

# Softwareprojekt IngInfo: Projekt MarsRover

## Dokumentation von Team1218

### Gliederung

1. Grundkonzept
2. GUI Anwenderdokumentation
3. Zustandsdiagramm
4. ABNF - Kommunikationsprotokoll
5. Mechanischer und elektrischer Aufbau

## 1 Grundkonzept

Unser Grundkonzept ist eine Kombination der vorgestellten Konzepte der Selbstlokalisierung und der Odometrie in der Vorlesung.

- **Relative Positionsbestimmung durch Odometrie (mit stochastischer Ungenauigkeit):**

Startpunkt ist (0,0) in unserem X,Y- Koordinatensystem.

Die Position des Roboters wird durch erhaltene Befehle (drive\_goto) berechnet und die Position des Roboters wird in unserem Koordinatensystem angepasst.

- **Absolute Positionsbestimmung durch Sensorik, z.B. Erkennung von Landmarken (ebenfalls mit stochastischer Ungenauigkeit):**

Am Roboter sind zwei Pings vorne und hinten auf Servos angebracht, welche sich jeweils um 180 Grad von links nach rechts für die Messungen drehen können. Dadurch erhält man Abstandsmesswerte für die komplette Umgebung, also 360 Grad (mit einem kleinen toten Winkel aufgrund des Abstands zwischen den Pings) des Roboters. Mögliche Hindernisse in der Umgebung des Roboters können so erfasst werden.

Die Steuerung des Roboters erfolgt mit C über die SimpleIDE. Der C-Teil des Projekts ist so knapp wie möglich gehalten, alle Berechnungen und Auswertungen werden auf den Java Script Teil ausgelagert (siehe kommentierter Quellcode).

### **Möglichkeiten der Fernsteuerung:**

- Lasse den Roboter zum Punkt (X,Y) des Koordinatensystems fahren
- Der Roboter kann sich um X Grad drehen
- Der Roboter kann die zwei PING-Sensoren um jeweils 180 Grad von links nach rechts (mit Hilfe der Servos) drehen und dabei alle 3 Grad Abstandsmessungen vornehmen
- Der Roboter kann in den „Laby-Mode“™ übergehen, indem er automatisch ein Labyrinth durchfährt und aufhört, wenn er wieder aus dem Labyrinth herausgefahren ist

### **Grad an Autonomie unseres Systems:**

- Der Laby-Mode ist völlig automatisiert, der Roboter braucht dafür keine Anweisungen
- Die Ping Funktion ist ebenfalls soweit automatisiert, dass die Pings alle 3 Grad messen und die Servos sich um 180 Grad drehen
- Die gemessenen Abstandswerte für Hindernisse zeichnen sich automatisch in unsere GUI ein
- Die Roboterposition wird automatisch nach Positionsänderung errechnet und eingezeichnet
- Die Winkel werden bei Positionsänderung angepasst und korrekt angezeigt (mit der Option der manuellen Korrektur)

## 2 GUI Anwenderdokumentation

Der Aufbau der graphischen Benutzeroberfläche sieht folgendermaßen aus:

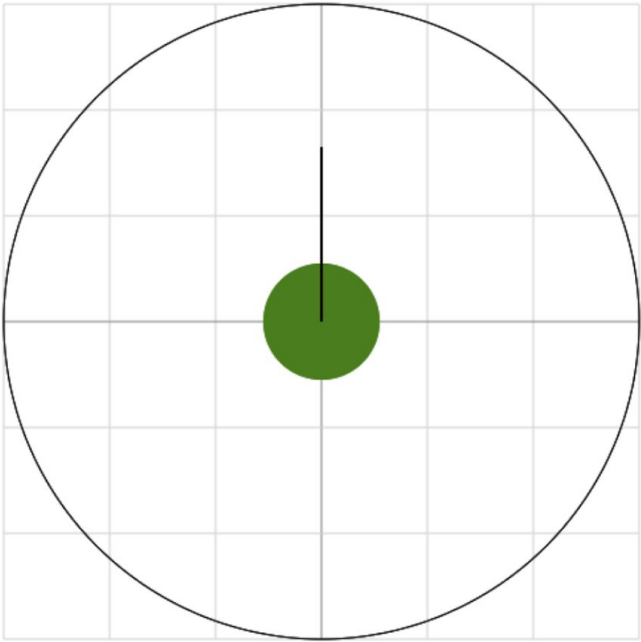
### MarsRover map

draw obstacles	delete spots	delete radius	60	Ping lines: <input checked="" type="checkbox"/>	auto-radar: <input type="checkbox"/>	scale %: 400	movement-lines: <input type="checkbox"/>
move robot (click)	mark spots (click)	drive to marker	request radar sweep	manual map save	go into labyrinth		
rot L	rot R	L	R	UP	DOWN		
90° L turn	90° R turn	45° L turn	45° R turn	180° turn			

1 unit equals 20 cm

---

Roboter: nicht verfügbar  
tot.distance: 0cm, tot.angle: 0°, tot.ticks: 0 (about 0s)  
robot position x:0, y:0, angle:90°



red: forwards left, blue: forwards right, green: backwards left, pink: backwards right

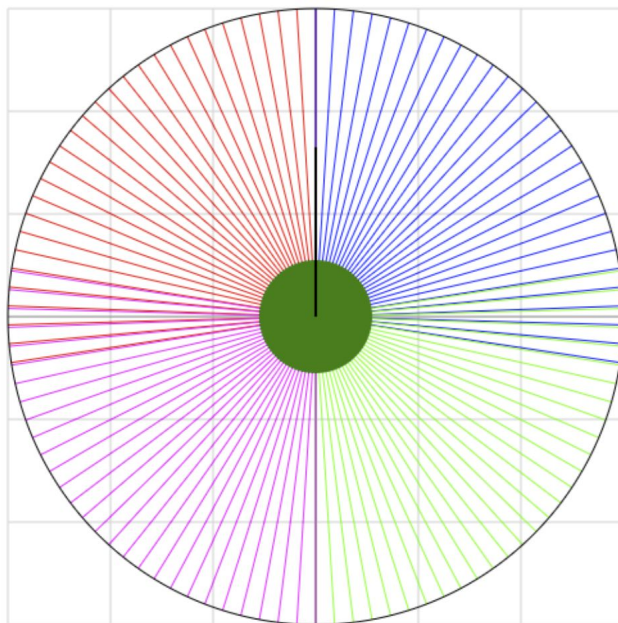
load	reset	status request expected every 3 seconds
------	-------	---

Abbildung 1: Graphische Benutzeroberfläche

Alle Buttons und jede Anzeige sollen in Bezug auf ihr Design möglichst simpel und selbsterklärend sein. Die Buttons, Anzeigen und Freifelder haben meist selbsterklärende Namen und werden nachfolgend dennoch kurz erläutert:

- **drawObstacles:**  
Manuelles Einzeichnen von Hindernissen.
- **deleteSpots:**  
Löscht die manuell gesetzten Markierungen.
- **deleteRadius:**  
Löscht alle Hindernisse in dem Radius des Roboters (Kreis um Roboter).

- **Feld neben deleteRadius:**  
Einstellen wie groß der Radius um den Roboter sein soll.
- **PingLines (check Box):**  
Verknüpft Hindernisse mit Linien, wenn diese nahe genug beieinander liegen.
- **Auto-radar (check Box):**  
Nach jedem Fahrbefehl wird automatisch ein radar sweep, also eine Messung durchgeführt.
- **Scale (Freifeld):**  
Man kann die Skala manuell einstellen.
- **movement-lines:**  
Die möglichen Fahrrichtungen des Roboters werden auf der Karte angezeigt.



red: forwards left, blue: forwards right, green: backwards left, pink: backwards right

Abbildung 2: movement-lines

- **move robot (click):**  
Nach dem Anklicken des Buttons und dem anschließenden Klick auf die gewünschte Stelle im Koordinatensystem bewegt sich der Roboter zu dieser Position.
- **mark spots (click):**  
Nach dem Anklicken des Buttons und dem anschließenden Klick auf die gewünschte Stelle im Koordinatensystem ist dort eine Markierung hinterlegt.
- **drive to marker:**  
Der Roboter fährt zu der vorher ausgewählten markierten Position.
- **request radar sweep:**  
Der Roboter misst den Abstand zu seiner Umgebung (2 Pings, 2 Servos). Dabei ist er nicht verfügbar / erreichbar für andere Anweisungen.
- **Manual map save:**  
Speichert die Karte für spätere Benutzung (als Cookie).

- **Go into labyrinth mode:**  
Der Roboter geht in den patentierten Laby Mode™ über und durchfährt automatisch das Labyrinth. Der Laby Mode endet automatisch nachdem der Roboter erneut aus dem Labyrinth gefahren ist. Der Roboter ist während des Laby Modes noch erreichbar für andere Anweisungen (Laby Mode läuft auf einem eigenen Cog).
- **Die Buttons rot L, rot R, L, R, UP, DOWN** sind alles Buttons für die manuelle Positionsanpassung des Roboters. Dies ist beispielsweise für die Positionsanpassung anhand von Messdaten nützlich (durch markante Landmarken etc.).
- **Die Buttons 90° L turn, 90° R turn, 45° L turn, 45° R turn, 180° turn** sind alles Schaltflächen, mit denen der Winkel des Roboters angepasst werden kann. Er dreht sich anschließend um die angegebenen Winkel und Richtung. Das kann beispielsweise für das Einfahren und Ausfahren des Labyrinths nützlich sein.
- **Freifeld:**

1 unit equals  cm

Abbildung 3: Unit Größe einstellen

Damit können die Abstände des Koordinatensystems manuell eingestellt und angepasst werden

- **Anzeige von Roboterinformationen:**  
Roboter: nicht verfügbar  
tot.distance: 0cm, tot.angle: 0°, tot.ticks: 0 (about 0s)  
robot position x:0, y:0, angle:90°

Abbildung 4: Roboterinformationen

- **Mit den Buttons load und reset** können Karten wieder geladen beziehungsweise der Ausgangszustand wiederhergestellt werden.



Abbildung 5: Load und reset

- Darunter ist noch das Dialogfeld für Statusmeldungen des Roboters mit der passenden Uhrzeit. Dort wird die Position des Roboters und die Verfügbarkeit angezeigt.

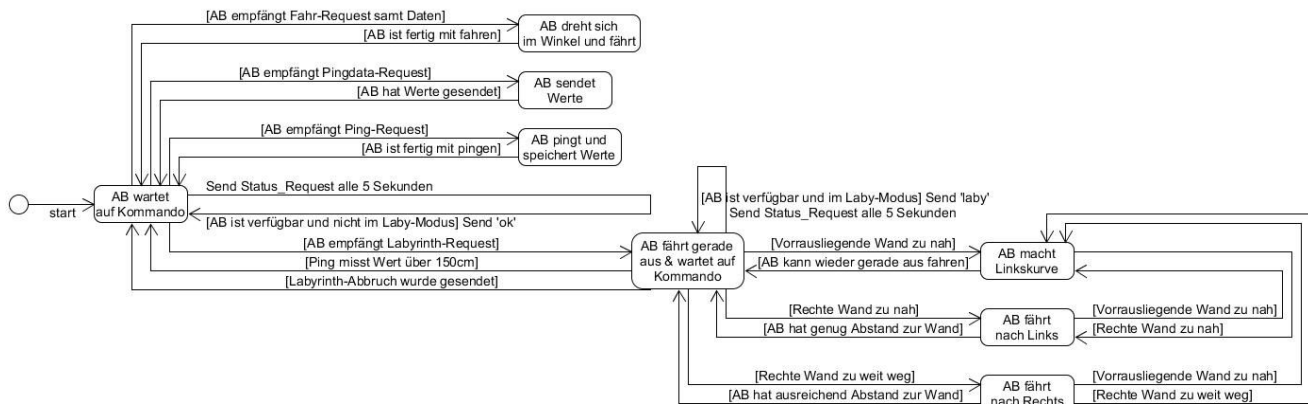
```
11:22:04 - FAILED sending request to /status
---
11:22:07 - FAILED sending request to /status
---
11:22:10 - FAILED sending request to /status
---
11:22:13 - FAILED sending request to /status
---
```

Abbildung 6: Statusmeldungen

Die Darstellung des Softwaredesigns anhand eines Klassendiagramms macht bei unserem Projekt leider wenig Sinn, da der JavaScript Quellcode nur eine Main-Funktion hat. Die

JavaScript Konvention ist eher verschiedene Dateien einzubinden, als mehrere Klassen zu erstellen (aber auch möglich, werden wir zukünftig berücksichtigen).

### 3. Zustandsdiagramm



### 4. ABNF - Kommunikationsprotokoll

```

ziffer = %x30-39 ;digits 0-9
value = [ "-" ] 1*3ziffer ;number from -999 to 999
value_pos = 1*3ziffer ;number from 0 to 999
command_no = value_pos
command = (drive_command | laby_command | ping_command |
pingdata_command | status_command)
answer = (status_answer | ping_answer | pingdata_answer |
cmd_answer)

; MAP -> PHP (command / POST)
request1 = "post=", command_no, ":", command
request2 = "post=number" | "post=clear" ; get current command count
or clear all files

; COMMANDS
drive_command = "cmd@", value, ",", value, ",", value, ",",
,value_pos
;sending left wheel, right wheel, forwards command,
;commandnumber as comma separated values (see POST below)
laby_command = "cmd@X"
ping_command = "ping"
pingdata_command = "pingdata"
status_command = "status"

; MAP -> PHP (responses / GET)
  
```

```

request3 = "connection.php&get=", command_no
status_answer = value_pos,("_us"|"_tus") | "laby" | ">10m"
ping_answer = "ok"
pingdata_answer = value_pos, 29 * ( "_", value_pos ) ;expecting this
command four times and sends one scanned quadrant at the time in
this order: front right, front left, back left, back right
cmd_answer = "done"

; JAVA <-> TXT
line_tobot = command_no, ":", command
line_frombot = command_no, ":", answer

; JAVA -> BOT
send_cmd = "cmd\n" + ("X" | value, ", " ,value, ", " ,value, ", "
,value_pos) ;see above
send_other = (ping_command | pingdata_command | status_command)

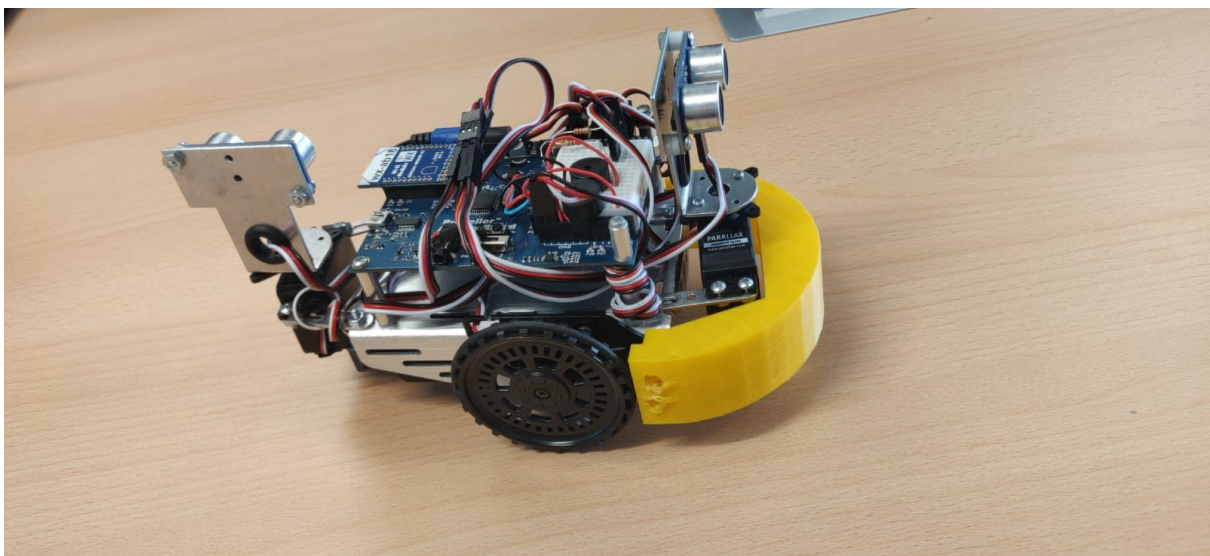
; BOT -> JAVA
recieve = answer

```

## 5. Mechanischer und elektrischer Aufbau

### 5.1 Mechanischer Aufbau

Auf dem ActivityBot sind zwei Servo Motoren vorne und hinten angebracht, welche die zwei Ping Sensoren um 180° von links nach rechts drehen. Außerdem haben wir eine "Stoßstange" mit Hilfe eines 3D Druckers angefertigt um ein mögliches Festklemmen an Hindernissen zu verhindern.





Vorderer Servo Motor: Auf Port 16  
Hinterer Servo Motor: Auf Port 17  
Vorderer Ping: Auf Port 11  
Hinterer Ping: Auf Port 10  
WLAN Modul: Auf Port 7 & 8

## WLAN Modul: Auf Port 7 & 8

The diagram illustrates the wiring for a 4-wire rotary switch assembly. It features a central 40-pin connector with pins labeled P1 through P17. The wiring includes two 5V power sources, two 180-degree rotary switches, two 360-degree rotary switches, and a 4-wire cable with pins labeled CTS, RTS, D1, and D0. The diagram is drawn on graph paper.