„ServiceRoboter (SR1)

* Patientendialogsystem (PDS) -“

Autor: Dobbelgarten

QS-Freigabe:

# Überblick

## Zweck

Das vorliegende Dokument dient dazu, eine genaue Vorstellung von dem zu entwickelnden Produkt zu entwickeln, in dem verschiedene Aspekte (Funktionen, Daten, Dynamik, Benutzungsschnittstelle, Infrastruktur) des Produkts modelliert werden. Das daraus entstehende Produktmodell ist die fachliche Problemlösung (Fachkonzept), in dem genau festgehalten ist, was das Produkt kann und wie es sich dem Benutzer darstellt.

## Inhaltsverzeichnis

[0 Überblick 2](#_Toc530233473)

[0.1 Zweck 2](#_Toc530233474)

[0.2 Inhaltsverzeichnis 3](#_Toc530233475)

[1 Infrastruktur (Hardware) 4](#_Toc530233476)

[2 Use Case – Analyse 5](#_Toc530233477)

[2.1 Use Case Diagramm 5](#_Toc530233478)

[2.2 Use Case Beschreibungen 5](#_Toc530233479)

[3 Benutzerschnittstellen 5](#_Toc530233480)

[3.1 Zum Patienten 5](#_Toc530233481)

[3.1.1 Über Terminal 5](#_Toc530233482)

[3.1.2 Über SCRS 5](#_Toc530233483)

[3.2 Zum Administrator 5](#_Toc530233484)

[4 Weitere Schnittstellen 5](#_Toc530233485)

[4.1 Zum Roboter 5](#_Toc530233486)

[4.2 Intern zum SCRS 6](#_Toc530233487)

[5 Datenbank-Entwurf (Entity-Relationship-Diagramm) 6](#_Toc530233488)

[6 SW-Architektur 6](#_Toc530233489)

[6.1 Klassendiagramm 6](#_Toc530233490)

[6.2 Sequenzdiagram 6](#_Toc530233491)

[7 Anhang 7](#_Toc530233492)

[7.1 Spezifikation des SCRS-Protokolls auf der MQTT-Schnittstelle 7](#_Toc530233493)

[7.1.1 Empfangsnachrichten 7](#_Toc530233494)

[7.1.2 Sendenachrichten 8](#_Toc530233495)

[7.1.3 Fehler-Codes 9](#_Toc530233496)

[7.2 Hinweise zum Roboter-Protokoll auf der ROS-Schnittstelle 10](#_Toc530233497)

## Abbildungsverzeichnis

[Abbildung 1: PDS-Hardwareumgebung 5](#_Toc531799482)

[Abbildung 2: Use Case Diagramm 6](#_Toc531799483)

[Abbildung 3: Entity-Relationship zwischen KIS und GIS 11](#_Toc531799484)

[Abbildung 4: Beispiel des Zusammenhangs KIS/GIS 11](#_Toc531799485)

[Abbildung 5: Klassendiagramm des PDS 11](#_Toc531799486)

[Abbildung 6: Sequenzdiagram des PDS 12](#_Toc531799487)

[Abbildung 7: Schnittstellen des "Navigation Stacke" des Roboters 16](#_Toc531799488)

# Infrastruktur (Hardware)

Für das PDS wird die bestehende Hardware-Umgebung des aktuellen Roboters ergänzt um einen an der THGA entwickelten Smart Connected RFID-Sensor. Hardware-technisch erfordert dieser jedoch keine Integration.

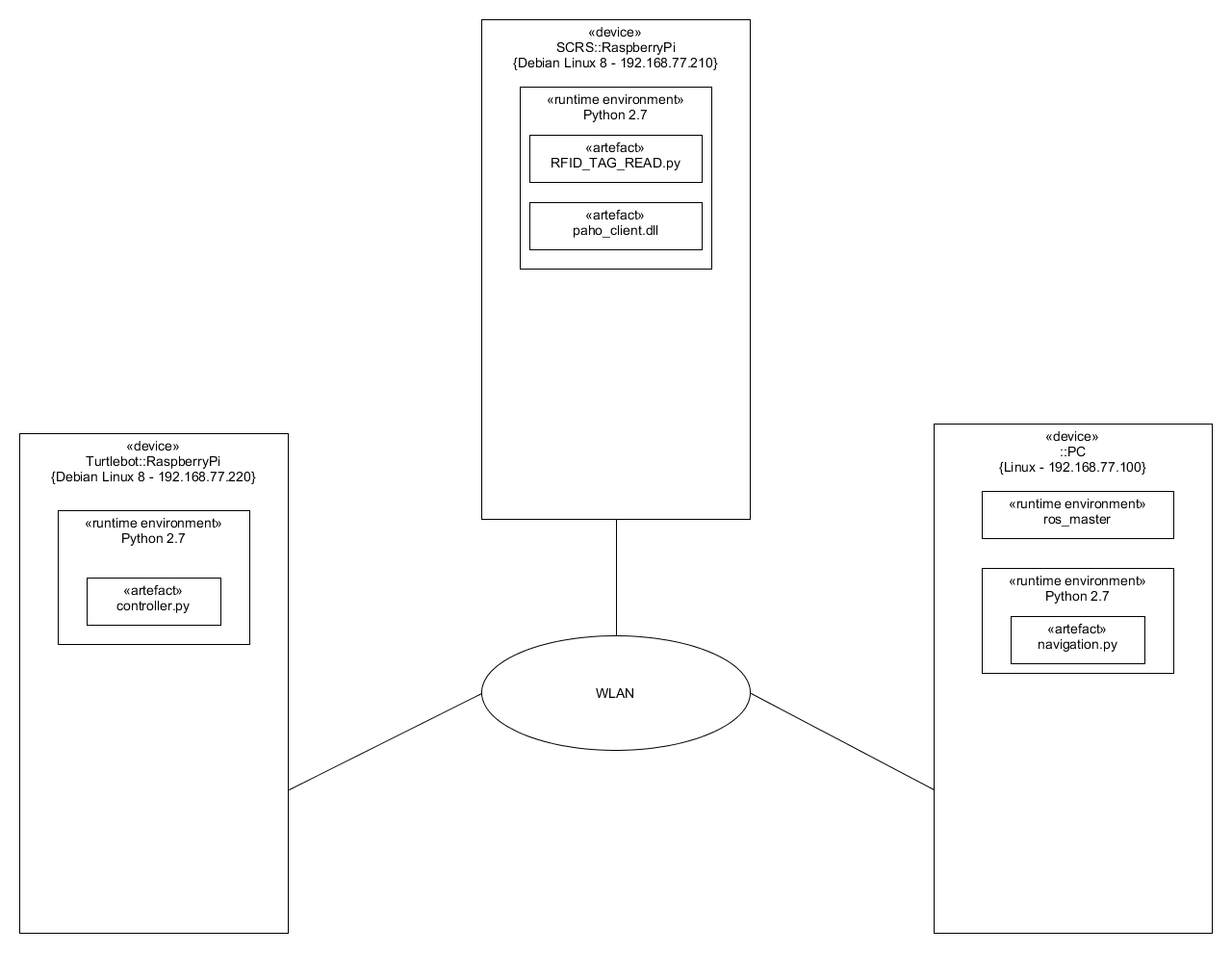


Abbildung 1: PDS-Hardwareumgebung

# Use Case – Analyse

## Use Case Diagramm

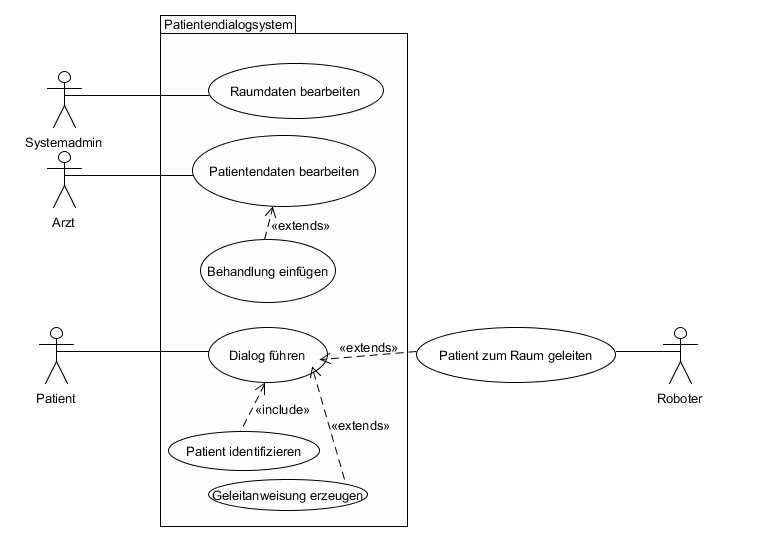


Abbildung 2: Use Case Diagramm

## Use Case Beschreibungen

**Use Case Dialog führen: Standardszenario**

**Vorbedingungen**:

1. Ein Patientenarmband verfügbar dessen RFID Nummer im PDS, mit einem Patienten verknüpft ist.
2. Für den Patienten stehen Untersuchungen an, z.b.
   1. Blutabnahme
   2. EKG
   3. Röntgen
3. Es gibt eine Standard-Raumliste die der Patient auswählen und dort hingeleitet werden kann u.a. „Naechstes WC“.
4. Der Service Roboter befindet sich im Wartezustand.
5. Das PDS zeigt „Oberfläche I“ an.

**Ablauf**

1. Der Patient hält das Patientenarmband an den SCRS des Service Roboters.
2. Das PDS sucht in der Patientendatenbank nach dem Patienten und findet dort anstehende Behandlungen.
3. Das PDS zeigt die „Oberfläche II“ mit dem Namen des Patienten, den hinterlegten Untersuchungen und den Standardzielen an.
4. Der Patient wählt ein aufgelistetes Ziel aus.
5. Das PDS sendet die Koordinaten des ausgewählten Ziels an den Roboter.
6. Der Roboter setzt sich in Bewegung.
7. Das PDS zeigt „Oberfläche III“ an.
8. Der Roboter erreicht das ausgewählte Ziel.
9. Das PDS zeigt „Oberfläche IV“ an.
10. Das PDS zeigt nach 5 Sekunden „Oberfläche I“ an.

**Alternative Szenarien:**

* Der Patient ist nicht in der Patientendatenbank vorhanden
  + „Oberfläche II“ wird angezeigt, jedoch ohne Begrüßung und nur mit Standardzielen.
* Der RFID ist defekt.
  + Der Reader kann einen Chip erkennen, kann diesen nicht lesen.(S.o.)
* Der Patient trifft bei „Oberfläche II“ keine Auswahl.
  + Das PDS zeigt nach 60 Sekunden „Oberfläche I“ an.
* Der Patient wählt „abbrechen“.
  + Das PDS zeigt „Oberfläche I“ an.
* Der Roboter kann den Raum nicht erreichen.
  + Das PDS zeigt „Oberfläche V“ an.
  + Das PDS zeigt nach 30 Sekunden „Oberfläche I“ an.
* Der Patient macht eine Fehleingabe auf „Oberfläche II“
  + Das PDS gibt „Oberfläche II“ erneut aus.

# Benutzerschnittstellen

## Zum Patienten

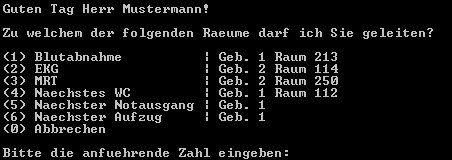
### Über Terminal

**Oberflächen des PDS:**

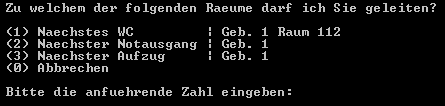
**Oberfläche I:**



**Oberfläche II:**

****

**Alternative Oberfläche II:**



**Oberfläche III:**



**Oberfläche IV:**



**Oberfläche V:**



### Über SCRS

Der SCRS kann RFID-Transponder des Typs ISO 15693 lesen und schreiben. Die Schnittstelle zwischen dem Patienten und dem SCRS wird über Patientenarmbänder mit RFID-Tags dieses Typs realisiert. Der SCRS liest im Poll-Modus kontinuierlich sich in Lesereichweite befindliche Transponder. Die Schnittstelle ist als reine Eingangsschnittstelle ausgelegt, über der lediglich die RFID-Transponder-Nummer (UID-Unique Identifier) eingelesen wird.

## Zum Administrator

Der Administrator verwaltet die Daten über eine MySQL Datenbank welche über einen Apache Server gehostet wird. Mit der URL <http://localhost/phpmyadmin> lassen sich die Datenbanken erreichen und komplett Verwalten. Der Apache Server und die Datenbank werden über das Tool „XAMPP“ gestartet.

# Weitere Schnittstellen

## Zum Roboter

Die Schnittstelle zum Roboter wird über ROS-Topics realisert.

1. PDS → Roboter

Über die Schnittstelle werden Zielkoordinaten für den Geleitvorgang übertragen. Mit der Übertragung der Zielkoordinaten startet der Roboter den Geleitvorgang, ohne das ein separater Startimpuls gegeben werden muss.

* ROS-Topic: /move\_base\_simple/goal
* Message-Typ: PoseStamped [geometry\_msgs.msg]
* Message-Inhalt
  + header.frame\_id = ‘map’ (Koordiantenursprung ???)
  + pose.position.x = x-Position des Ziels
  + pose.position.y = y-Position des Ziels
  + pose.position.z = 0
  + pose.orientiation.w = 1
  + alle anderen Felder = 0

1. Roboter → PDS

Über die Schnittstelle werden Statusmeldungen über einen Abbruch oder das Ende des Geleitvorgangs übertragen.

* ROS-Topic: /move\_base/result
* Message-Typ: MoveBaseActionResults [move\_base\_msgs.msg]
* Message-Inhalt
  + falls Geleitvorgang beendet wurde
    - status.status = 3
    - status.text = 'Goal reached'
  + falls Geleitvorgang abgebrochen wurde
    - status.status ≠ 3 (siehe 7.2)
    - status.text ≠ 'Goal reached' (siehe 7.2)

## Intern zum SCRS

Zum Einlesen der RFID-UID muss intern die durch den SCRS vorgegebene Schnittstelle implementiert werden. Die Spezifikation des Protokolls ist im Anhang 7.1 zu finden.

# Datenbank-Entwurf (Entity-Relationship-Diagramm)

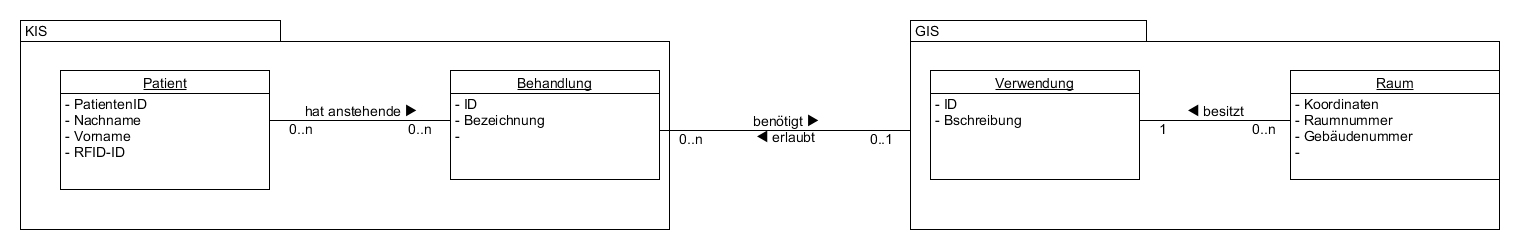


Abbildung 3: Entity-Relationship zwischen KIS und GIS

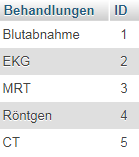
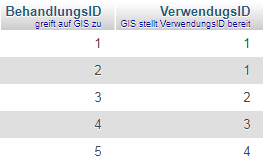
 

Abbildung 4: Beispiel des Zusammenhangs (KIS/GIS)

# SW-Architektur

## Klassendiagramm

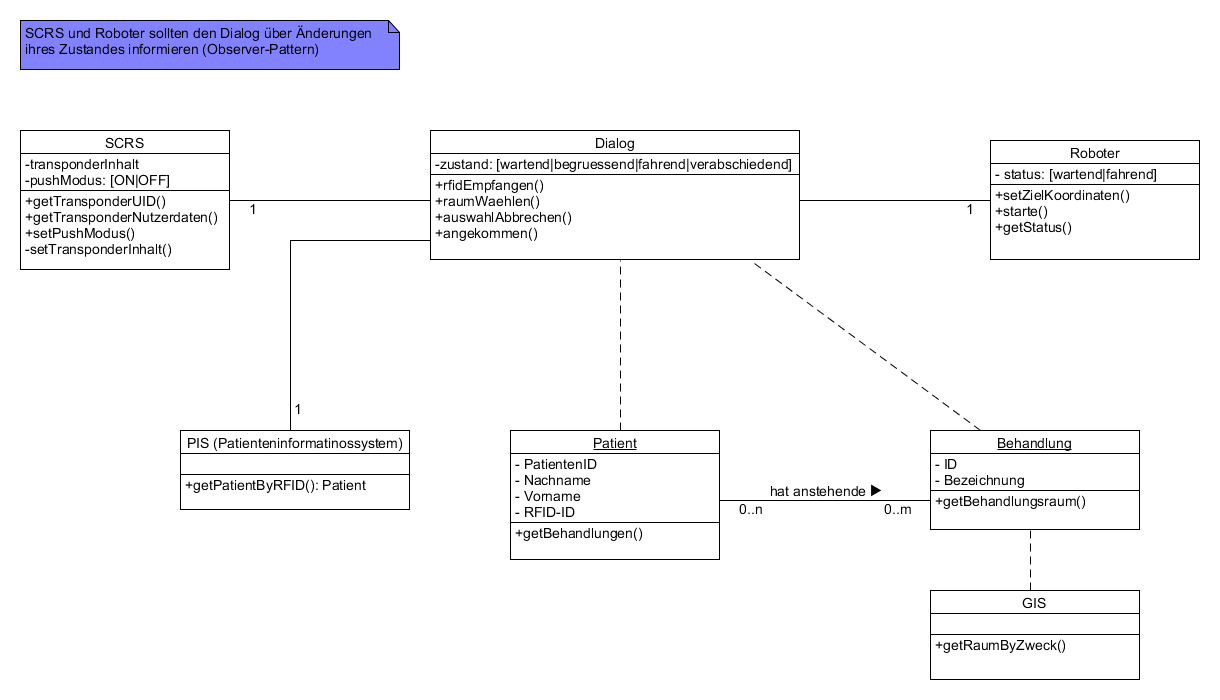


Abbildung 5: Klassendiagramm des PDS

## Sequenzdiagram

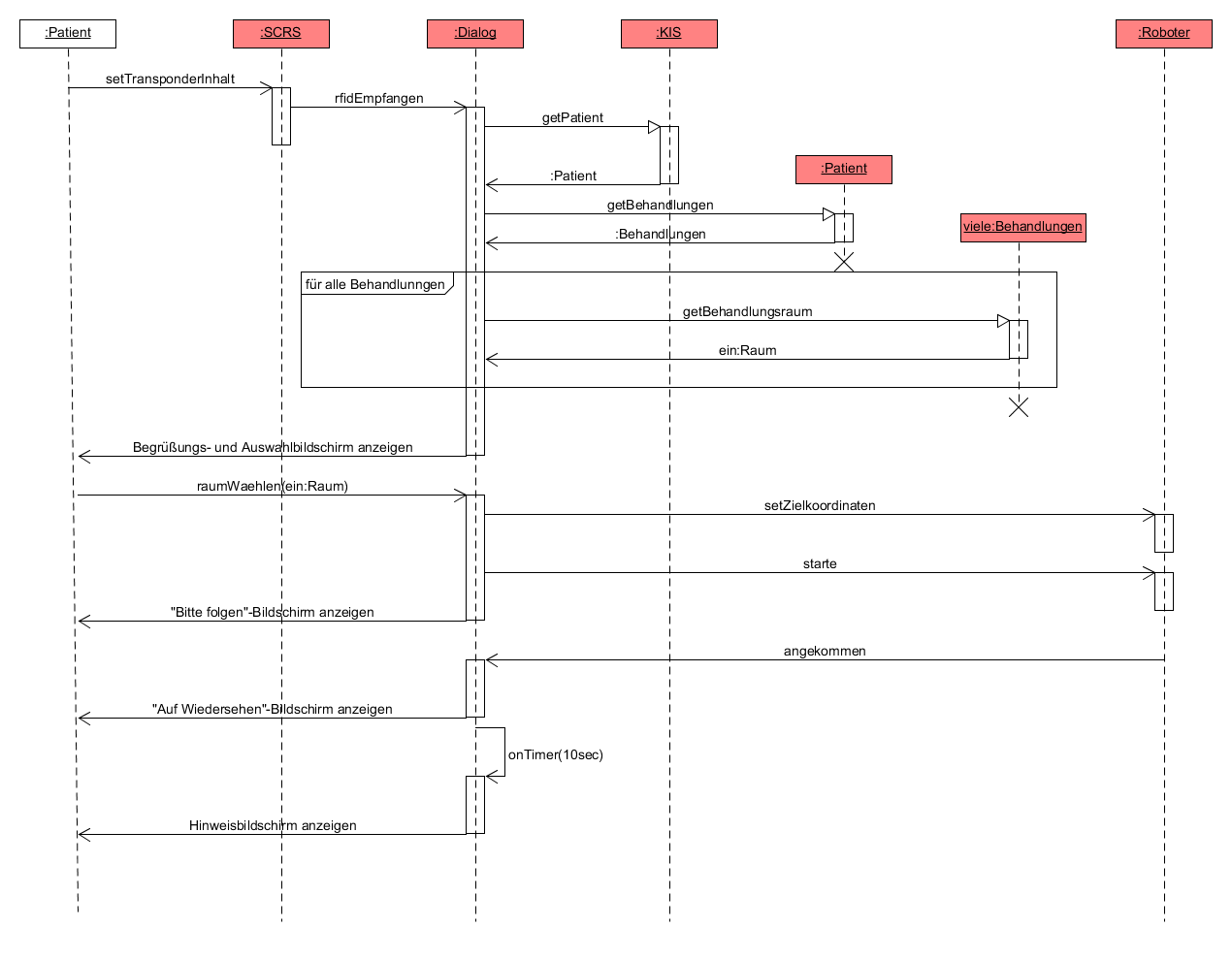


Abbildung 6: Sequenzdiagram des PDS

# Anhang

## Spezifikation des SCRS-Protokolls auf der MQTT-Schnittstelle

### Empfangsnachrichten

Nachrichten werden über Topics der Form

\*/<SENSORNAME>/COMMAND

empfangen.

Mit den Empfangsnachrichten können Kommandos an den SCRS gesendet werden. Die Nachrichten sind:

#### Umschalten Push/Poll-Modus

Puon

[4 Zeichen, Befehl]

Der SCRS wird in den Push-Modus geschaltet. Sobald sich ein Tag im Lesebereich des Readers befindet, wird dessen UID und Inhalt mit einer Tagdaten-Nachricht im 10 sec- Takt versendet. Falls sich kein Tag im Lesebereich des Readers befindet, werden keine Tagdaten-Nachrichten versendet. read- und write-Nachrichten werden nicht verarbeitet.

puoff

[5 Zeichen, Befehl]

Der SCRS wird in den Poll-Modus geschaltet. Das automatische Versenden von Tagdaten-Nachrichten wird beendet. read- und write-Nachrichten werden verarbeitet.

#### Lese aktuellen Tag (read)

td

[2 Zeichen, Befehl]

Der SCRS antwortet mit einer Tagdaten-Nachricht (siehe unten) mit der UID und dem Inhalt des aktuellen Tags. Falls sich kein Tag im Lesebereich des Readers befindet, ist das ID-Feld der Tagdaten-Nachricht mit 0en gefüllt.

#### Schreibe aktuellen Tag

wa<DATEN>

[226 Zeichen, Befehl + hex[string]]

Der SCRS beschreibt den aktuellen Tag ab dem 0ten Block mit dem in der MQTT Nachricht übergebenenInhalt. Der Inhalt wird als Hexdezimalstring kodiert und umfasst 224 Zeichen. Die UID des Tags wird nicht benötigt, der Payload der MQTT Nachricht sieht wie folgt aus:

wa00000000055550000000044440008888000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000EEEE00000000000000000000000000000000000000000000000000000000000000000

Als Bestätigung des Schreibvorganges wird eine td Nachricht mit den aktuellen Daten des Tags versendet.

wi<UID><DATEN>

[242 Zeichen, Befehl + hex[string]]

Der SCRS beschreibt den Tag mit der übergebenen UID ab dem 0ten Block mit dem in der MQTT Nachricht übergebenen Inhalt. Der Inhalt wird als Hexdezimalstring kodiert und umfasst 224 Zeichen.

Die ersten 16 Zeichen hinter dem Befehlswort „wi“ enthalten die als Hexadezimalstring kodierte UID des zu schreibenden TAGs.

Beispiel:

wiE00401009E77C01E0000000005555000000004444000888800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000EE00000000000000000000000000000000000000000000000000000

Als Bestätigung des Schreibvorganges wird eine td Nachricht mit den aktuellen Daten des Tags versendet.

### Sendenachrichten

Nachrichten werden über Topics

<SENSORNAME>/TAGDATEN

<SENSORNAME>/WARTUNG

<SENSORNAME>/FEHLER

gesendet.

#### Tagdaten des aktuellen Tag

td<ID><Inhalt der Blöcke 0 bis 27>

[242 Zeichen, Befehl + hex[String]]

Die Nachricht liefert die Tag-ID und den Inhalt des sich aktuell im Lesebereich befindlichen Tags. Wird die Nachricht im Poll-Modus versendet und es befindet sich aktuell kein Tag im Lesebereich wird sowohl das ID- als auch des Inhalts-Feld der Nachricht mit 0en gefüllt. Sowohl die ID als auch der Inhalt der Blöcke wird in einem hexadezimalen String aus 242 Zeichen nach folgendem Beispiel übertragen:

tdE00401009E77C01E00000000000011100000000000EE000000000000000000000000000000000000000000000000FF00000000000000000000000000000000000000000000000000000000000000000000000000000000004444888844448888444488884444888844448888444488880000000000000000

#### Wartungsinformation

mt<#Lesevorgänge><#Schreibvoränge>

[22 Zeichen, Befehl + hex[string]]

Diese Nachricht wird automatisch versendet, sobald die Anzahl der Lese- oder Schreibvorgänge die in der Konfigurationsdatei angegebenen Werte überschritten haben. In den Feldern „#Lesevorgänge“ und „#Schreibvorgänge“ wird die Gesamtzahl aller protokollierten Lese- bzw. Schreibvorgänge übertragen. Die beiden Werte werden in einem 16 Zeichen langen hexadezimalen String übertragen, der 2 jeweils 4 Byte große Ganzzahlen kodiert (Datentyp unsigned long).

Siehe Beispiel:

mt000000000B0000000005

### Fehler-Codes

ec<FEHLERCODE>

[6 Zeichen, Befehl + hex[string]]

Tritt ein Fehler aus der Fehlercode Tabelle auf, wird eine 4 Stellige ec Nachricht versendet.

Beispiel:

ec1001

Fehlercode Tabelle

|  |  |
| --- | --- |
| ec1000 | Tag Lesefehler  [Wenn der Tag während des Lesens entfernt wird] |
| ec1001 | Tag Schreibfehler  [Wenn der Tag während des Schreibens entfernt wird] |
| ec1002 | Nachrichtenlänge fehlerhaft (wi)  [Nachrichtenlänge entspricht nicht dem wi Nachrichtenstandart ] |
| ec1003 | Nachrichtenlänge fehlerhaft (wa)  [Nachrichtenlänge entspricht nicht dem wi Nachrichtenstandart ] |
| ec1004 | Kommando nicht gefunden  [gesendetes Kommando existiert nicht] |
| ec1005 | config Datei konnte nicht gelesen werden  [config Datei fehlerhaft] |

## Hinweise zum Roboter-Protokoll auf der ROS-Schnittstelle



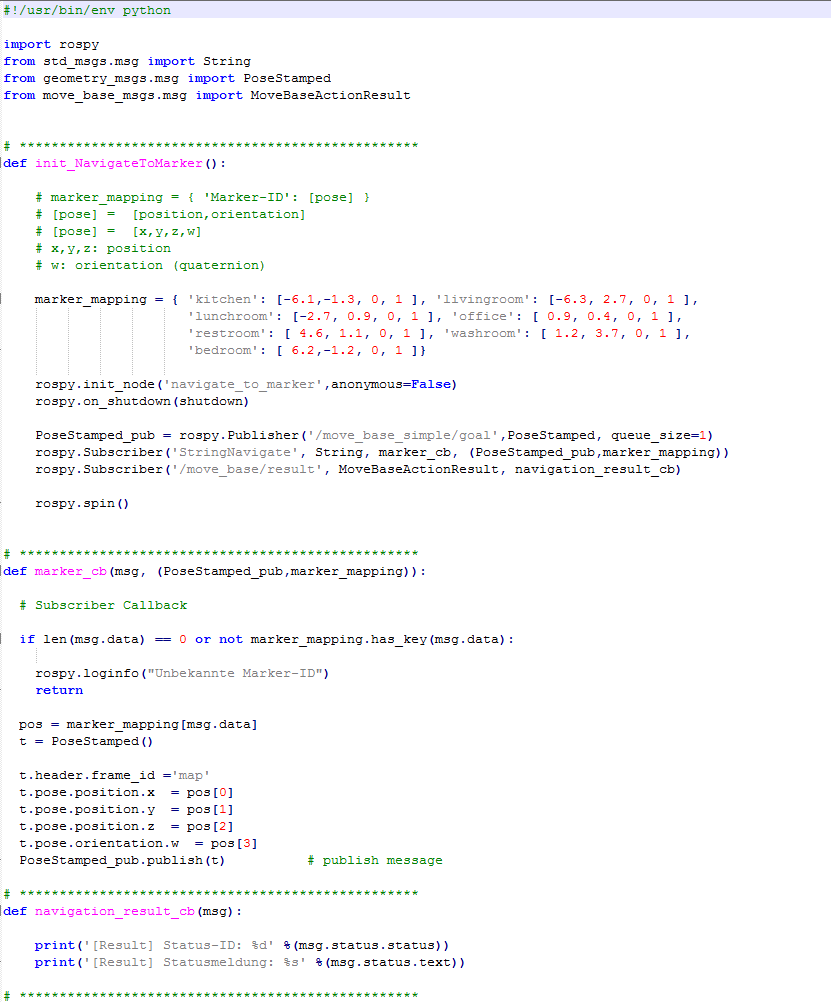
Abbildung 7: Schnittstellen des "Navigation Stacke" des Roboters

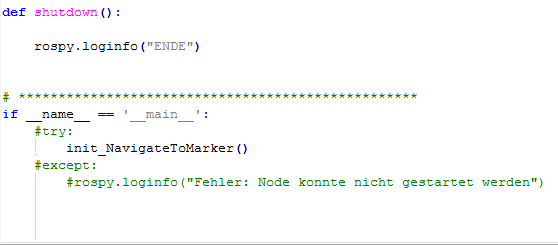
**Auswertung der Statusmeldung:**

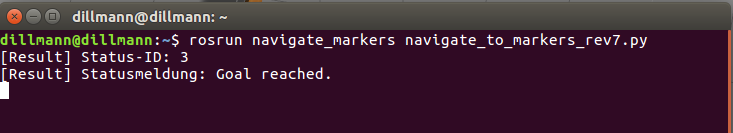
* Ausgabe von insgesamt 10 verschiedenen Statusmeldungen
* Jede Statusmeldung ist mit einer eindeutigen Status-ID verknüpft (0 bis 9 (uint8))
* Zu jeder Status-ID kann eine zugehörige Textnachricht (Ausgabestring) ausgeben werden
* Die Statusmeldungen werden automatisch, beim Eintreten des jeweiligen Ereignisses, unter dem Topic „/move\_base/result“ veröffentlicht

**Beispiel zur Auswertung der Statusmeldung:**

* entsprechende Quellcode-Stellen sind durch rote Pfeile gekennzeichnet







* **Gesamtübersicht: Status-ID**

|  |  |
| --- | --- |
| Status-ID (uint8) | Status |
| 0 | The goal has yet to be processed by the action server |
| 1 | The goal is currently being processed by the action server |
| 2[[1]](#footnote-1) | The goal received a cancel request after it started executing and has since completed its execution (Terminal State) |
| 3[[2]](#footnote-2) | The goal was achieved successfully by the action server (Terminal State) |
| 4[[3]](#footnote-3) | The goal was aborted during execution by the action server due to some failure (Terminal State) |
| 5 | The goal was rejected by the action server without being processed, because the goal was unattainable or invalid (Terminal State) |
| 6 | The goal received a cancel request after it started executing and has not yet completed execution |
| 7 | The goal received a cancel request before it started executing, but the action server has not yet confirmed that the goal is canceled |
| 8 | The goal received a cancel request before it started executing and was successfully cancelled (Terminal State) |
| 9 | An action client can determine that a goal is LOST. This should not be sent over the wire by an action server |

Quelle: http://docs.ros.org/groovy/api/actionlib\_msgs/html/msg/Go

1. Beispiel: Vor dem Erreichen der Zielposition erfolgt die Übermittlung aktualisierter/neuer Zielkoordinaten [↑](#footnote-ref-1)
2. Beispiel: Navigation zum Zielpunkt erfolgreich abgeschlossen [↑](#footnote-ref-2)
3. Beispiel: Keine Pfadplanung zum Zielpunkt möglich (z.B.: Hindernisse verhindern das Erreichen des Zielpunktes) [↑](#footnote-ref-3)