



Entwicklung einer Wegpunktnavigation auf der Roboterplattform TurtleBot3

PROJEKTPRÄSENTATION VON

CHIARA TERESA GRUß

JAN-LUCA REGENHARDT

ROBIN SCHEEL

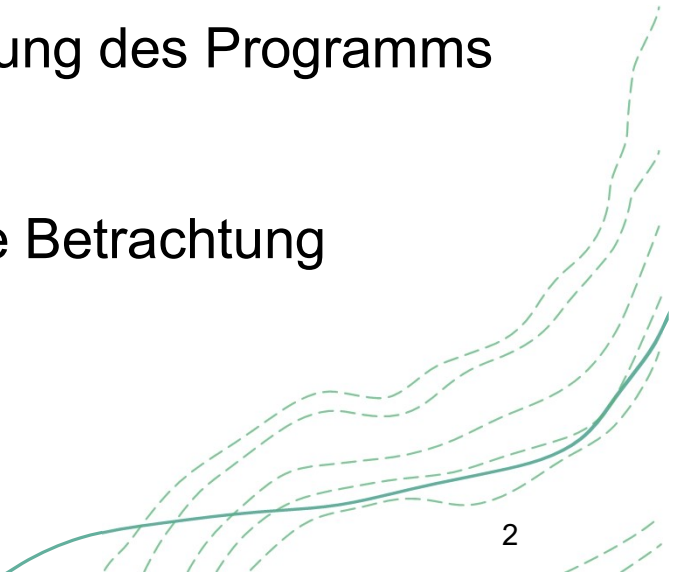
07.01.2021

C.GRUß, J. REGENHARDT, R. SCHEEL

1



Gliederung

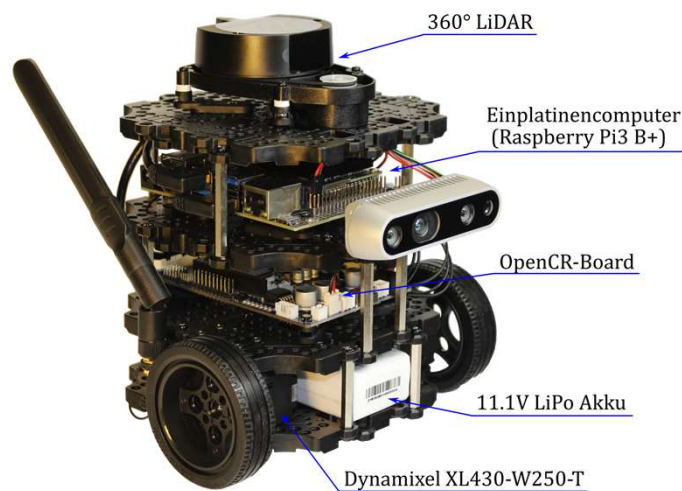
1. Einführung
 2. Technische Grundlagen
 3. Vorstellung des Programms
 4. kritische Betrachtung
- 



Einführung

- Großer Anwendungsbereich von Robotern
- Unterscheidung in verschiedenen Kategorien
- Wie hilft die Wegpunktsteuerung für die weiterentwicklung von Technologien und Konzepten

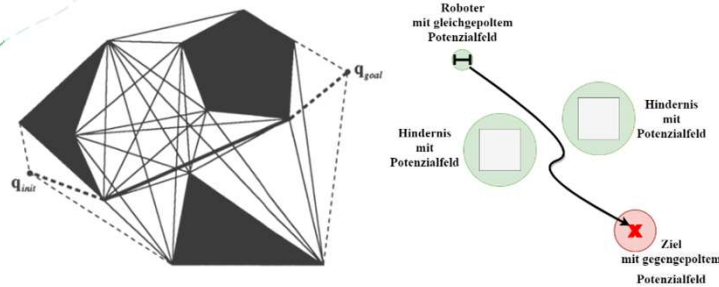
TurtleBot3



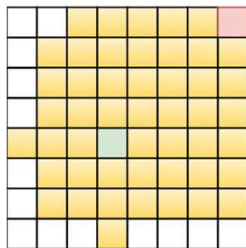
TurtleBot3-Burger:

- mobiler Roboter von u.a. Open Robotics, ROBOTIS
- Programmierbar mit ROS
- Aufbau:
 - LiDAR
 - Einplatinen Computer
 - OpenCR-Board
 - Akku

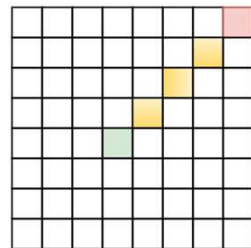
Pfadplanung



Dijkstra-Algorithmus



A*-Algorithmus



Hindernisse

- Gitter
- Polyeder

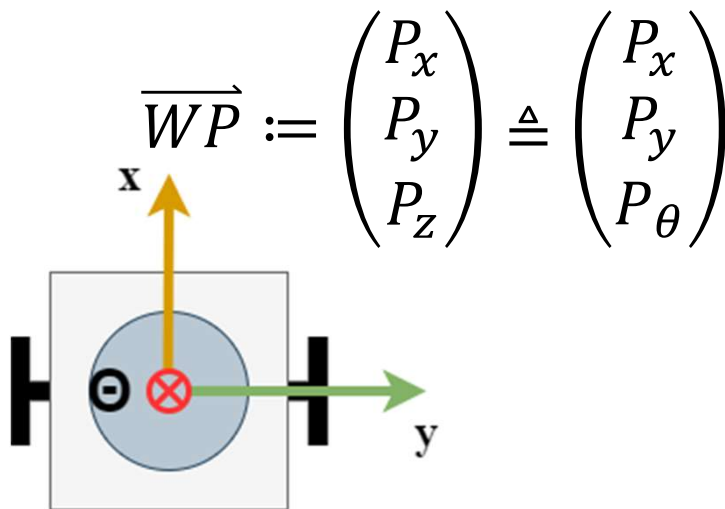
Global-Planner → besten Pfad berechnen

- Roadmap
- Zellzerlegung
- Potentialfeld

Local-Planner → Pfad anpassen an dynamische Umgebung

- Setzen von Zwischenpunkten im globalen Pfad

Wegpunkte



Koordinatensystem der Karte

- kartesisch

Koordinatensystem des Turtlebots

- kartesisch
 - X → nach Norden
 - Y → nach Osten
 - Z → Orientierung

Gazebo



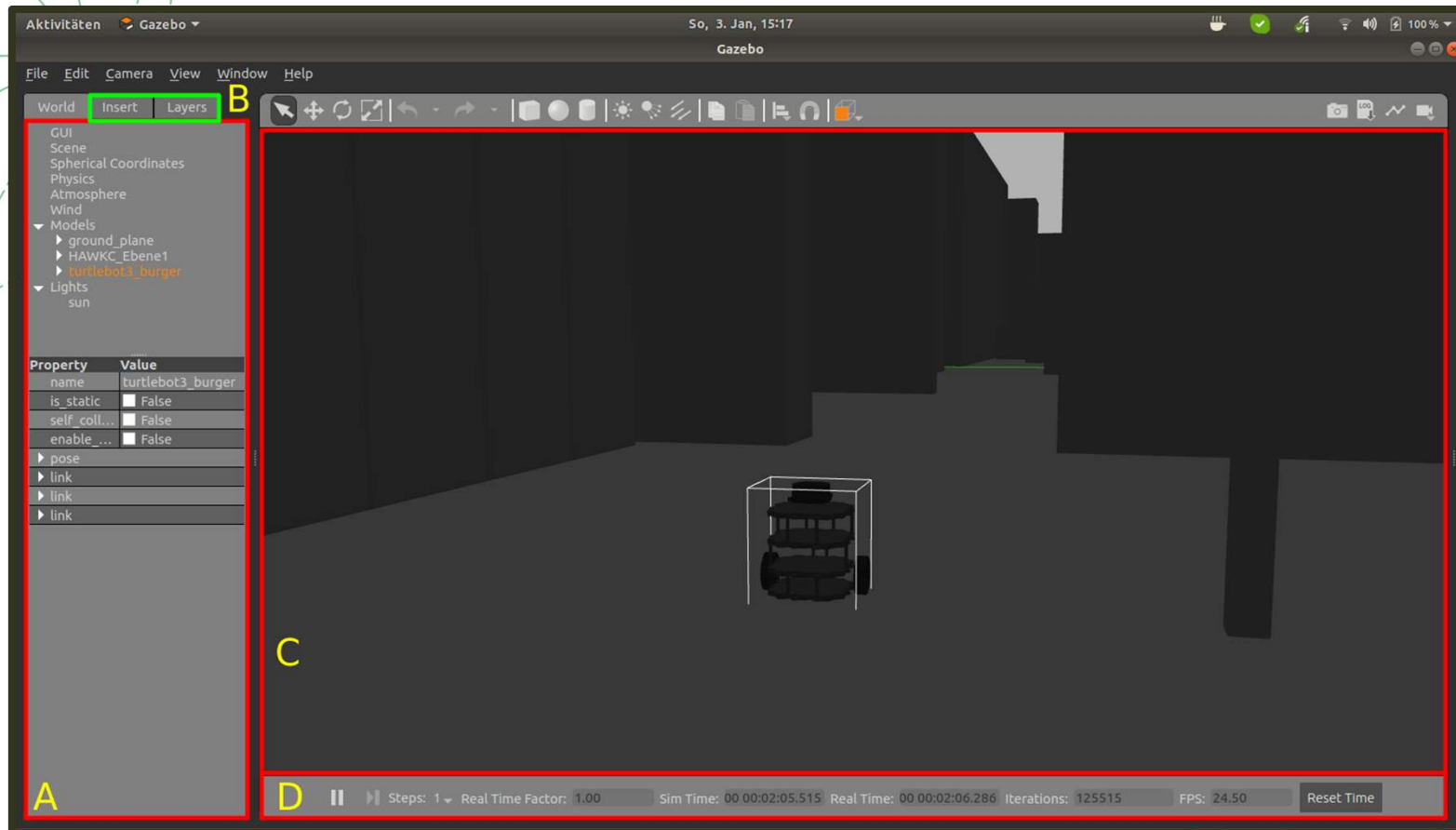
Simulationsumgebung (**Linux**)

2002 durch Dr. Andrew Howard, Nate Koenig (**USC**)

Seit **2012** *Open Source Robotics Foundation* (**OSRF**)

- Gazebo **Server**: Physik- und Sensor-Engine
- Graphical **Client**: GUI, Visualisierung

```
$ roslaunch turtlebot3_gazebo turtlebot3_HAWK1.launch
```



Modellbildung

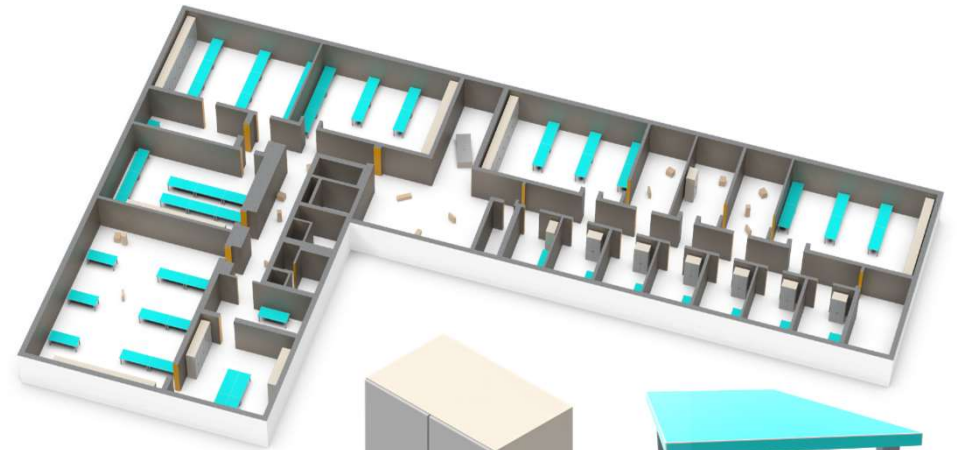
2D/3D CAD Software

Solid Edge (**Siemens**)

- Grundriss, Tisch, Schrank, Kartonbox

Über Gazebo *Modell-Builder* **.stl** importieren

<https://github.com/regenhardthawk/Robotik-Projekt-WiSe2020>



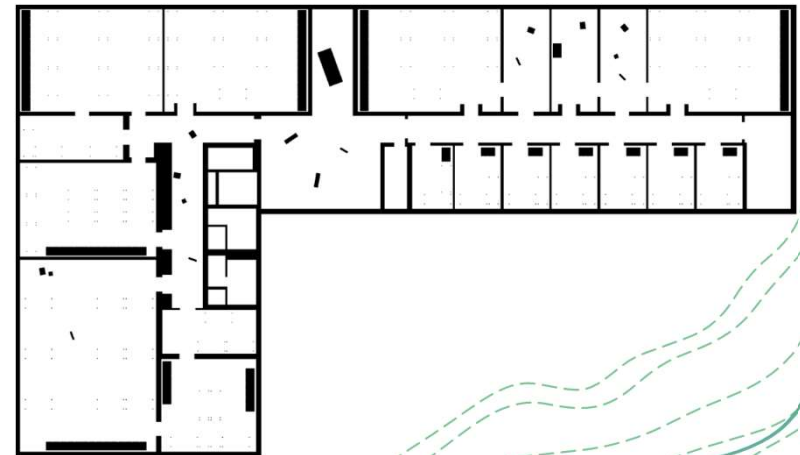
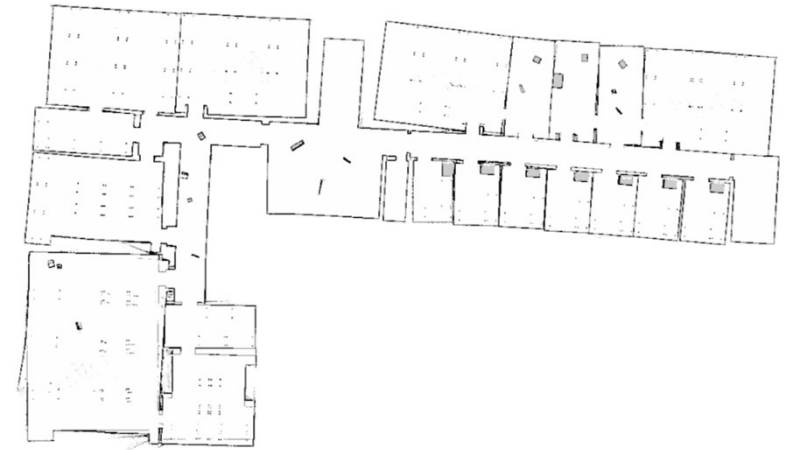
RViz

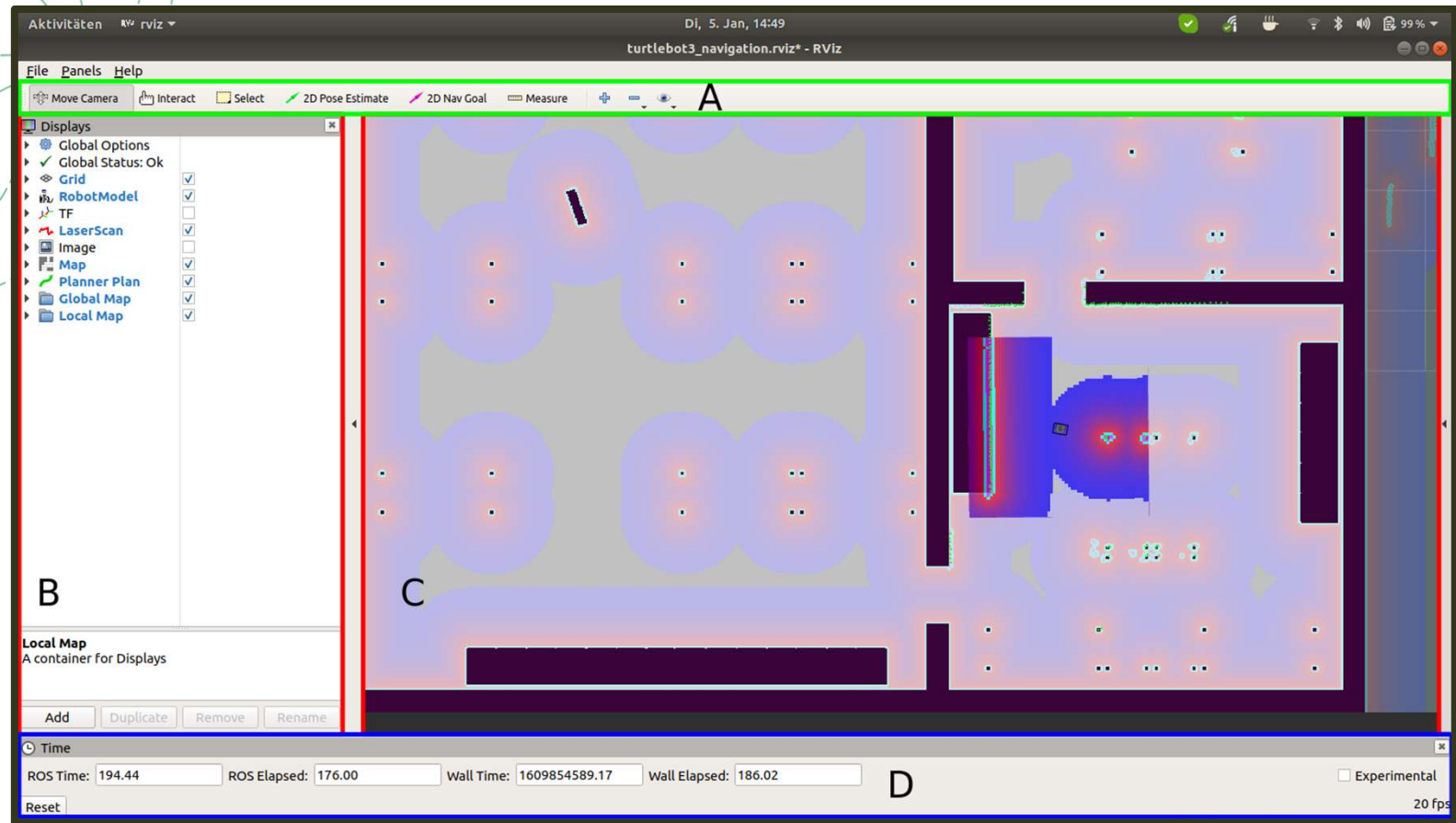
Visualisierung der Sensorik

Abmessung: 2940 x 1700 pixel

Maßstab: 0.01845 m/px

```
$ roslaunch turtlebot3_navigation turtlebot3_navigation  
.launch map_file:=$HOME/Fak_I_Map2.yaml
```







Das Programm

- Genutzte Bibliotheken
- Wegpunkte einlesen
- Wegpunkte publishen
- Callback
- Aktion am Wegpunkt

Bibliotheken

```
1 #include "ros/ros.h" //ROS-Standard Bibliothek
2 #include "geometry_msgs/PoseStamped.h" //Publisher Objekt
3 #include "move_base_msgs/MoveBaseActionResult.h" //Subscriber Objekt
4 #include <tf/tf.h> //Transformation (für Quaternionen)
5 #include <vector>
6 #include <iostream> //C++ Bibliothek für Konsolenausgabe
7 #include <fstream> //Zum Auslesen der Wegpunkt-Datei
8 #include <string>

10 namespace rob_proj4
11 {
12     class waypoint_nav
13     {
14     public:
15         waypoint_nav(); //Standardkonstruktor
16
17     private:
18
19         //Callback wenn der Action-Server die Ankunft-Nachricht für einen Wegpunkt published
20         void mb_resultCallback(const move_base_msgs::MoveBaseActionResult::ConstPtr& msg);
21
22         //Funktionen zum Auslesen und Publiken der Wegpunkte. Und Aktion nach ankunft am Wegpunkt.
23         void read_waypoints();
24         void update_waypoints();
25         void action_waypoints();
26
27         //Handler für das abonnieren des Ergebnisses der Navigation
28         ros::NodeHandle m_ns;
29         ros::Subscriber m_waypoint_sub;
30
31         //Handler für das Publiken eines Wegpunktes
32         ros::NodeHandle m_np;
33         ros::Publisher m_waypoint_pub;
34
35         std::vector<float> m_wp; //Vektor-Objekt zum zwischenspeichern der Wegpunkte aus der Datei.
36         int m_num_wp; //Zähler, wie viele Wegpunkte geladen wurden.
37         int seq; //Aktueller Stand (Sequenz) der Wegpunkte.
38         bool m_update; //Schalter, ob entweder ein Wegpunkt angefahren oder eine Aktion ausgeführt werden soll.
39     };
40 }
```

- ros.h - Publisher, Subscriber und Timer
- PoseStamped.h für Angabe der Position
- MoveBaseActionResult.h für Abfrage ob das Ziel erreicht
- tf.h Transformation der Quaternion

Wegpunkte aus einer .txt-Datei einlesen

```
38 /***** Funktion: Einlesen der Wegpunkte aus einer .txt Datei *****/
39 void waypoint_nav::read_waypoints(){
40
41     float m_x, m_y, m_w; //Variablen für die Koordinatenübergabe
42
43     std::ifstream inputfile("/home/jan/catkin_ws/src/waypointnavigation/src/waypoints.txt"); //Dateipfad
44     std::string line; //Variable für die aktuelle Zeile
45
46     while (std::getline(inputfile, line)){ //Lese Zeile ein und führe Aktion bis Zeilenbruch aus.
47         for (int i = 0; i < line.size(); i++){ //Jedes einzelne Zeichen in der aktuellen Zeile durchgehen.
48             if (line[i] == ',' || line[i] == '\t'){ //Ersetze alle Tabulatoren und Komma mit Leerzeichen
49                 line[i] = ' ';
50             }
51         }
52
53         if (line[0] != '#'){ //Wenn KEIN Kommentarzeichen "#" genutzt wurde, verarbeite aktuelle Zeile
54             std::stringstream linebuffer(line); //Zwischenspeichern der aktuellen Zeile
55             linebuffer >> m_num_wp >> m_x >> m_y >> m_w; //Abspeichern der Koordinaten in Zwischenvariablen
56             std::cout << m_num_wp << " : " << m_x << "\t" << m_y << "\t" << m_w << std::endl; //Ausgabe der Wegpunkte
57
58             m_wp.push_back(m_x); //x-Koordinate an Vektor hinten anhängen
59             m_wp.push_back(m_y); //y-Koordinate an Vektor hinten anhängen
60             m_wp.push_back((m_w*3.14)/180); //z-Achsenrotation an Vektor hinten anhängen (Umrechnung von Deg in Rad)
61         }
62     }
63 }
64
65 }
```

Der Turtlebot3 soll beliebig viele Wegpunkte aus einer .txt-Datei lesen können.

C++: Einlesen und Zuordnen durch Stream-Operator >> möglich.

=> Komma und Tabulatoren mit Leerzeichen ersetzen (Zeile 48)

=> Zeilen die mit '#' beginnen dienen als Kommentar und werden nicht eingelesen

Übergabe der Werte durch *push_back* hinten an den Vektor

Wegpunkte publishen

```
66 /***** Funktion: Publishen eines Wegpunktes *****/
67 void waypoint_nav::update_waypoints(){
68
69     if(seq < m_num_wp){ //Nur ausführen, wenn neue Wegpunkte vorhanden sind
70         geometry_msgs::PoseStamped Wegpunkt; //Objekt Wegpunkt vom Datentyp PoseStamped (Enthält Position und Orientierung)
71
72         //Zusammenbau der Nachricht für den Topic
73         Wegpunkt.header.seq = seq; //Sequenz des Topics
74         Wegpunkt.header.stamp = ros::Time::now(); //Aktueller Zeitstempel in ROS-Zeit
75         Wegpunkt.header.frame_id = "map"; //frame_id (wichtig für RViz)
76
77         Wegpunkt.pose.position.x = m_wp[(seq)*3+0]; //Eingabe der x-Koordinate
78         Wegpunkt.pose.position.y = m_wp[(seq)*3+1]; //Eingabe der y-Koordinate
79         Wegpunkt.pose.orientation = tf::createQuaternionMsgFromYaw(m_wp[(seq)*3+2]); //Eingabe der z-Achsenrotation mit Umrechnung in Quaternionen
80
81         m_waypoint_pub.publish(Wegpunkt); //Publish des Wegpunktes
82         m_update = false; //Kein Publishen eines neuen Wegpunktes, zunächst die Aktion
83     }
84     else std::cout << "All Waypoints processed" << std::endl;
85 }
86
```

Werte aus der Textdatei in Variable m_wp gespeichert

seq = Aktueller Stand (Sequenz) der Wegpunkte

Zuordnen der Werte:

1. Wert für x-Achse
2. Wert für y-Achse
3. Orientierung

Beispiel für ersten Messwert seq=0:

$0*3+0=0$ ->ließ den Wert an der Stelle 0 für Position.x

$0*3+1=1$ ->ließ den Wert an der Stelle 1 für Position.y

Callback

```
18 /***** Callback-Funktion: Erreichen/Abbruch der Wegpunktnavigation *****/
19 void waypoint_nav::mb_resultCallback(const move_base_msgs::MoveBaseActionResult::ConstPtr& msg){
20
21     if(msg->status.status == 3){ //3: Ziel wurde vom Action Server erfolgreich angefahren.
22
23         switch(m_update){ //Entscheidet, ob ein neuer Wegpunkt angefahren, oder eine Aktion ausgeführt werden soll.
24             case true:
25                 update_waypoints();
26                 break;
27             case false:
28                 std::cout << "SUCCEEDED: " << seq-1 << ". Goal Reached" << std::endl;
29                 action_waypoints();
30                 break;
31         }
32     }
33     else{
34         std::cout << "NOT SUCCEEDED: Goal not Reached or Aborted" << std::endl;
35     }
36 }
37
```

Abfrage ob der Wegpunkt erreicht wurde über Variable *status* aus MoveBaseActionResult.h

Entscheidung, ob ein neuer Wegpunkt angefahren, oder eine Aktion ausgeführt werden soll

Aktion am Wegpunkt

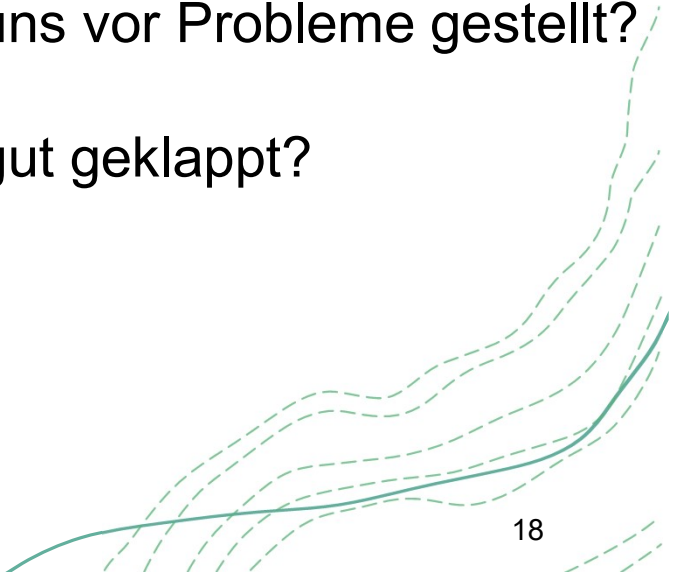
```
87  /***** Funktion: Aktion nach Ankunft an einem Wegpunkt *****/
88  void waypoint_nav::action_waypoints(){
89
90      if(seq <= m_num_wp){
91          geometry_msgs::PoseStamped Wegpunkt;    //Objekt Wegpunkt vom Datentyp PoseStamped (Enthält Position und Orientierung)
92
93          //Zusammenbau der Nachricht für den Topic
94          Wegpunkt.header.seq = seq; //Sequenz des Topics
95          Wegpunkt.header.stamp = ros::Time::now(); //Aktueller Zeitstempel in ROS-Zeit
96          Wegpunkt.header.frame_id = "map"; //frame_id (wichtig für RVis)
97
98          Wegpunkt.pose.position.x = m_wp[(seq-1)*3+0]; //Eingabe der x-Koordinate
99          Wegpunkt.pose.position.y = m_wp[(seq-1)*3+1]; //Eingabe der y-Koordinate
100
101          //Hier die Aktion: Drehung um 180° am Wegpunkt
102          Wegpunkt.pose.orientation = tf::createQuaternionMsgFromYaw(m_wp[(seq-1)*3+2]-(180*3.14)/180);
103
104          m_waypoint_pub.publish(Wegpunkt); //Publish der Aktion am Wegpunkt
105          seq++; //Erhöhung des Sequenzzählers um 1
106      }
107      m_update = true; //Nächster Wegpunkt kann gepublished werden
108  }
109  //end namespace rob_proj4
110
```


180°-Drehung
wenn das Ziel
erreicht worden
ist (Zeile 102)

In Radian
umgerechnet



kritische Betrachtung

- Was hat nicht so funktioniert, wie vorgestellt?
 - Was hat uns vor Probleme gestellt?
 - Was hat gut geklappt?
- 



**Wir bedanken uns für Ihre
Aufmerksamkeit.**