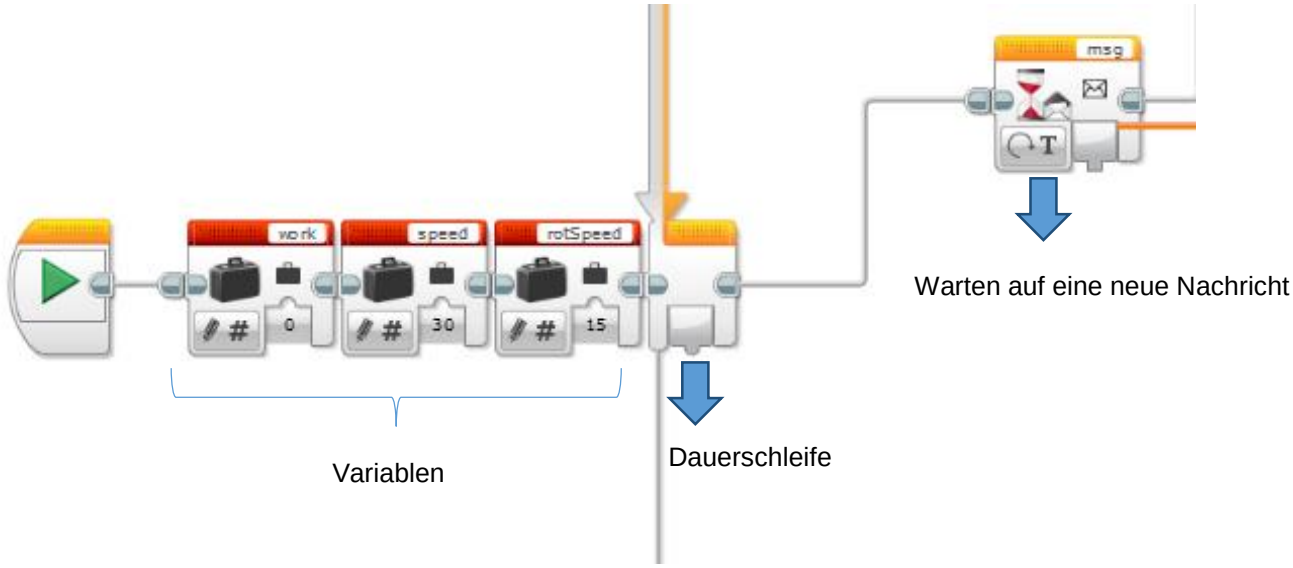

Abbildung 24: EV3 Hauptprogramm Teil1

Im ersten Teil werden ein paar Variablen vordefiniert. Danach geht es in eine Dauerschleife. In dieser Dauerschleife wartet der EV3-Stein auf eine neue Nachricht, die per Bluetooth geschickt wird. Diese Nachricht wird dann in einer Art „Switch/Case“-Anweisung ausgewertet.

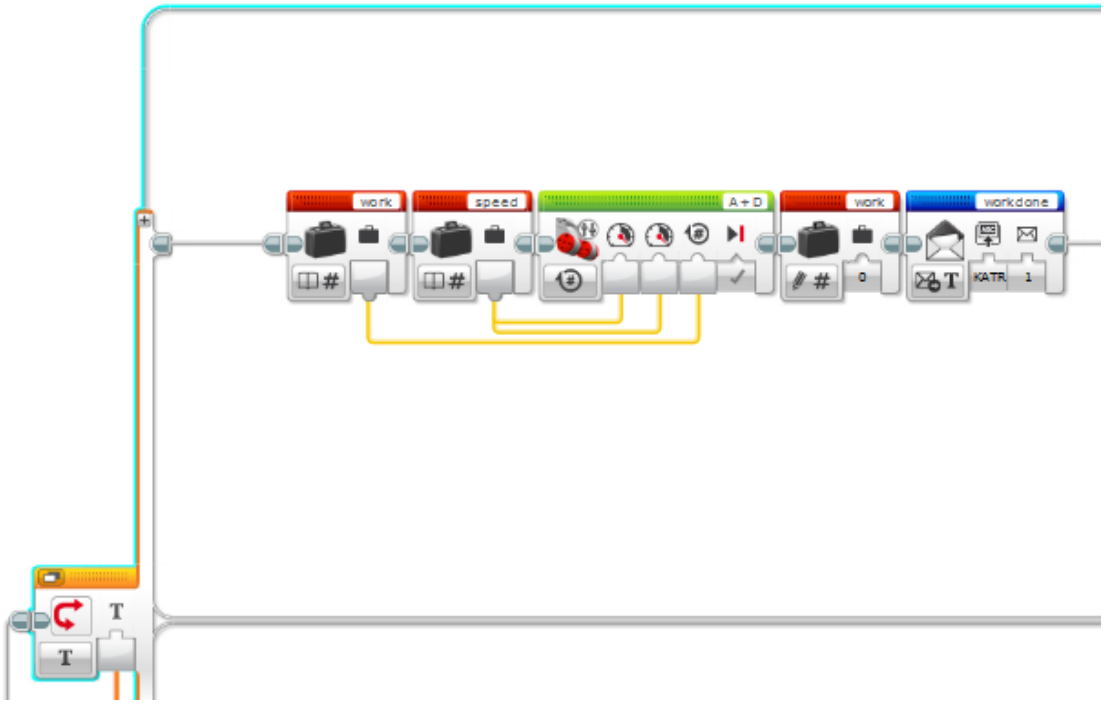

Abbildung 25: EV3 Hauptprogramm Teil2

Die Nachrichten werden im Folgenden dann genauer beleuchtet.

6.2. Drivestep

Diese Funktion ermöglicht es dem Roboter genau von Tag zu Tag zu fahren. Das wird gebraucht, damit der Roboter sich auf der „Karte“ von Element zu Element bewegen kann. Durch Ausprobieren wurde herausgefunden, dass die Rad-Umdrehungen, die man benötigt, um von Element zu Element zu fahren bei ca. 1,5 Umdrehungen je Rad liegen.

Folgender Programmablauf wurde hierzu erstellt:

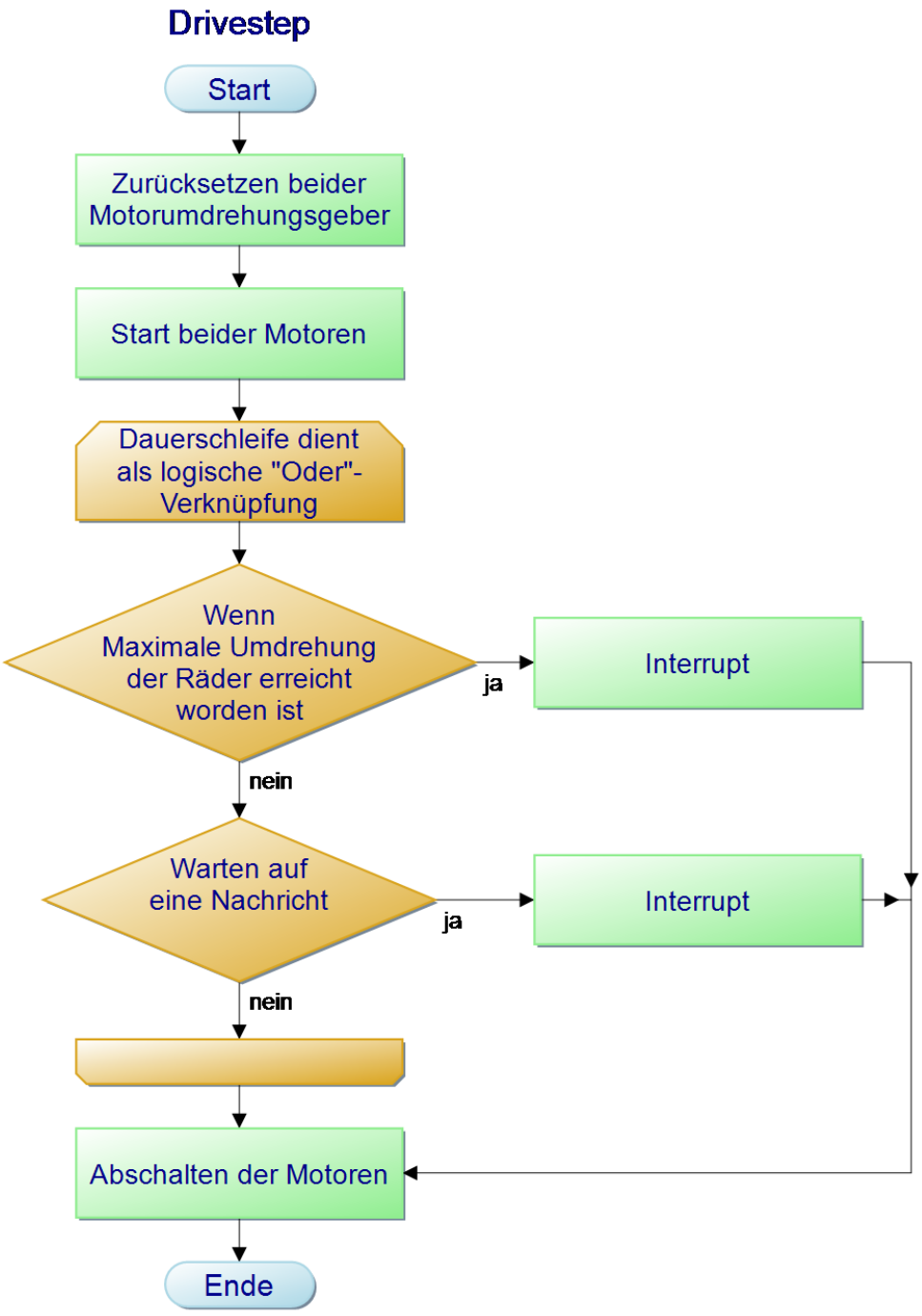

Abbildung 26: Ablaufplan Drivestep

Im „EV3-Programmcode“ ist der Ablauf folgendermaßen implementiert:

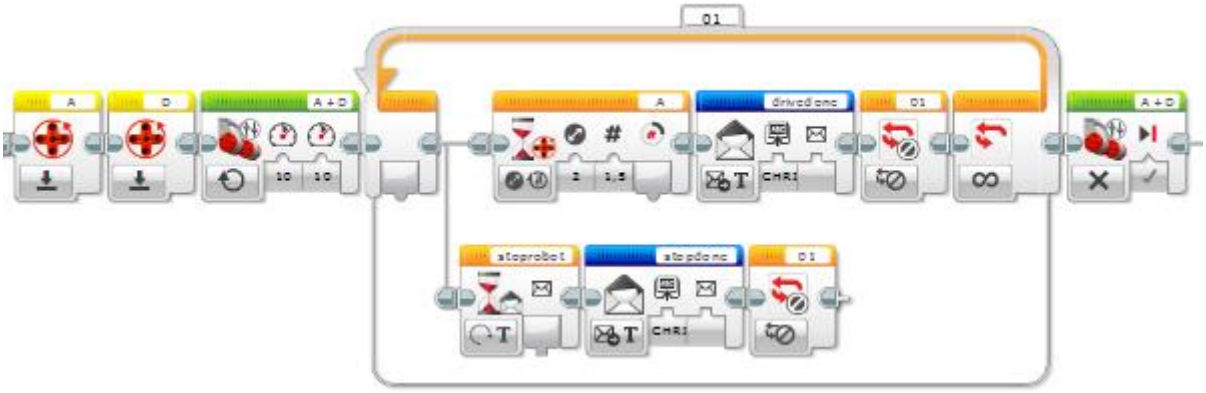

Abbildung 27: EV3 Programm Drivestep

Am Anfang werden die Rotationszähler beider Motoren zurückgesetzt. Danach werden die Motoren, die in den Steckplätzen A und D stecken mit der Geschwindigkeit von 10% gestartet. Anschließend geht das Programm in eine Dauerschleife mit zwei Zweigen. Im oberen Pfad wird so lange gewartet bis die Räder 1,5 Umdrehungen gemacht haben. Danach wird eine Nachricht an den PC geschickt, die Schleife wird mit dem darauf folgenden Interrupt unterbrochen. Nach der Schleife werden die Motoren gestoppt.

Im unteren Pfad wird auf eine Nachricht an die Mailbox „stoprobot“ gewartet. Ist diese eingetroffen so wird eine Nachricht an den PC geschickt. Der folgende Interrupt beendet die Dauerschleife. Auch hier kommt es danach zu einem stoppen der Motoren.

Diese Konstruktion stellt eine „logische Oder“-Verknüpfung dar. Entweder wartet man 1,5 Umdrehungen oder der EV3 Stein erhält eine Nachricht vom PC. Beides führt dazu, dass die Motoren am Ende gestoppt werden. Im Endeffekt ist es somit möglich die Fahrt des Roboters jederzeit zu unterbrechen, wenn er beispielsweise auf einen Tag fährt. Sonst könnte es passieren, dass der Roboter über das Ziel hinausfährt. Das Anhalten des Roboters während der Fahrt wurde in der endgültigen Umsetzung jedoch nicht genutzt, da dieser mit den eingestellten Werten schon sehr genau von Element zu Element fährt.

6.3. Driveforward

Um bei der Suchfunktion sehr genau fahren zu können, wird eine Funktion benötigt, die es dem Roboter erlaubt bei jeder Fahrt die Distanz zu variieren. Dieses Problem wurde mit der „driveforward“ Funktion behoben. Bei dieser Funktion lässt sich die genaue Anzahl der Umdrehungen angeben.

Folgender Programmablauf wurde hierzu erstellt:

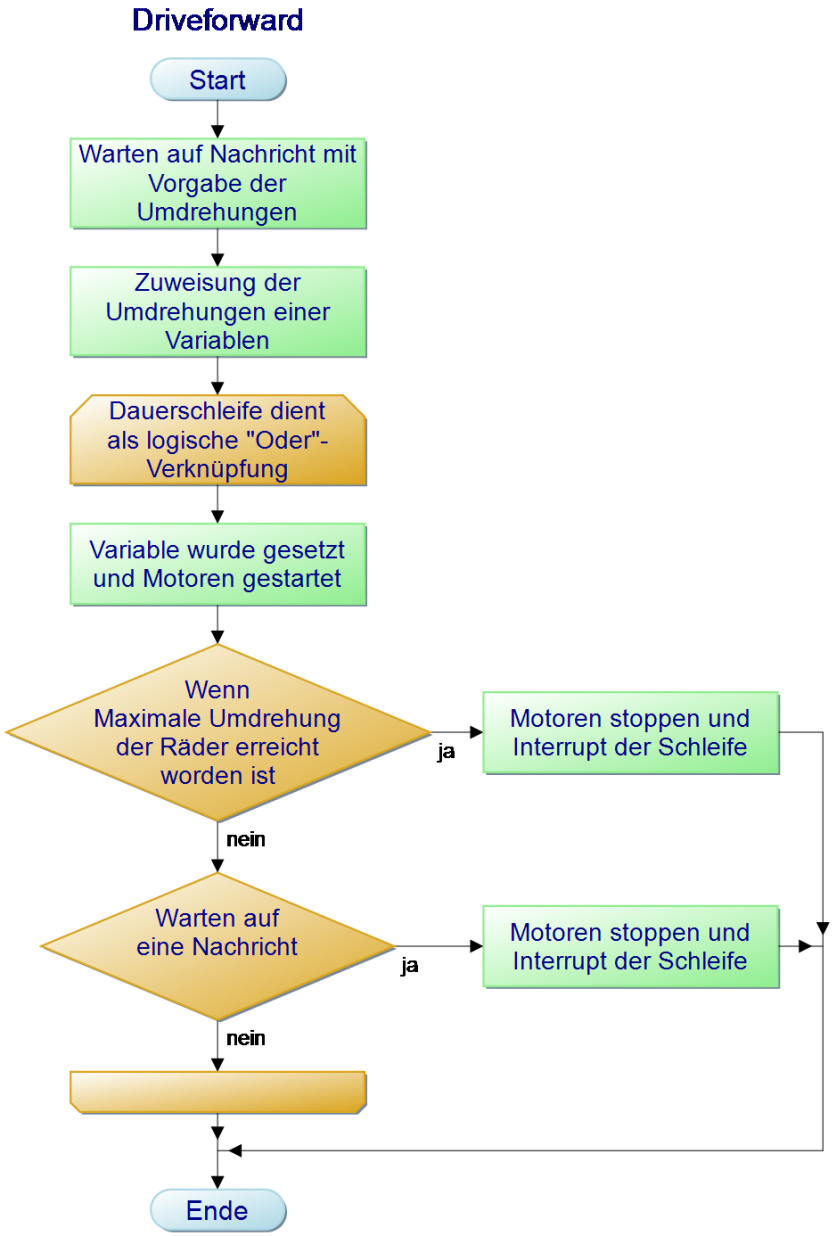

Abbildung 28: Ablaufplan Driveforward

Der EV3-Programmcode dazu sieht wie folgt aus:

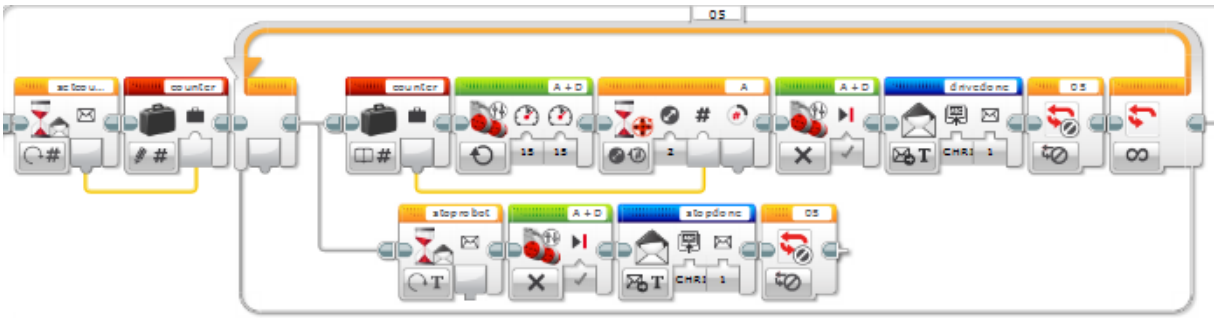

Abbildung 29: EV3 Programm Driveforward

Nach Abschicken des Befehls „driveforward“ sendet der PC noch eine Nachricht an den EV3-Stein. In dieser Nachricht wird die Anzahl der Umdrehungen übergeben. Danach geht es wieder in die Oder-Konstruktion. Im Oberen Pfad wird wieder gewartet bis die vorgegebenen Umdrehungen erreicht worden sind. Im Anschluss daran werden die Motoren gestoppt. Die erfolgreiche Fahrt wird mit einer Nachricht an den PC quittiert.

Im unteren Pfad wird wieder auf eine Nachricht gewartet. Nach Erhalt werden die Motoren gestoppt. Ein erfolgreicher Stopp wird mit einer Nachricht quittiert. Nach beiden Zweigen wird die Schleife durch einen Interrupt unterbrochen.

Die Möglichkeit die Fahrt zu unterbrechen ist bei dieser Funktion essentiell. Würde es diese nicht geben, würde der Roboter bei der Suchfunktion möglicherweise über einen Tag hinaus fahren und somit wäre es reiner Zufall, dass der Roboter auf einem Tag zum Stehen kommt.

6.4. Rotate_l/Rotate_r

Der Roboter muss auf seinem Weg zum Ziel auch Drehungen vollführen oder gegebenenfalls Hindernissen ausweichen.

Die Links- und Rechtsdrehung wurde nach folgendem Ablauf implementiert:

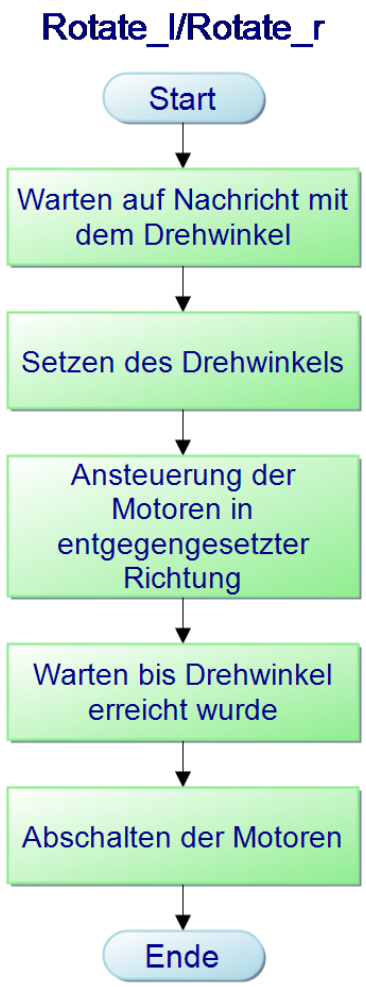

Abbildung 30: Ablaufplan rotate_l/rotate_r

Im EV3-Programm wurde das folgendermaßen umgesetzt:

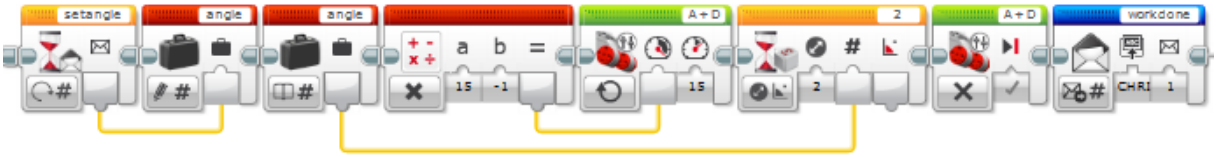

Abbildung 31: EV3 Programm rotate_l/rotate_r

Nachdem der Befehl „rotate_l“ an den EV3-Stein gesendet wurde, wartet der EV3-Stein auf eine weitere Nachricht mit dem Drehwinkel. Dieser wird in die Variable „angle“ geschrieben. Danach werden beide Motoren mit entgegengesetzten Signalen angesteuert, um eine Drehung zu erreichen. Die jeweils entgegengesetzte Ansteuerung rührt daher, dass der Roboter sich auf der Stelle drehen soll. Blockiert man ein Rad und lässt nur das kurvenäußere Rad drehen, würde der Roboter nicht mehr auf dem Tag stehen.

Der Lego Kreiselsensor unterstützt hier bei der Drehung. Er misst die aktuelle Winkeländerung. Hat der Kreiselsensor die vorgegebene Rotation detektiert, so werden die Motoren wieder gestoppt. Am Ende wird eine „1“ an den PC gesendet. Bei der Funktion „rotate_r“ ist die Ansteuerung andersherum.

Da der Kreiselsensor auch eine Abweichung von 2-3° Grad hat, wurde der übergebene Winkel (± 90°)empirisch durch viele Messungen ermittelt.

6.5. Das Telegramm

Jeder aufgeführte Befehl wird über ein Telegramm an den EV3-Stein geschickt. Dabei muss das Telegramm folgende Informationen mindestens tragen:

- Mailbox-Titel
- Message-Type
  - Number
  - Text
  - Bool
- Message

Der Mailbox-Titel ist eine Zuordnung im EV3-Programm. Man kann den Empfangsbausteinen einen Namen geben. Im Hauptprogramm wird der Steuerbefehl zuerst an Baustein mit dem Titel „msg“ geschickt. Im Fall von „rotate_l“ wartet der Baustein noch auf eine weitere Nachricht. Die Nachricht muss hier dann an die Mailbox mit dem Namen „setangle“ gesendet werden.

Bei der Konfiguration eines Nachrichtenempfangsbausteins kann man aus drei verschiedenen Datentypen wählen (Number, Text, Bool). Dies ist immer abhängig von der jeweiligen Anwendung. Die Nachrichtenbox „msg“ wartet auf einen Befehl im Textformat. Wohingegen die Nachrichtenbox „setangle“ auf eine Zahl wartet.

Die Konfiguration ist somit abhängig vom Inhalt der Nachricht.

7. Fazit

Abschließend kann gesagt werden, dass das Projekt trotz einiger Probleme zu einem sehr guten Ergebnis geführt hat. Der Roboter navigiert autonom im Raum, erkennt und umfährt Hindernisse zuverlässig und erreicht das richtige Ziel in fast allen Fällen. Dabei wird stets der kürzeste Weg abgefahren ein deterministisches Verhalten eingehalten.

Während der gesamten Projektlaufzeit ist das Team auf verschiedene Probleme gestoßen.

Zum einen gab es einige Probleme mit dem Kompass-Modul. Die gelieferten Werte waren sehr ungenau. Die Bereiche, in denen sich die Werte des Moduls änderten, waren nicht einheitlich und somit nicht brauchbar. Nachdem ein zweites Kompass-Modul bestellt wurde, um die Möglichkeit des Defekts auszuschließen, ist beim Anbringen an den Roboter ein ähnliches Verhalten aufgetreten. Hat das Modul abseits des Roboters betrieben funktionierten es jedoch einwandfrei. Aus dieser Begebenheit lässt sich der Schluss ziehen, dass die Störeinflüsse der anderen Sensoren und des Lego Mindstorm EV3-Bausteins die Werte des Kompass‘ so weit verfälschen, dass diese nicht mehr tauglich für eine Verwendung im Projekt sind.

Das Kompass-Modul am EV3-Baustein ist ebenfalls etwas ungenau, wodurch es nach mehreren Drehungen zu einer Abweichung kommen kann. Da die Fläche, in der ein RFID-Tag erkannt wird, nur wenige Zentimeter groß ist, kommt es vor, dass RFID-Tags nicht erkannt werden. Bei längeren Strecken kann diese Abweichung dazu führen, dass der Roboter sein Ziel nicht exakt erreicht. Durch den implementierten Suchalgorithmus wird in diesem Fall jedoch meistens sichergestellt, dass der Ziel-RFID-Tag gefunden wird.

Ein weiteres Problem gab es beim Anbringen eines zweiten RFID-Readers an dem Roboter. Da die beiden RFID-Reader vom gleichen Hersteller und vom gleichen Typ waren, konnten nicht beide über den I2C-Bus angesteuert werden. Der RFID-Reader dieses Typs besitzt eine voreingestellte Adresse für den I2C-Bus die, auch nach längerer Recherche, nicht verändert werden konnte. Das führte dazu, dass nur ein Reader benutzt werden konnte. Ein RFID-Reader eines anderen Herstellers konnte aufgrund der geringen Reichweite nicht eingesetzt werden.

Als letztes aufzuführendes Problem ist die fehlerhafte Kommunikation zwischen der EV3 und der PC-Komponente zu nennen. Ab und zu kamen alte Signale des EV3 verspätet bei der PC-Station an. Dies hatte zur Folge, dass Kommandos quittiert wurden, welche noch nicht vollendet waren. Das Resultat war eine fehlerhafte Steuerung. Gelöst wurde das Problem, indem vor jedem Sendevorgang der Puffer mit den anstehenden Nachrichten geleert wird.

Durch die Vielzahl an Threads, die teilweise im PC-Programm parallel laufen, wurde eine Fehlersuche erschwert. Zudem war die korrekte Synchronisation der Threads mit sehr viel Zeit- und Testaufwand verbunden.

8. Abbildungsverzeichnis