Übung 3— Navigation, Lokalisierung und Kartierung

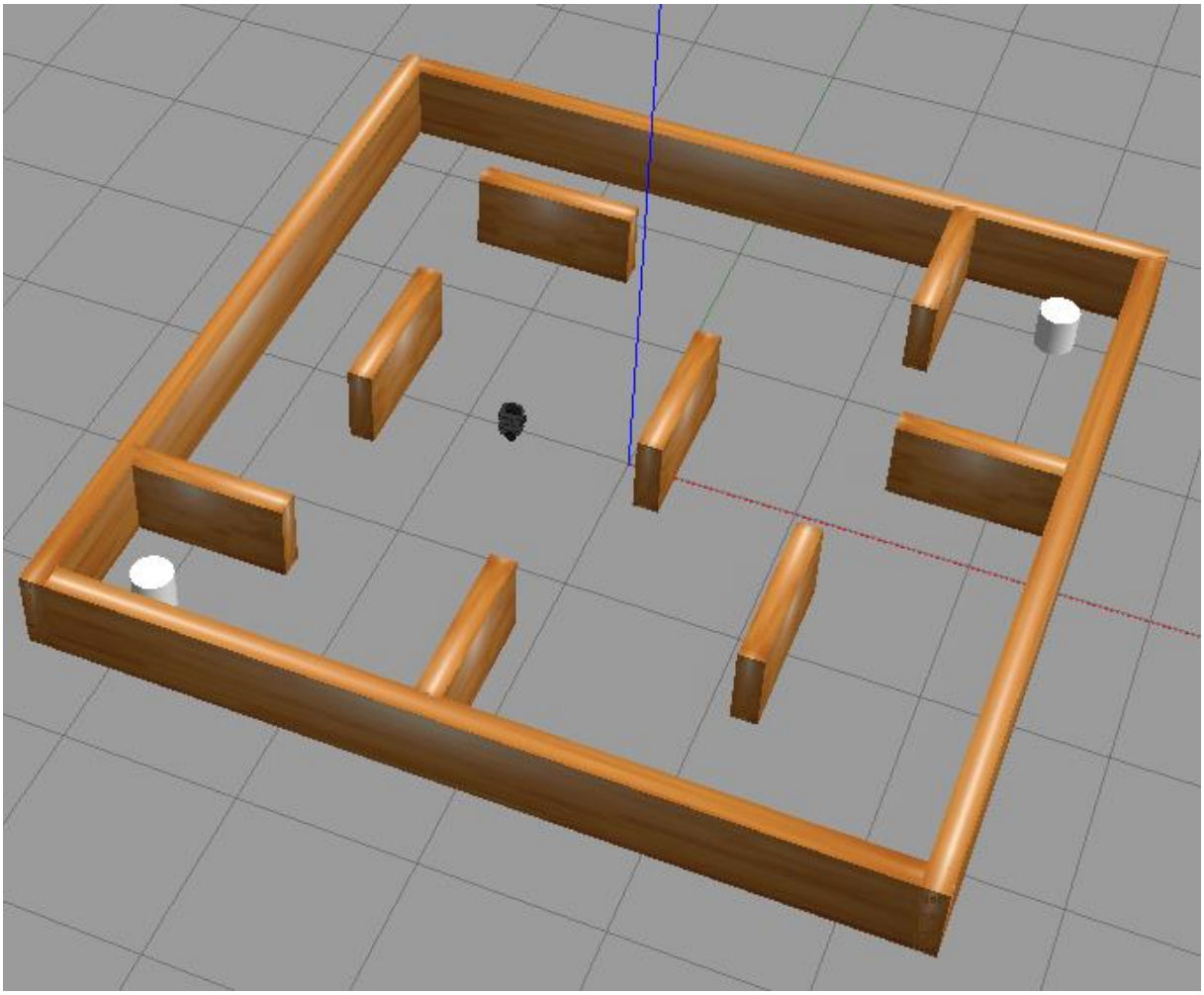
Teil 1- Navigation

```python
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import math

class NavigationNode:
    def __init__(self):
        rospy.init_node('navigation_node')

        self.velocity_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rospy.Subscriber('/scan', LaserScan, self.laser_callback)

        self.d_min = 0.2
        self.d_max = 1.0
        self.v_max = 0.5

    def calc_weight(self, angle, distance):
        w_ori = math.cos(angle / 1.2)
        w_dist = 1.0 / (1.0 + math.exp(-(distance - 1.5) / 0.20))
        w_dist = 1.0 if distance > 3.5 else w_dist
        return w_ori * w_dist

    def laser_callback(self, laser_data):
        range1 = range(330, 360)
        range2 = range(0, 31)
        desired_range = list(range1) + list(range2)

        filtered_ranges = [1.0 if math.isinf(val) or val == 0 else val
                                for i, val in enumerate(laser_data.ranges)]
        min_distance = min(val for i, val in enumerate(filtered_ranges) if i in desired_range)

        v = 0.0
        if min_distance > self.d_min:
            if min_distance < self.d_max:
                v = self.v_max * min_distance / self.d_max
            else:
                v = self.v_max

        range1 = range(270, 360)
        range2 = range(0, 91)
        desired_range = list(range1) + list(range2)

        weights = []
        angle = 0
        for (i, d) in enumerate(filtered_ranges):
            if i in desired_range:
                weights.append((angle, self.calc_weight(math.radians(angle), d)))
                angle += 1
                if angle > 90:
                    angle = -90

        alpha = math.atan2( sum(math.sin(math.radians(angle)) * w for i,(angle,w) in enumerate(weights)),
                            sum(math.cos(math.radians(angle)) * w for i,(angle,w) in enumerate(weights)))

        twist_msg = Twist()
        twist_msg.linear.x = v
        twist_msg.angular.z = alpha
        self.velocity_pub.publish(twist_msg)

if __name__ == '__main__':
    try:
        nav_node = NavigationNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

Um den oben gezeigten Code zu testen, wird zunächst eine Gazebo Simulation gestartet. Hierfür wird folgender Command ausgeführt:

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch
... logging to /home/vm/.ros/log/3192c946-9c1d-11ee-b418-0800275bd42b/roslaunch-ros-vm-2020-25496.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

Dies startet Gazebo mit der folgenden Welt:

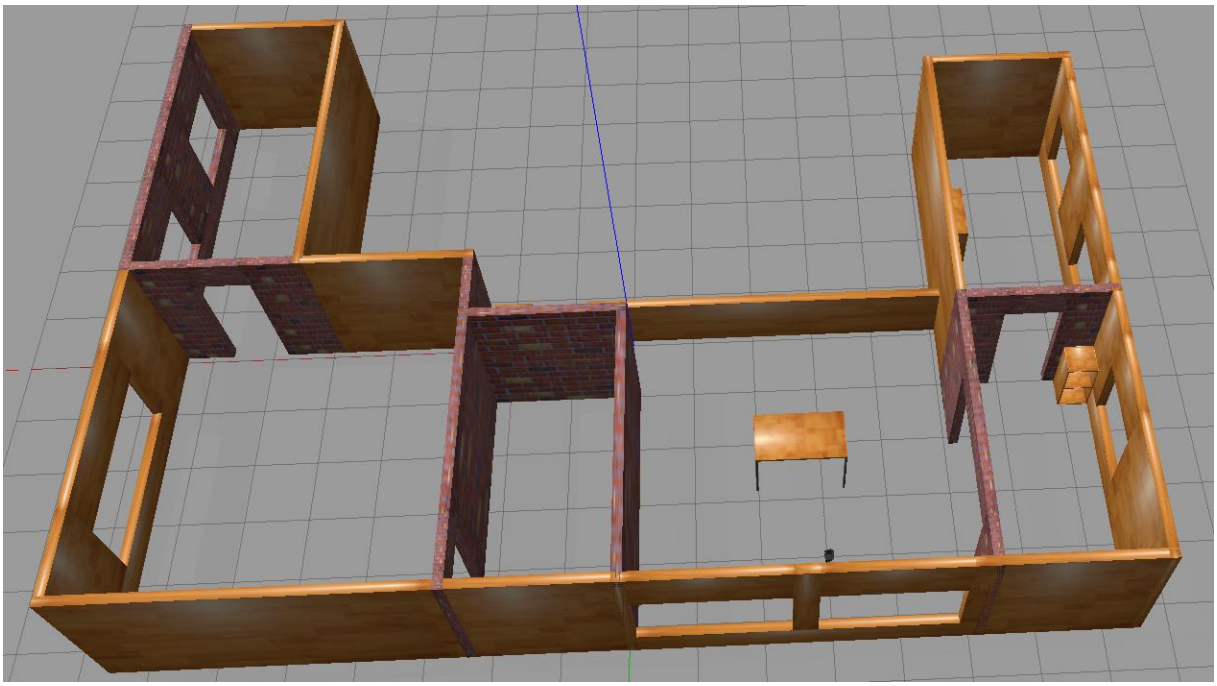


Anschließend wird mit folgendem Command der vorher gezeigte Node ausgeführt:

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrun exercise-003 NavigationNode.py
```

Teil 2 – Kartierung

Hierfür wurde für Testzwecke folgende Map verwendet:



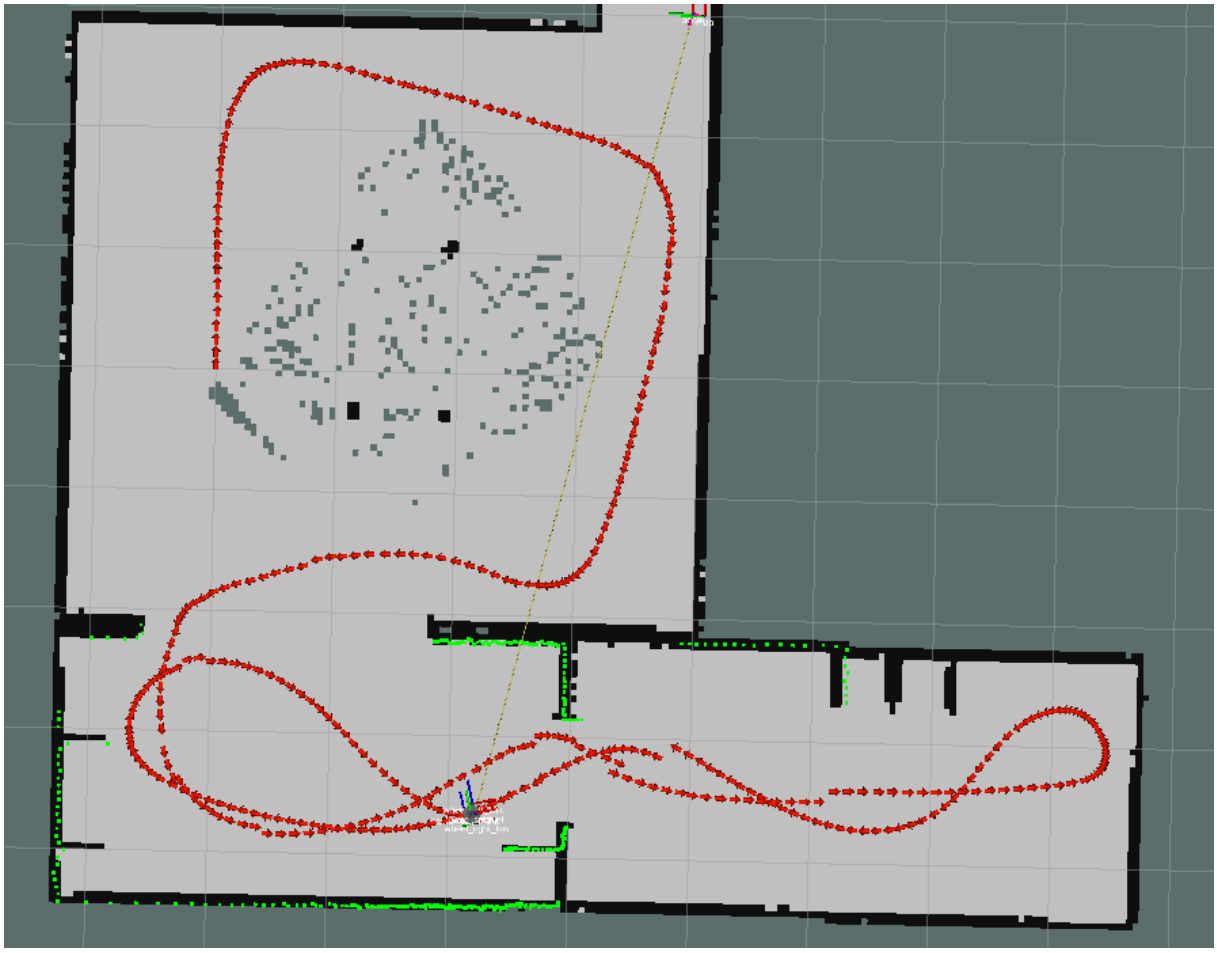
Anschließend wurde ein Rosbag aufgenommen während der Roboter mithilfe der Node aus Teil 1 durch die Map navigiert. Anschließend wurde folgender Command ausgeführt, um RViz zu starten und für das SLAM-Verfahren alles vorzubereiten:

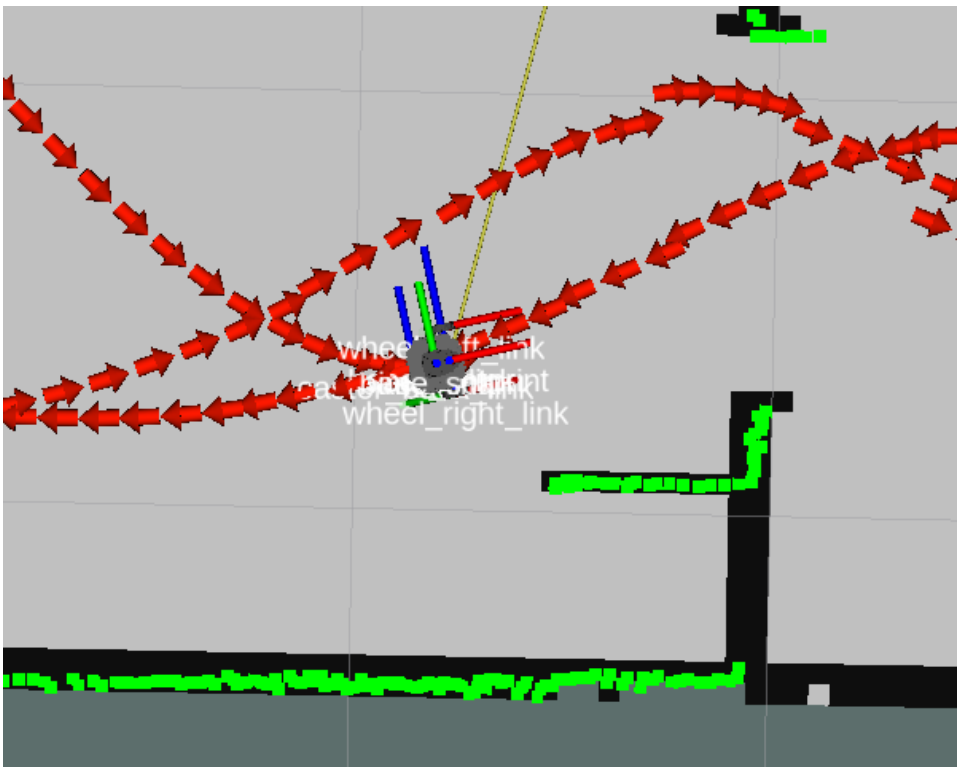
```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch turtlebot3_slam turtlebot3_slam.launch
... logging to /home/vm/.ros/log/3192c946-9c1d-11ee-b418-0800275bd42b/roslaunch-ros-vm-2020-7059.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

In RViz wurde noch eine Odometry eingefügt, um die Trajektorie mit anzuzeigen. Anschließend wurde das aufgenommene bag-File mit folgenden Command abgespielt:

```
^Cvm@ros-vm-2020:~/ARS-ROS-Exercises$ rosbag play src/exercise-003/bagfiles/2023-12-16-16-43-15.bag
[ INFO] [1702741610.265106259]: Opening src/exercise-003/bagfiles/2023-12-16-16-43-15.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING]  Bag Time:    137.108122   Duration: 85.549122 / 85.564000
Done.
```

In den nächsten Abbildungen wird das fertige Ergebnis in RViz gezeigt:





Abschließend wurde mit folgenden Command die Map auf dem map_server gespeichert.

```
^Cvm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrun map_server map_saver -f excercise_3_2_map
[ INFO] [1702741885.740489433]: Waiting for the map
[ INFO] [1702741886.501742749]: Received a 384 X 384 map @ 0.050 m/pix
[ INFO] [1702741886.501800051]: Writing map occupancy data to excercise_3_2_map.pgm
[ INFO] [1702741886.502725138]: Writing map occupancy data to excercise_3_2_map.yaml
[ INFO] [1702741886.502962373]: Done
```

Teil 3 – Monte Carlo Lokalisierung

Zunächst wird wieder die Gazebo-Simulation mit folgenden Command gestartet.

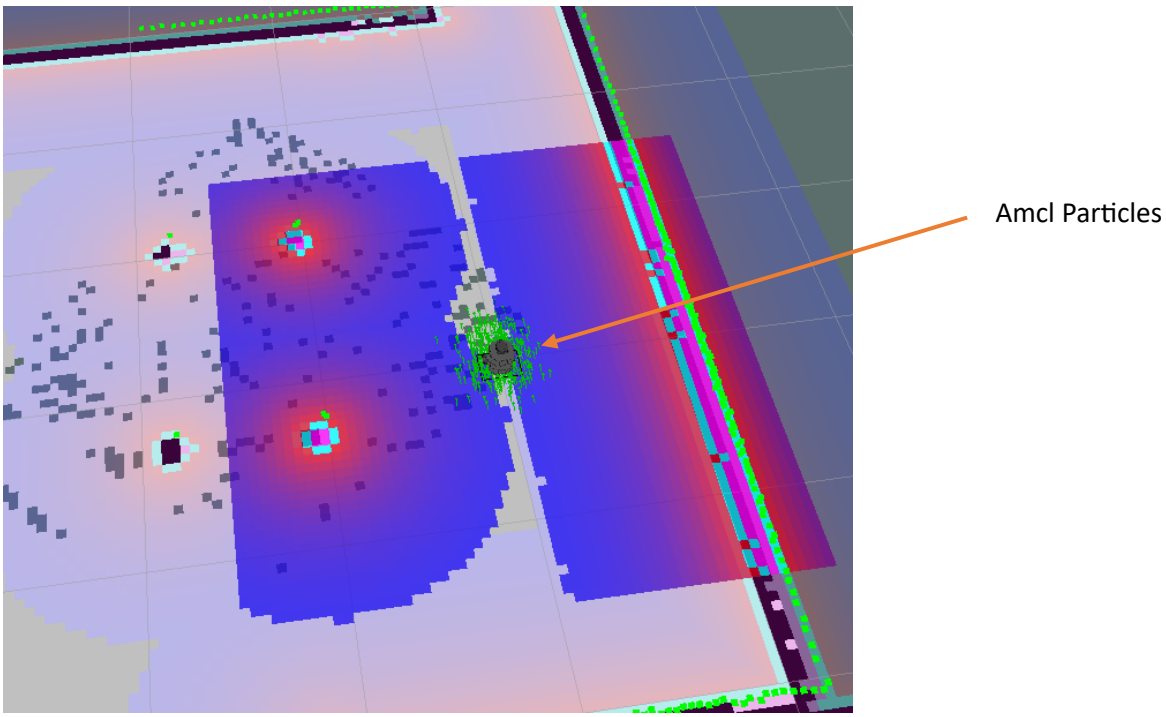
```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
... logging to /home/vm/.ros/log/326ac49e-9ce6-11ee-84d9-0800275bd42b/roslaunch-ros-vm-2020-2636.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

Anschließend wird indirekt durch ausführend des nachfolgenden Commands AMCL und RViz gestartet.

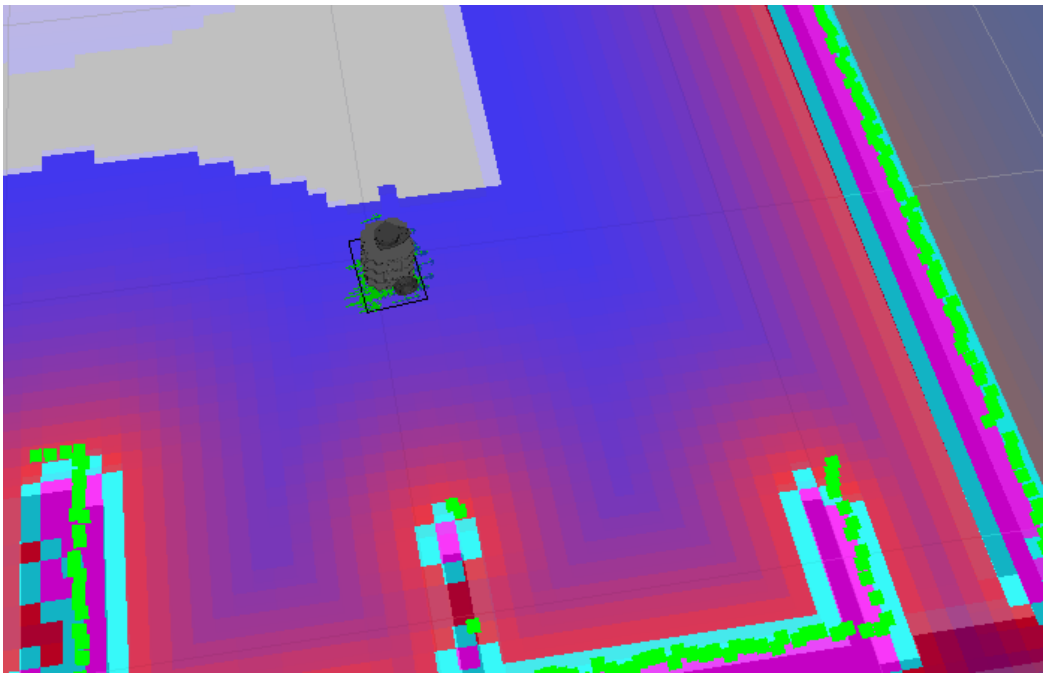
```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch
... logging to /home/vm/.ros/log/326ac49e-9ce6-11ee-84d9-0800275bd42b/roslaunch-ros-vm-2020-5179.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

Hierfür muss jedoch im Skript „amcl.launch“ die initial Pose angegeben werden, hierfür wurde die initial Pose von „turtlebot3_house.launch“ übernommen. Theoretisch würde AMCL keine korrekte initial Pose benötigen, passt die Pose nicht mit den Sensordaten überein, werden die „Amcl Particles“ stark über die Map gestreut und je weiter der Roboter bewegt wird, desto geringer wird die Streuung. Trotzdem wurde hier die richtige initial Pose angegeben. Und im „turtlebot3_navigation.launch“ muss die im vorherigen Teil erstellte Karte bzw. der Dateipfad zur Karte angegeben werden.

Anschließend wird in RViz durch die „Amcl Particles“ folgendes dargestellt:

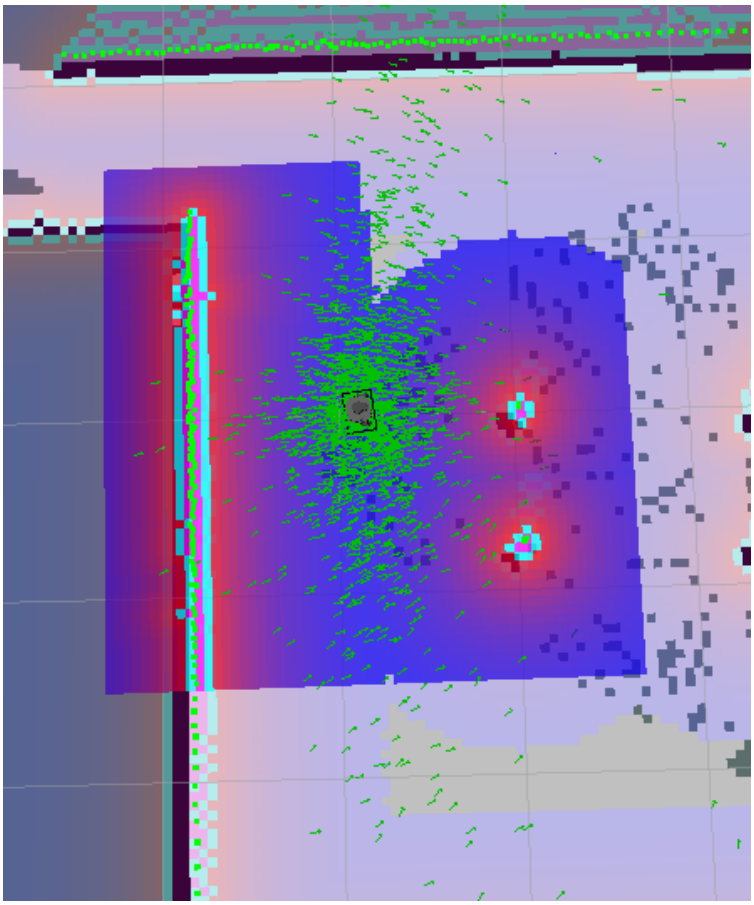


Man erkennt das zu Beginn die geschätzte Pose die durch die „Amcl Particles“ noch relatiiv weit gestreut sind, fährt man jedoch mit dem Turtlebot durch die Map wird die Streuung immer geringer, siehe nachfolgende Abbildung und nachfolgenden Command um die Steuerung des Turtlebots zu aktiveren.



```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

... logging to /home/vm/.ros/log/326ac49e-9ce6-11ee-84d9-0800275bd42b/roslaunch-ros-vm-2020-4699.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

In der nächsten Abbildung wird das Kidnapped-Robot-Problem gezeigt, hierfür wurde der Roboter in Gazebo verschoben. Dementsprechend haben die „Amcl Particles“ in RViz wieder eine große Streuung.



Um dieses Problem zu verbessern kann man im „amcl.launch“-File folgende Parameter optimieren:

- Recovery_alpha_slow / recovery_alpha_fast: Diese Parameter beeinflussen die Art und Weise, wie AMCL versucht, sich nach unerwarteten Ereignissen zu erholen, wie zum Beispiel, wenn der Roboter "gekidnappt" wird. Die Werte sollten so eingestellt werden, dass sie eine angemessene Balance zwischen Robustheit und Konsistenz bieten. Ich habe die Werte von 0 auf 0.1 jeweils geändert.
- Erhöhung der Partikelanzahl: Um generelle Robustheit zu erhöhen habe ich „min_particles“ und „max_particles“ auf 1000 bzw. 5000 erhöht.
- Optimierung update_min_d / update_min_a: Diese Werte geben an, wie schnell sich die Partikeln updaten, bzw. wie viel Translation und Rotation durchgeführt werden müssen, bis die Partikeln geupdated werden.

Anmerkung:

Für Videos siehe https://github.com/markus-senger/ARS-ROS-Exercises/tree/main/src/exercise-003