Technische Hochschule Wildau [FH]
Technical University of Applied Sciences
WILDAU

Projekt: Nibahn

Teammitglieder: Alexandr Petrov, Arina Lavrova, Jan Bischof

Nibos: Darwin, Optimus, Nope. No1

Bearbeitungszeitraum: 22.10.2019 – 02.12.2019

Betreuende Hochschullehrerin: J Prof. Dr. rer. nat. Mohnke, Janett

Fachhochschule: TH-Wildau

Studiengang: Telematik

Fachbereich: Eingebettete Systeme und Robotik - Teil 1

Semester: 5

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Im Rahmen vom Fachbereich „Eingebettete Systeme und Robotik 1“ an der TH-Wildau haben wir kleinen Robotern gekriegt, die von einem Hauptcontroller und einem Co-Controller gesteuert sind, haben ganze Menge von Sensoren, können sich bewegen und auch die Daten auf den Display zeigen. Als Aufgabenstellung haben wir Pflicht- und Kuraufgaben bekommen. Pflichtaufgaben waren vom Anfang ganz deutlich definiert und wir mussten dementsprechend Lösungen entwickeln. Als Kürafugaben mussten wir Kommunikation zwischen Nibos oder zwischen Nibo und PC realisieren mit Nutzung von NXB2-Modulus, diese mit mindestens 2 weitere Funktionen kombinieren und inzwischen Ressourcen sparen mit passende Datenstrukturen. Wie wir es machen können, war die Aufgabe für unsere Fantasie und wir wollten nicht nur die Funktionen zu realisieren, sondern auch die sie für ein gemeinsames Zweck zu kombinieren. So wurde das Konzept vom Projekt “Nibahn“ erstellt.

1.2. Projektvorhaben

Als Pflichtaufgaben müssen folgende Funktionen realisiert werden. Nibo muss in der Lage sein, Hindernisse zu erkennen und zu umgehen. Es gibt auch die besonderen Situationen (Tunnel und Sackgasse) mit denen Nibo schaffen muss. Die erkennten Hindernisse müssen optisch dargestellt werden.
Als Kürafugabe haben wir das Projekt Nibahn gemacht. Der Name vom Projekt wurde aus Wörter Nibo und Bahn erstellt, weil es Nibos von PC als eine Kolonne entlang die schwarze Linie steuern lässt. Nibos tauschen auch die Daten miteinander, sie signalisieren optisch welche Rollen sie in der Kolonne nehmen optisch und auch durch die Kommunikation mit andere Nibos über Funk. Sie vermeiden dadurch physische Kollisionen. Nibos können auch individuell gesteuert werden, sie können selbst eigene Rollen in der Kolonne bestimmen. Nibos nutzen bei uns entwickeltes Protokoll, sie erstellen und interpretieren die „XBee Transmission Units“ entsprechend dem von uns definiertem Format. Die Kolonne selbst ist dynamisch und skalierbar und maximaler Anzahl von Nibos hängt von maximaler Datenpacket in der Kommunikation. Dieses Mal waren wir leider von insgesamt nur 8 Bits begrenzt.

2. Pflichtaufgaben

2.1. Anforderungsanalyse

2.1.1. Technische Komponenten

Im Projekt in der Tabelle 1 beschriebene Komponenten genutzt wurden:

ID	Komponent	Name und Parametern	Zweck	Funktionsprinzip
ID_1	Hauptprozessor	ATmega128A, 128kByte Flash, 16MHz	Ausführung des Programms	Elektronisches
ID_2	Co-Prozessor	ATmega88A, 8kByte Flash, 16MHz	Motorsteuerung und Distanzmessung	Elektronisches
ID_3	Distanzsensoren	5 * IR Emitter/Sensoren, Reflexionsverfahren	Messung von Distanzen	Messung von Spiegelung des Infraroten Lichts
ID_4	Aktorik	2 Motoren (a 1,5W) mit 1:25 Übersetzung	Bewegung von Rädern	Elektromagnetische Induktion
ID_5	Odometrie	4 IR Sensoren	Messung von Drehrichtung und Drehzahl	Messung von Spiegelung des Infraroten Lichts
ID_6	LEDs	8 zweifarbige LEDs (rot/grün)	Anzeige von Hindernissen	Messung von Spiegelung des Infraroten Lichts
ID_7	Räder	Gummiräder mit 44 mm Durchmesser	Bewegung von Nibo	Reibungskraft
ID_8	Display	128x64 Pixel LC-Grafikdisplay (monochrom)	Anzeige von Hindernissen	Steuerung von Flüssigkristalle mit Strom
ID_9	Rahmenkonstruktion	Metallbolzen/Leiterplatten-Kombination	Physische Unterstützung von anderen Komponenten	N/A

Tabelle 1. Ausrüstung von Nibo [1]

2.1.2. Anforderungen

Die Anforderungen für die Anwendung unterteilen sich in funktionale Anforderungen, sowie in nicht-funktionale Anforderungen jeweils für die Hardware und für die Software. Um diese Anforderungen zu realisieren, werden den Hauptcontroller und den Co-Controller benutzt.

Funktionale Anforderungen

ID	Name	Beschreibung der Anforderung	Komponenten des Roboters	Kriterium
F_1	Hinderniserkennung	Nibo muss der Hindernisse erkennen	ID_3	muss
F_2	Hindernisumgehung	Nibo muss der Hindernisse umgehen	ID_4, ID_7, ID_5	muss
F_3	Tunnel	Nibo muss den Tunnel verlassen	ID_3, ID_4, ID_7, ID_5	muss
F_4	Optische Darstellung	Das Erkennung von Hindernisse muss mit der Hilfe von LEDs und auf dem Display dargestellt werden	ID_6, ID_8	muss

Tabelle 2. Funktionale Anforderungen(eigene Darstellung)

Nicht funktionale Anforderungen

ID	Name	Beschreibung der Anforderung	Komponenten des Roboters	Kriterium
NF_1	Distanzsensorgrenze	Das Hindernis muss mindestens von 2cm erkennt werden	ID_3	muss
NF_2	Hinderniszeichen	Das Hindernis muss auf dem Display mit der Hilfe vom Achtungszeichen dargestellt werden	ID_8	muss
NF_3	Programmstart	Das Hauptprogramm soll nach dem S3-Knopfdruck starten	S3-Knopf	soll
NF_4	LED-Anzahl	Das Hindernis soll mindestens mit einem LED bezeichnet werden	ID6	soll
NF_5	Lesbarkeit	Funktionen sollten je nach Verwendungszweck in separate Dateien unterteilt werden.		soll

Tabelle 3. Nicht funktionale Anforderungen(eigene Darstellung)

2.1.3. Geplannte Funktionen

Die Anwendung könnte aus folgenden Funktionen bestehen:

1. void start();

Funktion zur visuellen und akustischen Darstellung eines Countdowns.
Bestandteilen des Roboters: ID_3

2. void findDirection();

Funktion zur Führung des Nibo durch Ermittlung der Position von Hindernissen.
Bestandteilen des Roboters: ID_6, ID_8

3. void drive();

Nibo fährt geradeaus.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

4. void turnLeft();

Nibo dreht sich um 90 Grad nach links.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

5. void turnHalfLeft();

Nibo dreht sich um 45 Grad nach links.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

6. void turnRight();

Nibo dreht sich um 90 Grad nach rechts.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

7. void turnHalfRight();

Nibo dreht sich um 45 Grad nach rechts.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

8. void goBack();

Nibo fährt zurück.
Bestandteilen des Roboters: ID_3, ID_4, ID_5, ID_7.

9. void LEDShow(char color, int start, int end);

Funktion zum Anzeigen der Position eines Hindernisses in Bezug auf Nibo mit LEDs:

- color Farbe der LED (r-rot, g-grün);
- start Anzahl der Start-LEDs;
- end Anzahl der End-LED.

Bestandteilen des Roboters: ID_6.

10. void HindranceShow(int x, int y);

Funktion zum Anzeigen der Position eines Hindernisses in Bezug auf Nibo auf dem Display. Zeichnet ein Dreieck mit einem Ausrufezeichen.

- x Abszisse Startpunkt;
- y Ordinate Startpunkt.

Bestandteilen des Roboters: ID_8.

11. void showNibo();

Funktion zum Anzeigen von Nibo auf dem Display.

Bestandteilen des Roboters: ID_8.

2.2. Konzeptentwicklung

Am Anfang der Entwicklung wurde den gesamten Programmablauf und Nibo's Bewegung graphisch dargestellt. Die Anforderungen und das Prinzip des Autofahrens wurden zugrunde gelegt. Die letzte Version des Programmablaufs ist auf der Abbildung 1 gezeigt.

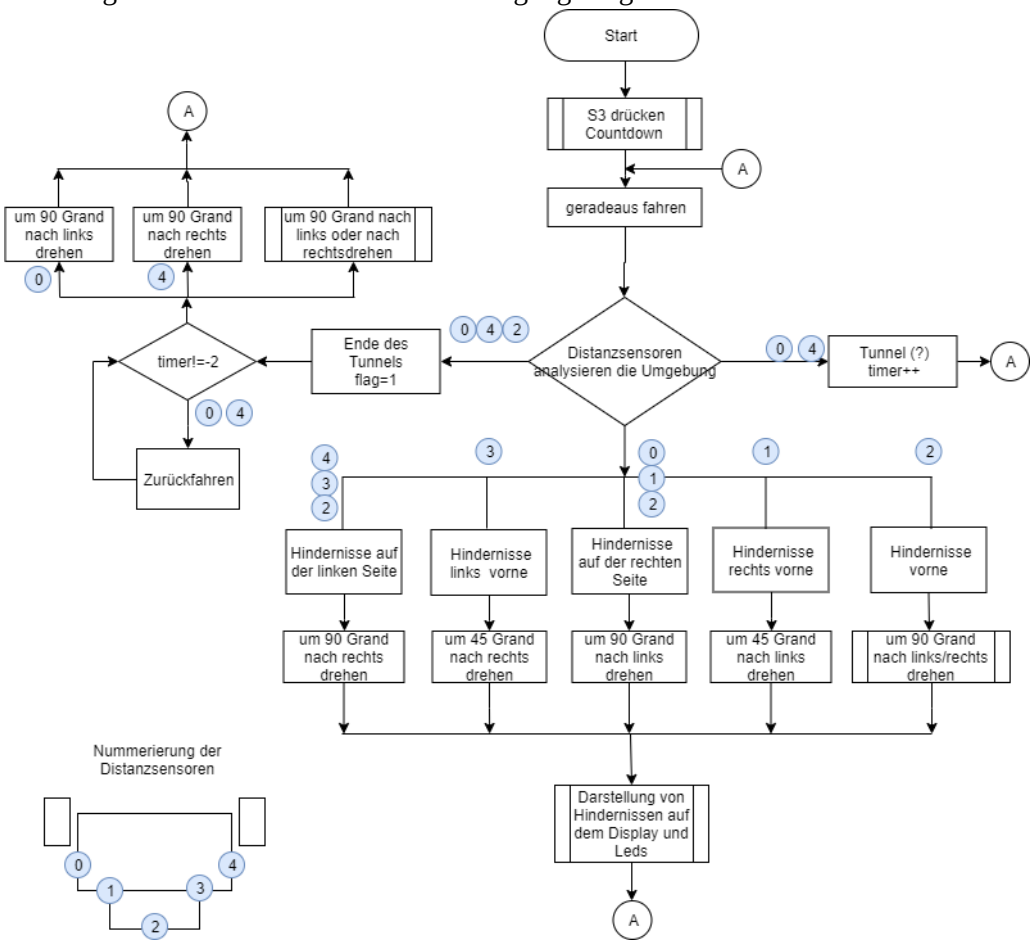

Abbildung 1. Programmablauf der Pflichtaufgaben (eigene Darstellung)

Aus der Abbildung 2 können 8 Zustände von Nibo bestimmt werden. Auf der Abbildung 2 sind die Zustände und die Bedingung für den Übergang von einem Zustand in einen anderen dargestellt:

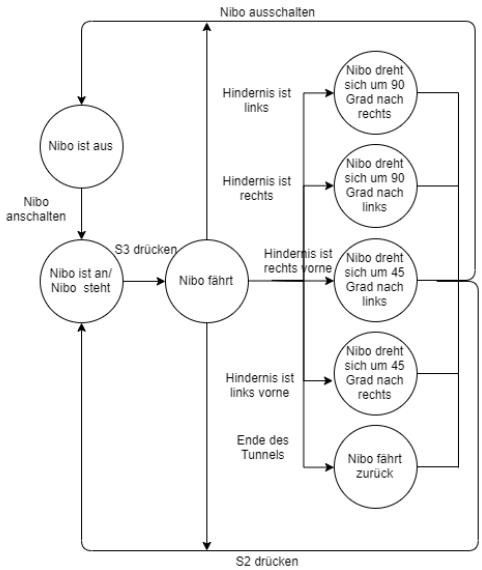

Abbildung 2. Zustandsgraph der Pflichtaufgaben (eigene Darstellung)

Die Bauteile von Nibo kommunizieren miteinander wie auf der Abbildung 3 (ohne den Countdown). Der Hauptcontroller verwaltet die LEDs zur Statusanzeige, die Scheinwerfer-LEDs und die Linien- und Bodensensoren und das Display. An den Co-Controller sind die Sensoren zur Hinderniserkennung, sowie die Motorsteuerung angeschlossen. Über eine SPI-Schnittstelle initiiert er jede Millisekunde eine Übertragung zum Hauptcontroller. [2]

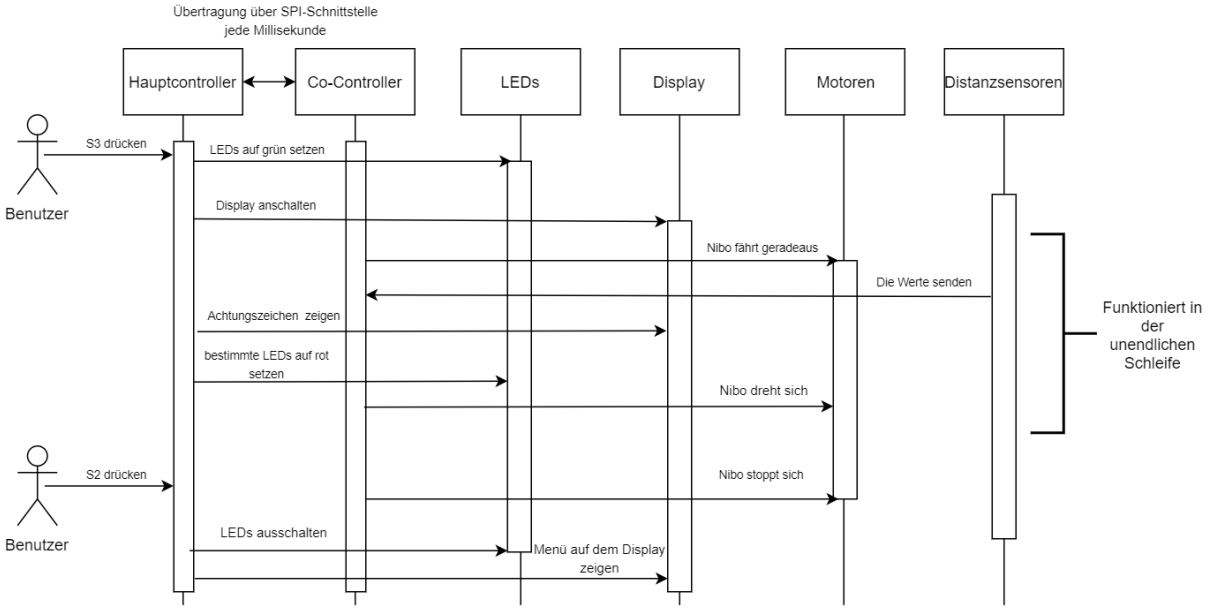

Abbildung 3. Sequenzdiagramm der Pflichtaufgaben (eigene Darstellung)

Beim Einschalten des Roboters wird dem Benutzer ein Menü wie auf der Abbildung 4 angezeigt.

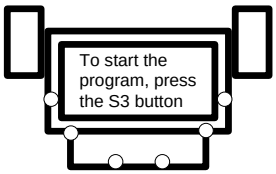

Abbildung 4. Willkommensmenü(eigene Darstellung)

Wenn man die S3-Taste drückt, startet der Countdown. Als Beispiel wurde der Countdown in Computerspielen (Rennen) genommen. Das Bild zeigt die Countdown-Sequenz. Das Blinken der Lampen wird von einem Ton begleitet (rote LEDs - leiser Ton, grüne LEDs - hoher Ton).

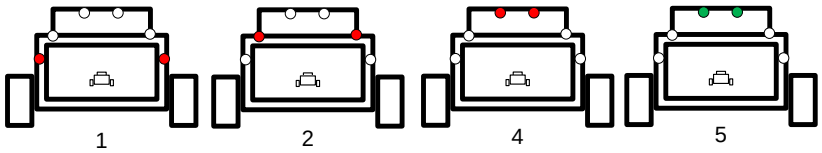

Abbildung 5. Countdown (eigene Darstellung)

Wenn der Countdown abgelaufen ist, leuchten alle Lampen grün und Nibo erscheint auf dem Display. Nibo fangt an, geradeaus mit bestimmter Geschwindigkeit zu fahren.

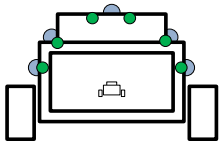

Abbildung 6. Nibo nach dem Drücken der S3 Taste (eigene Darstellung)

In der unendlichen Schleife messen die 5 Distanzsensoren die Distanz. Abhängig von den Sensorwerten und der Sensornummer dreht sich der Roboter. Das Prinzip ist im Sequenzdiagramm dargestellt. Auf der Abbildung 7 ist ein Bespiel der Darstellung von einem Hindernis präsentiert.

Wenn z. B. der Distanzsensor ein Hindernis erkennt, wird der entsprechende LED auf rot gesetzt. Außerdem wird auf dem Display ein Achtungszeichen dargestellt.

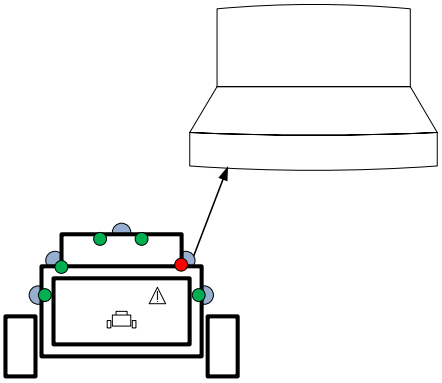

Abbildung 7. Hinderniserkennung (eigene Darstellung)

Dieses Achtungszeichen wird laut der Abbildung 8 und der Tabelle 4 gemallt.

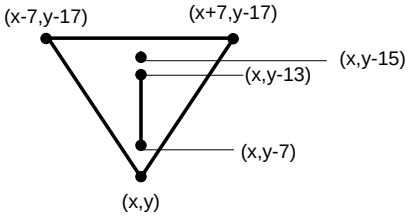

Abbildung 8. Symbolbildprinzip (eigene Darstellung)

№	Die Nummer des Sensors, der das Hindernis erkennt	x	y
1	0	10	30
2	1	40	50
3	2	65	50
4	3	85	50
5	4	90	30

Tabelle 4. Koordinaten des Zeichnungsbeginns eines Achtungszeichens(eigene Darstellung)

Der Roboter erkennt den Tunnel dank der Sensoren 0 und 4. Wenn er im Tunnel geradeaus fährt, erhöht er den Zähler. Am Ende des Tunnels (Sensoren 0, 2, 4) setzt er die Flagge und schaltet die Rückleuchten ein. Nibo fährt so lange zurück, bis der Zähler 0 wird und die rechten und linken Wände enden. Abhängig von den Sensorwerten (siehe Sequenzdiagramm) verlässt Nibo dann den Tunnel und senkt die Flagge.

Die Auswahl eines der Datenstrukturen wirkt sich auf die Größe des reservierten Speichers und die Größe der dargestellten Werte aus. Im Programm gibt es keine Notwendigkeit die Datenstrukturen wie z.B. Array, Structure und Union zu verwenden. Deswegen werden die Datentypen wie Short, Char, Int benutzt.

2.3. Umsetzung

2.3.1. Probleme und deren Lösungen

Am Anfang fanden alle Funktionen in main.c statt. Das Ändern des Programms war schwierig, da eine große Menge des Codes die Lesbarkeit erschwerte und zu Fehlern führte.
Da das Programm viele Funktionen enthalten muss, wurden mehrere Source-Dateien erstellt. In der Tabelle 5 sind die Dateien beschrieben. Das Ergebnis der Trennung kann als bessere Lesbarkeit und einfachere Unterstützung für das Programm bezeichnet werden.

№	Name	Beschreibung
1	Main.c	Hauptdatei. Enthält unendliche Schleife.
2	Main.h	Enthält grundlegende Typedefs, globale Makros und alle Includes, die man für den Nibo2 braucht.
3	Distance.c	Funktionen zum Auffinden von Hindernissen und zum Finden eines freien Pfades. Funktionen lesen Informationen von den Abstandssensoren des Nibo2
4	Drive.c	Funktionen für Nibos Bewegung (Fahren und Wenden)
5	Start.c	Funktionen mit dem Countdown vor dem Start des Hauptprogramms.
6	Presentation.c	Funktionen zur Visualisierung der Position eines Hindernisses in Bezug auf Nibo

Tabelle 5. Dateibeschreibung(eigene Darstellung)

Ein weiteres Problem bestand darin, dass die Variablen für die Flagge und den Zähler ständig aktualisiert wurden. Da das Programm in der unendlichen Schleife läuft, werden die statischen Variablen für die Flagge und den Zähler verwendet. Der Compiler speichert ihre Werte von einem Funktionsaufruf zu einem anderen. Deswegen werden die Werte der Variablen nicht gelöscht.
Das größte Problem ist die Besonderheit von Nibo, dass er geradeaus ohne die Abweichung nicht fahren kann. Leider kann man dieses Problem selbst nicht lösen, denn es besteht in der Funktionsweise und dem Aufbau des Motors.

2.3.2. Evaluation der Anwendung

Die Tabelle 6 vergleicht die Programmversionen und deren Übereinstimmung mit den Anforderungen. Die letzte Version der Anwendung entspricht allen Anforderungen:

Anforderungs-ID \ Versionsnummer	F_1	F_2	F_3	F_4	NF_1	NF_2	NF_3	NF_4	NF_5
0.1	+			+	+			+	
0.2	+			+	+	+		+	
1.0	+			+	+	+		+	+
1.1	+	+		+	+	+		+	+
2.0	+	+	+	+	+	+		+	+
2.1	+	+	+	+	+	+	+	+	+

Tabelle 6. Programmversionen (eigene Darstellung)

3. Küraufgaben

3.1. Anforderunganalyse

3.1.1. Funktionale Anforderungen.

ID	Akteur	Anforderung	Ergebnis	Priorität
F_1	Der Roboter	Muss eigenes ID bestimmen können	Roboter kann eigene Rolle bestimmen	Muss
F_2	Der Roboter	Muss schwarze Linie folgen	Robotern können derselben Weg folgen	Muss
F_4	Der Roboter	Muss Information mit anderen Nibos austauschen können	Nibos sind kommunikationsfähig	Muss
F_5	Der Benutzer	Muss Nibos steuern können	Ein Operator kann die Arbeit von Nibos steuern	Muss
F_6	Der Roboter	Muss mithilfe von LEDs eigene Rolle zeigen	Rolle von Nibo ist für Benutzer sichtbar	Muss

Tabelle 7 Funktionale Anforderungen (eigene Darstellung)

3.1.2. Nicht funktionale Anforderungen

ID	Akteur	Anforderung	Ergebnis	Priorität
NF_1	Der Roboter	Muss niemals wegen Logik des Programms nicht bewegen können	Roboter ist immer bereit für die Bewegung	Muss
NF_2	Der Roboter	Muss für die Kommunikation bestimmte Protokolle und TU Struktur nutzen	Robotern können die übertragenen Daten interpretieren	Muss
NF_3	LEDs	Können für den Mensch faszinierend arbeiten	Indikation mit Hilfe von LEDs ist Benutzer freundlich	Kann

Tabelle 8 Nicht funktionale Anforderungen (eigene Darstellung)

Prioritäten:
Kann - wünschenswert, aber nicht obligatorisch.
Muss - sind den erfolgreichen und zufriedenstellenden Betrieb des Systems notwendig und müssen erfüllt werden.

3.1.3. Technische Komponenten

Im Projekt in der Tabelle 9 beschriebene Komponenten genutzt wurden:

ID	Komponent	Name und Parametern	Aufgabe	Funktionsweise	Qualität der Daten
ID_1	Hauptprozessor	ATmega128A, 128kByte Flash, 16MHz	Ausführung des Programms	Elektronisches	N/A
ID_2	Co-Prozessor	ATmega88A, 8kByte Flash, 16MHz	Motorsteuerung	Elektronisches	N/A

Fortsetzung der Tabelle					
ID_3	Aktorik	2 Motoren (a 1,5W) mit 1:25 Übersetzung	Bewegung von Rädern	Elektromagnetische Induktion	N/A
ID_4	Odometrie	4 IR Sensoren	Messung von Drehrichtung und Drehzahl	Messung von Spiegelung des Infraroten Lichtes	Gleich auf beiden Seiten sein soll
ID_5	Linienfolgesensoren	2 IR Reflexlichtschranken	Messung von Helligkeit der Oberfläche	Messung von Spiegelung des Infraroten Lichtes	Muss schwarze Linie auf dem Boden erkennen lassen
ID_6	LEDs	4 zweifarbige LEDs (rot/grün)	Rollenanzeige	Messung von Spiegelung des Infraroten Lichtes	N/A
ID_7	Räder	Gummiräder mit 44 mm Durchmesser	Bewegung von Nibo	Reibungskraft	N/A
ID_8	NBX2-Modulus	IR-Empfänger für Datenübertragung bzw. Fernsteuerungen	Empfang und Sendung von den Daten	Funkkommunikation	N/A

Tabelle 9 Ausrüstung von Nibo *[1]*

3.1.4. Geplannte Funktionen

Die Anwendung besteht aus 2 Teile, Funktionen von Nibos die mit Xbee von PC und andere Nibos gesteuert werden und Funktionen, die Nibo realisiert ohne Kontrolle von PC oder anderen Nibos.
Funktionen von Nibo die werden mit Hilfe von Xbee gesteuert:

1. Speedup()

Nibo fährt vorwärts, wenn es früher ruhig gewesen.
Wenn Nibo früher negative Geschwindigkeit hätte, stoppt es.
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8

2. Slowdown()

Nibo fährt rückwärts, wenn es früher ruhig gewesen.
Wenn Nibo früher positive Geschwindigkeit hätte, stoppt es.
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8

3. Directionchange_left();

Nibos können 180 links biegen und dadurch die Richtung wechseln
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8

4. Directionchange_right();

Nibos können 180 rechts biegen und dadurch die Richtung wechseln
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8

5. void Turn_right();
Nibos können 90 Grad rechts biegen
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8

6. void Turn_left();
Nibos können 90 Grad links biegen

Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8
7. Reelection_turn(byte ownId, short reelection_delay, byte maxnibos);

Nibos starten Bestimmung von Rollen in der Kolonne erneut, wenn sie die Richtung wechseln. je großer das ID von Nibo war, desto schneller wird dieser Nibo im Prozess Teilnehmen und eigenes ID holen.
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8
8. Reelection_forced(byte ownId, short reelection_delay, byte maxnibos);

Nibos starten Bestimmung von Rollen in der Kolonne erneut, wenn sie ein Befehl von PC kriegen. je großer das ID von Nibo war, desto später wird dieser Nibo im Prozess Teilnehmen und eigenes ID holen.
Bestandteilen des Roboters genutzt: ID_2, ID_3, ID_4, ID_7, ID_8
Funktionen von Nibo die werden auf Nibo gesteuert:

9. buildSendData(u8_t reciver, u8_t sender, u8_t data);
Baut und sendet ein Byte entsprechend vordefiniertem Format
Bestandteilen des Roboters genutzt: ID_1, ID_8

10. sendDataGetReciver(u8_t sendData);
Gibt zurück Zieladresse aus empfangenen Daten
Bestandteilen des Roboters genutzt: ID_1, ID_8

11. sendDataGetSender(u8_t sendData);
Gibt zurück Zieladresse aus empfangenen Daten
Bestandteilen des Roboters genutzt: ID_1, ID_8

12. sendDataGetData(u8_t sendData);
Gibt zurück Zieladresse aus empfangenen Daten
Bestandteilen des Roboters genutzt: ID_1, ID_8

13. sendDataGetFunctionSet(u8_t sendData);
Gibt zurück Zieladresse aus empfangenen Daten
Bestandteilen des Roboters genutzt: ID_1, ID_8

14. sendDataGetFunction(u8_t sendData);
Gibt zurück Zieladresse aus empfangenen Daten
Bestandteilen des Roboters genutzt: ID_1, ID_8

15. Schwarze_Linie_Folgen(byte ignore_line);
Nibo folgt Schwarze Linie auf dem Boden, wenn variable ignore_line !=1 ist.
Und sendet auch stop signal an anderen Nibos mit großeres ID als eigenes ID von Nibo.
Bestandteilen des Roboters genutzt: ID_1, ID_8, ID_7

16. Status_LEDs(byte ownId);
Zeigt mithilfe von LEDs die Rolle von Nibo in der Kolonne
Bestandteilen des Roboters genutzt: ID_2, ID_6

3.2. Konzept

3.2.1. Allgemeines

Nibos werden in Eine Kolonne fahren, die Kolonne wird sich an Schwarze Linie unten orientieren und sich automatisch korrigieren, falls Irgendeinen Nibo fährt raus die schwarze Linie.

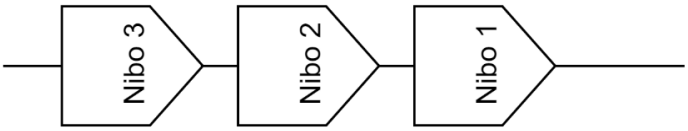

Abbildung 9 die Kolonne (eigene Darstellung)

Nibos werden wir steuern mit Hilfe von Xbee Moduls auf Nibos und mit PC-verbundenen Xbee Moduls. Hier Moduls auf PC ist ein Operator der Kolonne.

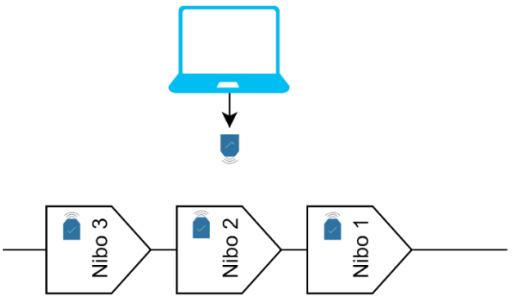

Abbildung 10 Kommunikation PC - Nibos (eigene Darstellung)

Nibos werden selbst die Rolle Bestimmen und entsprechend dieser Rolle LEDs einschalten
Master Nibo wird LEDs mit oranger Farbe haben andere werden LEDs mit grüner Farbe haben. So wird ein Operator sehen welches Nibo Zurzeit ein Master ist.

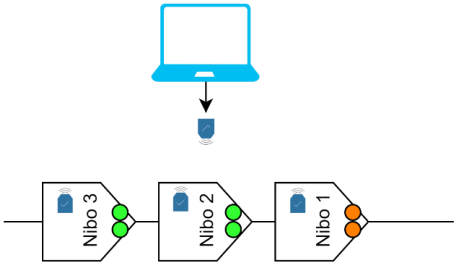

Abbildung 11 Darstellung von die Rollen in der Kolonne (eigene Darstellung)

Die Kolonne und Nibos dran werden wir mit festem Set von Befehlen für Broadcast und Steuerung von einzelnes Nibo steuern.

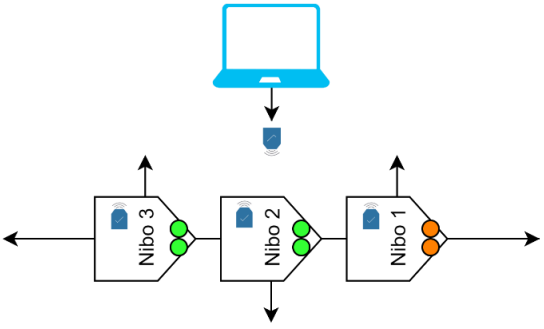

Abbildung 12 Darstellung von der Steuerung der Nibos in der Kolonne (eigene Darstellung)

3.2.2. Zustandsgraph von Nibo

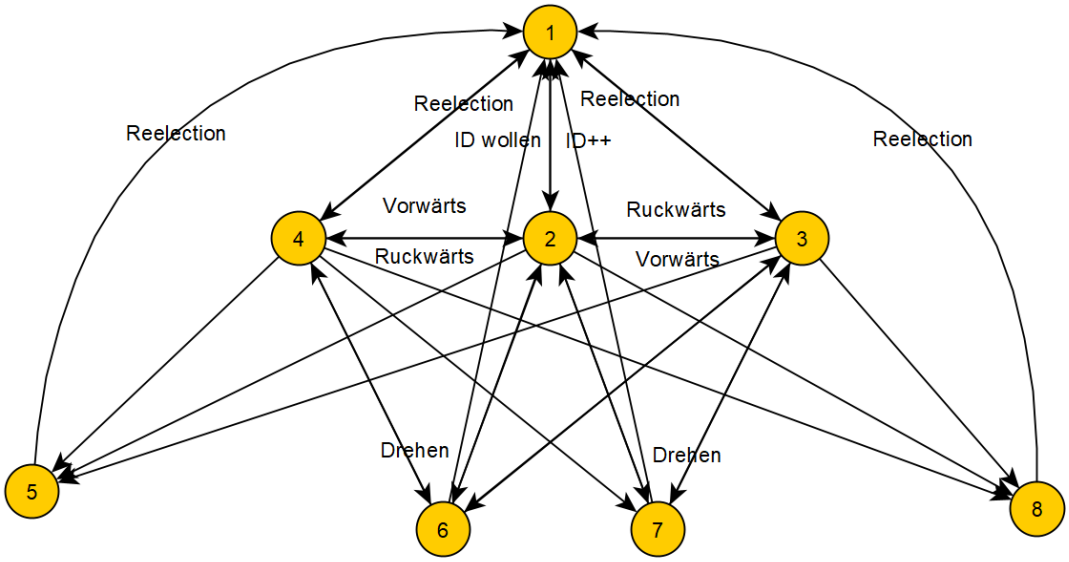

Abbildung 13 Zustandsgraph der Nibahn (eigene Darstellung)

1. ID nicht abgestimmt
2. Ruhig
3. Bewegt Ruckwärts
4. Bewegt Vorwärts
5. Wechselt die Richtung (Rechts)
6. Dreht Links
7. Dreht Rechts
8. Wechselt die Richtung (Rechts)

3.2.3. Darstellung von verwendenen Algorithmen und Datenstrukturen

Um die Steuerung von Nibos mit Hilfe von XBee zu ermöglichen haben wir eigenes Format von Transmission Units für unseres System definiert. Als Basis für unseres TU haben wir Byte genommen, dieser Aufgeteilt und für alle Bits eigene Bedeutung definiert. (Abbildung 14)

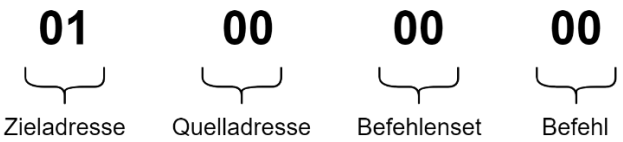

Abbildung 14 Format von Transmission Units (eigene Darstellung)

Mit Hilfe von Zieladresse kann richtiges Nibo reagieren, Quelladresse lässt Bestimmen von wem kommt der Befehl und mit Hilfe von Befehlensets können wir die Befehle voneinander Isolieren.
Komplexe Datenstrukturen waren für unseres Programm nicht benötigt, aber wir haben Anzahl von benötigtem Speicherplatz zu minimieren durch Nutzung von passender primitiver Variablen.

3.2.4. Sequenzdiagramm von Nibo

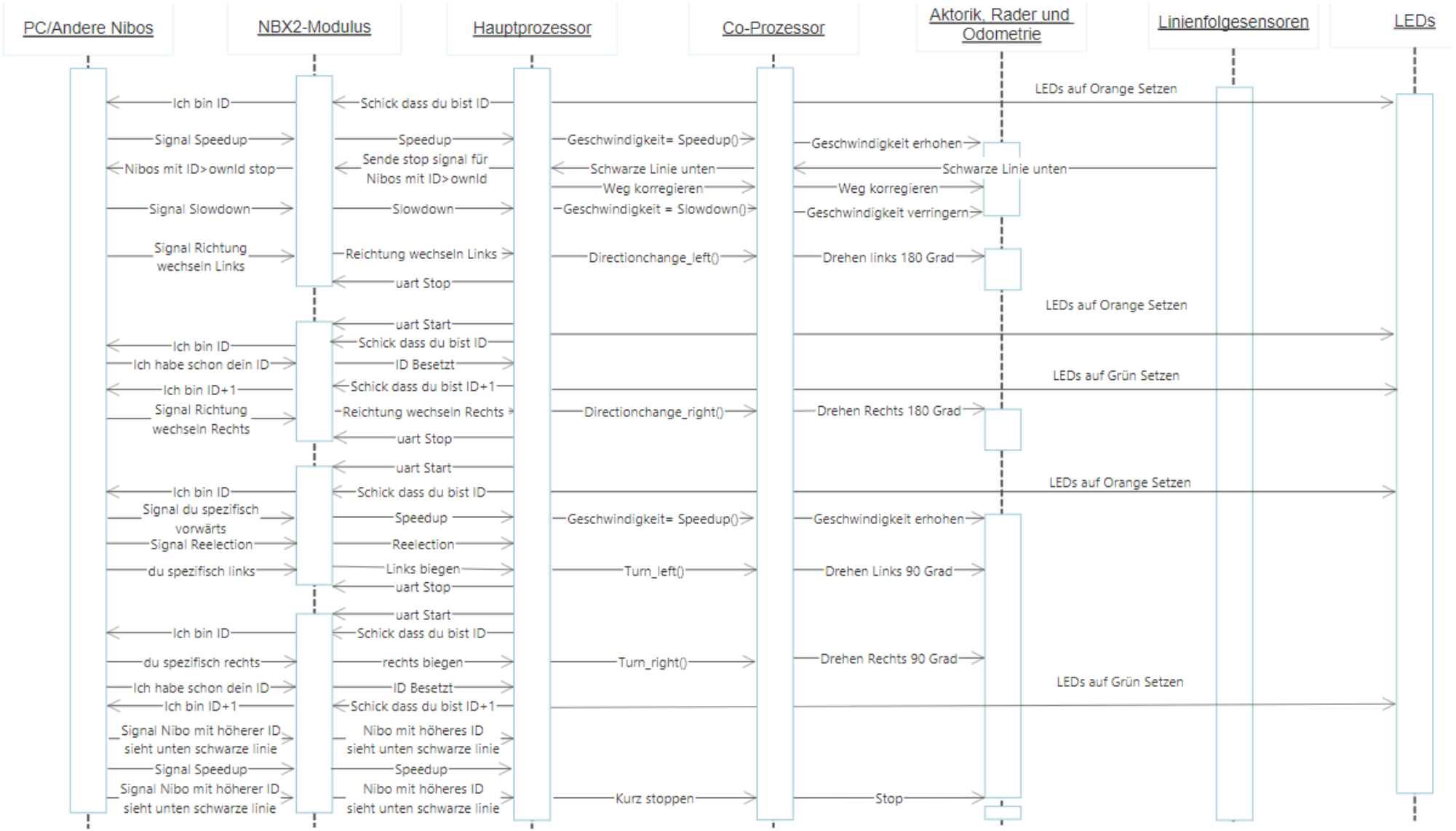

Abbildung 15 Sequenzdiagramm der Kommunikation (eigene Darstellung)

3.3. Umsetzung

Wir haben einige Problemen während der Entwicklung getroffen und haben diese priorisiert als offene Fragen:

Version	Progress von geplanten Funktionen	Merkmale	Offene fragen
01.10.2019	5%	Befehlen Set 0 war realisiert	Testing von Befehlenset 0 Format für Datenübertragung noch nicht implementiert.
08.10.2019	15%	Neues Format für Datenübertragung implementiert wurde	Befehlen 0, 1 in Befehlen Set 0 können zwischen Zustand haben.
15.10.2019	30%	Nibos Können die Schwarze Linie Folgen, Befehlen 0,1 haben zwischenzustand	Keine
22.10.2019	40%	Befehlen Set 1 ist realisiert	Testing von Befehlen Set 1
29.11.2019	55%	Erstellung von externen Dateien und Ersetzung von Code mit Funktionen	Die Rollen in der Kolonne sind nicht dynamisch
05.11.2019	70%	Einführung von Kommunikation Nibo-Nibo (Prozess von Masterauswahl ist dynamisch)	Keine
17.11.2019	85%	Befehlen Set 2 ist realisiert	Testing von Befehlen Set 2, Protokoll ist nicht skalierbar, reicht nur für 2 Nibos
24.11.2019	95%	Protokoll ist skalierbar, Rolle in der Kolonne wird mit Hilfe von LEDs angezeigt	Rolle wechseln bei der Wechselung der Richtung
28.11.2019	100%	Testing und Bugfixes	Passieren Kollisionen, wenn nibos zu oft stoppen
30.11.2019	105%	Nibos kommunizieren Miteinander und schicken Stoppsignal an alle nibos hinten sich, wenn sie auf Schwarze Linie stehen	Keine

Tabelle 10 Evaluation der Nibahn-Anwendung (eigene Darstellung)

Ergebnisse von Testen sind in der Tabelle 11 zusammengefasst

Objekt	Version	Resultat	Merkmale
Befehlen Set 0	01.10.2019	Funktioniert	Kein Format
Zwischenstand Befehlen 0 und 1	08.10.2019	Funktioniert wie geplant	Keine
Befehlen Set 1	15.10.2019	Funktioniert wie geplant	Keine
Dynamische Rolle in der Kolonne	05.12.2019	Funktioniert	Nicht skalierbar
Befehlen Set 2	17.11.2019	Funktioniert	
Gemeinsames Projekt	28.11.2019	Funktioniert wie geplant	ist der Antwort

Tabelle 11 Ergebnisse von Testen der Nibahn-Anwendung (eigene Darstellung)

Wichtigste Problem, das wir in unserer Arbeit getroffen haben, war die Unskalierbarkeit von unserem System. IDs von Nibos waren festgelegt und es war nicht möglich diese dynamisch zu wechseln mehr IDs zu haben und die Nummerierung von Nibos steuern, deshalb wurde das Verfahren implementiert der Sorgt um die Nummerierung. Das Sequenzidaigramm für dieses Verfahren steht in der Abbildung 16.

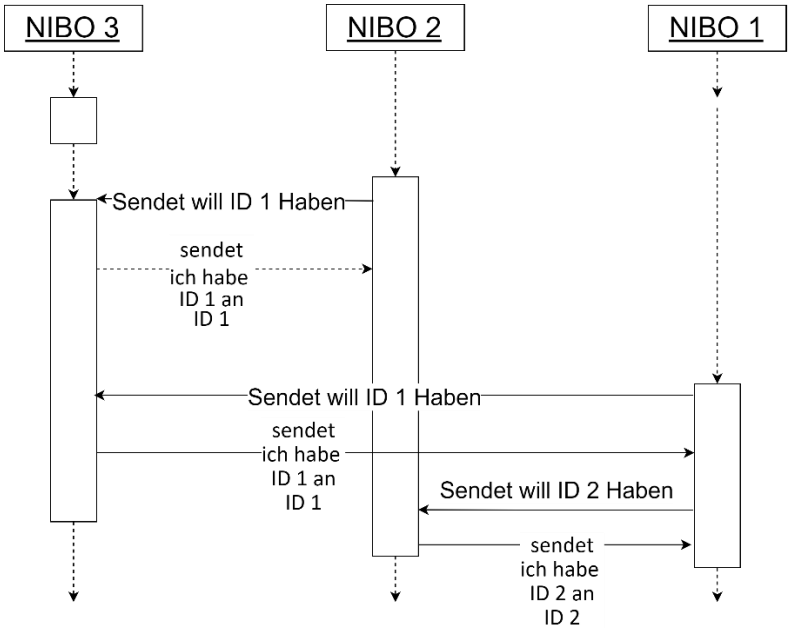

Abbildung 16 Sequenzdiagramm für die Nummerierung (eigene Darstellung)

3.4. Fazit

Bei der Ausführung der Aufgaben wurden die auf den „ESuR“ Vorlesungen erworbenen Kenntnisse praktisch umgesetzt. Besonders hilfsreich war das Prinzip von der Verteilung der Funktionen je nach Verwendungszweck in separaten Dateien. Dieses Prinzip erheblich erleichtert das Schreiben von Code, die Lesbarkeit sowie die Weiterentwicklung und Hinzufügung neuer Funktionen. Für unseres Projekt hat die Arbeit mit Bits sich wirklich ausgezahlt, weil wir uns sehr oft mit Bitoperationen und deren Interpretation beschäftigt haben.

Laut der Ergebnisse wurden alle Anforderungen der Pflicht- und Küraufgaben erfüllt. Es muss bemerkt sein, dass die Entwicklung des Programms für die Umgehung der Hindernisse, besonders Tunnel, bringt interessante Herausforderungen mit sich. Als Hauptvoraussetzung kann man die Zeit, während der Nibo den Tunnel verlassen, nennen. Hier sind die Kenntnisse nichtur in der C-Programmierung, sondern auch in den Algorithmen notwendig. Mit der optischen Darstellung wurde Kreativität eingesetzt. Es gibt eine Reihe von Möglichkeiten, wie man das ermöglicht kann.

Küraufgabe gibt echt viel Platz für Fantasie und Recherche, wegen freier Themen bietet diese Aufgabe die Möglichkeit etwas Eigenes zu verfassen aber stellt gleichzeitig auch eine Herausforderung für Einschätzung von eigenen Fähigkeiten und Kenntnisse. Besonders interessant war die Entwicklung von Anwendungen mit XBee Module, weil wir eigenes einfaches Protokoll entwickeln konnten.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] nicai-systems, „Nicai-Systems,“ 2016. [Online]. Available: http://www.nicai-systems.com/de/nibo2. [Zugriff am 02 12 2019].

[2] Prof. Dr. Janett Mohnke, Daniel Wittekind, Wegweiser zur Roboter-Programmierung, TH Wildau.