

Um die Anzahl der Messwerte zu bestimmen, wurde ein einfaches Python-Skript geschrieben (siehe Code-Snippet 1). Es werden 360 Messwerte vom Laserscanner geliefert, Abbildung 3 zeigt den Aufruf dieses Skriptes.

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

import rospy
from sensor_msgs.msg import LaserScan

def laser_scan_callback(data):
    rospy.loginfo("Anzahl der Messwerte in 'ranges': %d" %
                  len(data.ranges))

rospy.init_node('laser_scan_listener')
rospy.Subscriber('/scan', LaserScan, laser_scan_callback)
rospy.spin()
```

Code-Snippet 1 - Python-Skript zum Bestimmen der Anzahl der Messwerte

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrn laser-scanner-statistics-1-1 checkNumbersOfMe
asurements.py
[INFO] [1699179989.638143]: Anzahl der Messwerte in 'ranges': 360
```

Abbildung 3 - Aufruf von checkNumbersOfMeasurements.py

Um die Winkelauflösung zu berechnen, verwendet man aus Abbildung 1 den Wert *angle_increment*, zu beachten ist jedoch, dass dieser Wert in Radianten angegeben ist. Die Winkelauflösung beträgt daher 1 Grad.

Betrachtet man in Abbildung 1 die Werte *angle_min* bzw. *angle_max* die in Radianten angegeben werden, erkennt man, dass das Field of View dieses Sensors 360 Grad beträgt.

2. **Nehmen Sie ein Bagfile mit einer Länge von ca. 5 Minuten auf, dass mindestens die Laserscandaten enthält, in dem der Roboter sich nicht bewegt.**

Abbildung 4 zeigt den Command um ein Bagfile für das Topic */scan* aufzunehmen.

```
vm@ros-vm-2020:~$ rosbag record scan
[ INFO] [1699199205.738768304]: Subscribing to scan
[ INFO] [1699199206.070836980, 652.762000000]: Recording to '2023-11-05-16-46-46.bag'.
```

Abbildung 4 - Command zum Aufnehmen eines Bagfiles für das Topic */scan*

Für diese Übung wurden zwei Bagfiles aufgenommen, einmal in einer Simulationsumgebung und einmal mit der echten Hardware.

3. Zeigen Sie die Daten in rviz an und machen Sie einen Screenshot, der die Umgebung zeigt.

Um das aufgenommene Bagfile in RViz anzuzeigen, wird RViz zunächst ganz normal gestartet. Anschließend wird der Command aus Abbildung 5 ausgeführt, um das Bagfile wieder abzuspielen.

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosbag play src/exercise-001/laser-scanner-statistics-1-1/bagfiles/laserdata_simulation.bag
[ INFO] [1699200463.106020826]: Opening src/exercise-001/laser-scanner-statistics-1-1/bagfiles/laserdata_simulation.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
[RUNNING] Bag Time: 660.654095 Duration: 7.833095 / 287.338000
```

Abbildung 5 - Command zum Abspielen des Bagfiles aus der Simulation

Anschließend wird in RViz mithilfe von Add das Topic `/scan` hinzugefügt. Anschließend werden die Daten des Bagfiles angezeigt. Zu beachten ist jedoch, dass in RViz der richtige ‚Fixed Frame‘ ausgewählt ist. Die nachfolgenden Abbildungen zeigen einerseits die Daten der echten Hardware und andererseits die Daten aus der Simulation.

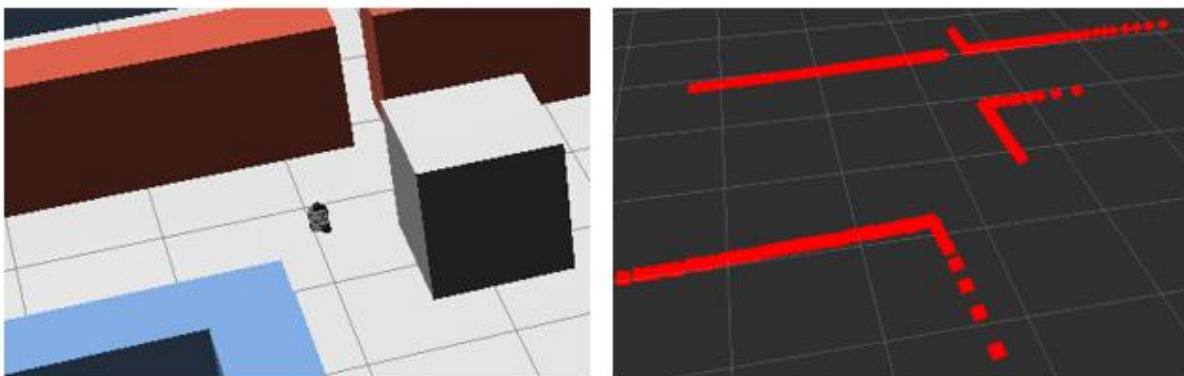


Abbildung 6 - Simulationsumgebung (links) und visualisierte Daten in RViz (rechts)

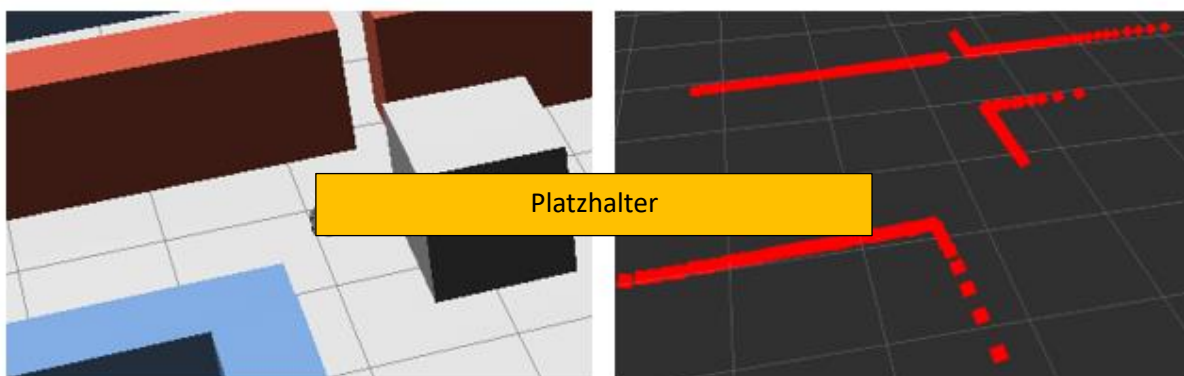


Abbildung 7 - Reale Umgebung (links) und visualisierte Daten in RViz (rechts)

4. Schreiben Sie einen ROS-Node zur Berechnung von Mittelwert, Varianz und Standardabweichung für jede der n Messrichtungen des Scanners. Plotten Sie die Werte (z.B. mit gnuplot). Ergeben sich signifikante Abweichungen in einzelne Richtungen? Wie interpretieren Sie das?

Code-Snippet 2 zeigt das erstellte Python-Skript.

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

import rospy
from sensor_msgs.msg import LaserScan
import statistics
import matplotlib.pyplot as plt
import numpy as np
import subprocess
import threading

mean_values = []
variance_values = []
std_deviation_values = []

def laser_scan_callback(data):
    ranges = np.array(data.ranges)
    ranges[np.isinf(ranges)] = 0
    cleaned_ranges = ranges.tolist()

    mean_values.append(statistics.mean(cleaned_ranges))
    variance_values.append(statistics.variance(cleaned_ranges))
    std_deviation_values.append(statistics.stdev(cleaned_ranges))

def play_rosbag(bag_file_path):
    subprocess.call(['rosbag', 'play', bag_file_path])

    plt.figure(figsize=(10, 6))
    plt.plot(mean_values, label='mean')
    plt.plot(variance_values, label='variance')
    plt.plot(std_deviation_values, label='standard deviation')
    plt.legend()
    plt.title('statistics for laserscan')
    plt.ylabel('values')
    plt.xlabel('scan number')
    plt.grid(True)
    plt.show()

rospy.init_node('laser_scan_listener_for_statistics')
rospy.Subscriber('/scan', LaserScan, laser_scan_callback)

bag_file_path = 'src/exercise-001/laser-scanner-statistics-1-1/bagfiles/laserdata_simulation.bag'

play_thread = threading.Thread(target=lambda: play_rosbag(bag_file_path))
play_thread.start()

rospy.spin()
```

Code-Snippet 2 - Skript für statistische Auswertung des Laserscanners

Abbildung 8 zeigt den Command zum Starten der Messungen, sobald das Bagfile abgespielt wurde, werden automatisch die Werte geplottet.

```
vm@gros-vm-2020:~/ARS-R05-Exercises$ roslaunch laser-scanner-statistics-1-1 statisticsForLaserScanner.py
[ INFO] [1699208181.995280848]: Opening src/exercise-001/laser-scanner-statistics-1-1/bagfiles/laserdata_simulation.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
[DELAYED] Bag Time: 652.821000 Duration: 0.000000 / 287.338000 Delay: 169920752
[RUNNING] Bag Time: 664.855233 Duration: 12.034233 / 287.338000
```

Abbildung 8 - Command zum Starten des statisticsForLaserScanner.py Skripts

Ergebnis Simulation:

Abbildung 9 zeigt die geplotteten Werte. Da es sich hierbei um die Werte aus der Simulation handelt, sind wie erwartet keine nennenswerten Abweichungen zu erkennen.

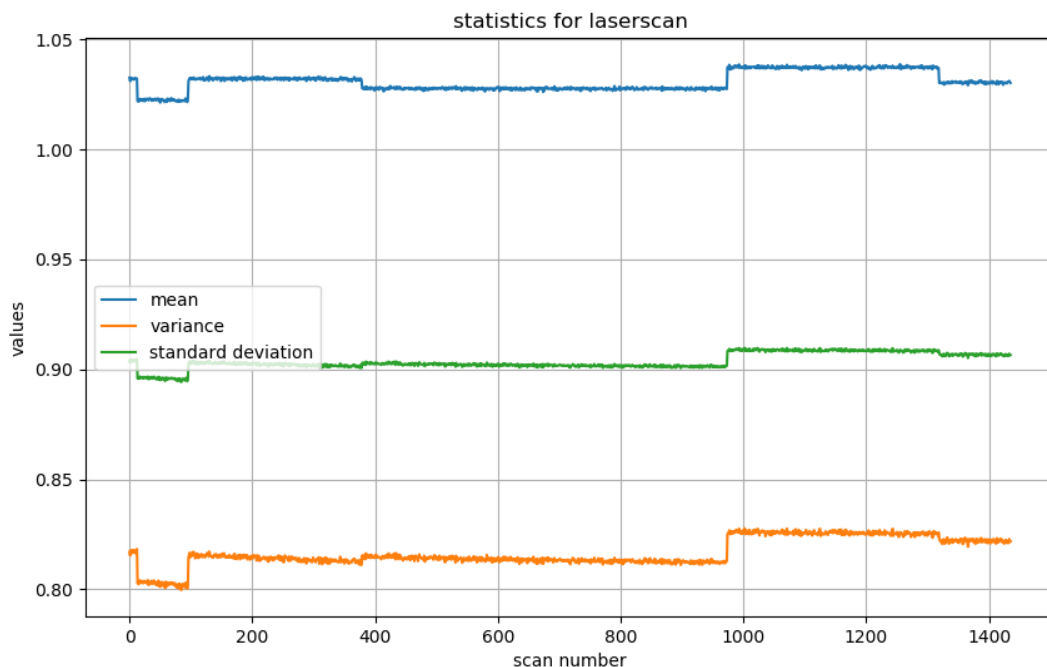


Abbildung 9 - Laserscanner-Statistik aus der Simulation

Ergebnis reale Hardware:

//TODO

Wenn der Mittelwert der Messungen in eine bestimmte Richtung abweicht (z. B. überdurchschnittlich hoch oder niedrig ist), kann dies auf systematische Fehler oder Trends in den Daten hinweisen. Dies könnte auf eine Kalibrierungsproblematik oder einen systematischen Einfluss in den Sensoroperationen hinweisen.

Eine hohe Varianz kann darauf hinweisen, dass die Daten sehr gestreut sind, was auf Unregelmäßigkeiten oder zufällige Störungen in den Messungen hindeuten kann. Eine niedrige Varianz deutet hingegen auf eine geringe Streuung hin, was auf präzisere Messungen hinweisen kann.

Die Standardabweichung ist ein Maß für die Streuung der Daten um den Mittelwert. Eine hohe Standardabweichung bedeutet, dass die Daten weit von ihrem Mittelwert abweichen, während eine niedrige Standardabweichung auf geringe Abweichungen hinweist. Wenn die Standardabweichung in einer bestimmten Richtung besonders hoch ist, kann dies auf eine erhöhte Unsicherheit oder Rauschen in den Messungen in dieser Richtung hinweisen.

5. **Schreiben Sie ein Programm, das alle Entfernungswerte eines einzelnen Laserscans aufsummiert. Erstellen Sie ein Histogramm über die Summen der Entfernungswerte des Scanners aus ihrem Bagfile. Achten Sie auf eine sinnvolle Diskretisierung des Histogramms. Interpretieren Sie die Ergebnisse.**

Code-Snippet 3 zeigt das erstellte Python-Skript und Abbildung 10 zeigt den Command zum Starten des Skripts.

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

import rospy
from sensor_msgs.msg import LaserScan
import matplotlib.pyplot as plt
import numpy as np
import subprocess
import threading

sum_values = []

def laser_scan_callback(data):
    ranges = np.array(data.ranges)
    ranges[np.isinf(ranges)] = 0
    cleaned_ranges = ranges.tolist()

    sum_values.append(round(sum(cleaned_ranges), 1))

def play_rosbag(bag_file_path):
    subprocess.call(['rosbag', 'play', bag_file_path])

    plt.hist(sum_values, bins=len(sum_values), edgecolor='black')

    plt.xlabel('values')
    plt.ylabel('frequency')
    plt.title('histogram summed values')
    plt.show()

rospy.init_node('sum_all_Values_from_laser_scan')
rospy.Subscriber('/scan', LaserScan, laser_scan_callback)

bag_file_path = 'src/exercise-001/laser-scanner-statistics-1-1/bagfiles/laserdata_simulation.bag'

play_thread = threading.Thread(target=lambda: play_rosbag(bag_file_path))
play_thread.start()

rospy.spin()
```

Code-Snippet 3 - Skript für Histogramm Erstellung von aufsummierten Entfernungen

```

vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrn laser-scanner-statistics-1-1 sumAllValuesFrom
LaserScanner.py
[ INFO] [1699208997.809309847]: Opening src/exercise-001/laser-scanner-statistics-1-1/ba
gfiles/laserdata_simulation.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
[DELAYED] Bag Time: 652.821000 Duration: 0.000000 / 287.338000 Delay: 169920834
[RUNNING] Bag Time: 812.999064 Duration: 160.178064 / 287.338000

```

Abbildung 10 - Command zum Starten des sumAllValuesFromLaserScanner.py Skripts

Ergebnis Simulation:

Abbildung 11Abbildung 9 zeigt die geplotteten Werte. Da es sich hierbei um die Werte aus der Simulation handelt, sind wie erwartet keine großen Streuungen im Histogramm erkennbar. Die Werte wurden auf eine Kommastelle aufgerundet.

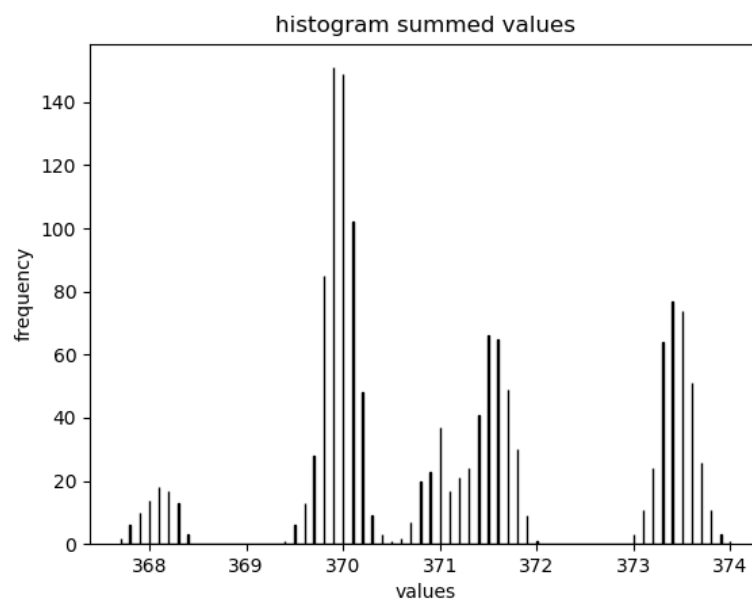


Abbildung 11 - Histogramm der aufsummierten Entfernungen

Ergebnis reale Hardware:

//TODO

Odometrie

1. Der *turtlebot node* berechnet die Odometrie des Turtlebots.

Abbildung 12 zeigt den Command und einen Teil der Ausgabe der Odometrie vom Turtlebot.

```
vm@ros-vm-2020:~/ARS-R0S-Exercises$ rostopic echo /odom
header:
  seq: 9410
  stamp:
    secs: 436
    nsecs: 888000000
  frame_id: "odom"
child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: -11.8369759986
      y: -8.81693968396
      z: -0.00100344172838
    orientation:
      x: -0.00219255015299
      y: 0.00318723698412
      z: 0.565081417831
      w: 0.825026075628
  covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001]
twist:
  twist:
    linear:
      x: -1.07206194692e-06
      y: 8.38015373203e-06
```

Abbildung 12 - Odometrie die vom Turtlebot zur Verfügung gestellt wird

2. In der Vorlesung wurden die Berechnung der Odometrie vorgestellt. Implementieren Sie diese Berechnung, um direkt aus den Sensordaten des Turtlebots die Odometrie zu bestimmen.

Code-Snippet 4 zeigt den erstellten Code zur Odometrie-Berechnung.

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

import rospy
import math
from geometry_msgs.msg import Twist, Pose

def euler_to_quaternion(roll, pitch, yaw):
    qx = math.sin(roll/2) * math.cos(pitch/2) * math.cos(yaw/2) -
        math.cos(roll/2) * math.sin(pitch/2) * math.sin(yaw/2)
    qy = math.cos(roll/2) * math.sin(pitch/2) * math.cos(yaw/2) +
        math.sin(roll/2) * math.cos(pitch/2) * math.sin(yaw/2)
    qz = math.cos(roll/2) * math.cos(pitch/2) * math.sin(yaw/2) -
        math.sin(roll/2) * math.sin(pitch/2) * math.cos(yaw/2)
    qw = math.cos(roll/2) * math.cos(pitch/2) * math.cos(yaw/2) +
        math.sin(roll/2) * math.sin(pitch/2) * math.sin(yaw/2)
    return qx, qy, qz, qw

def cmd_vel_callback(data):
    global current_x
    global current_y
    global current_theta
    global last_time

    current_time = rospy.Time.now()
    dt = (current_time - last_time).to_sec()
    linear_speed = data.linear.x
    angular_speed = data.angular.z

    current_x += linear_speed * dt * math.cos(current_theta)
    current_y += linear_speed * dt * math.sin(current_theta)
    current_theta += angular_speed * dt

    pose = Pose()
    pose.position.x = current_x
    pose.position.y = current_y
    qx, qy, qz, qw = euler_to_quaternion(0, 0, current_theta)
    pose.orientation.x = qx
    pose.orientation.y = qy
    pose.orientation.z = qz
    pose.orientation.w = qw

    pose_pub.publish(pose)
    last_time = current_time

rospy.init_node('turtlebot_pose_calculator')
pose_pub = rospy.Publisher('my_odom', Pose, queue_size=10)
cmd_sub = rospy.Subscriber('cmd_vel', Twist, cmd_vel_callback)
current_x = 0.0
current_y = 0.0
current_theta = 0.0
last_time = rospy.Time.now()

rospy.spin()
```

Code-Snippet 4 - Skript für Odometrie-Berechnung

In den nächsten Abbildungen werden die Commands gezeigt, um das Odometrie-Skript zu testen.

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ roslaunch laser-scanner-statistics-1-1 simulation.launch
... logging to /home/vm/.ros/log/3376b34e-7c1a-11ee-adbd-0800275bd42b/roslaunch-ros-vm-2020-9
738.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.
```

Abbildung 13 - Command, um Simulation zu starten

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrn my-odometry-1-2 myOdom.py
```

Abbildung 14 - Command der Odometrie-Skript startet

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rosrn rqt_robot_steering rqt_robot_steering
```

Abbildung 15 - Command, um Roboter-Controller zu starten

```
vm@ros-vm-2020:~/ARS-ROS-Exercises$ rostopic echo /my_odom
position:
  x: 6.4215
  y: 0.0
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 1.0
---
```

Abbildung 16 - Topic 'my_odom' ausgeben

3. Schreiben Sie einen ROS-Node, der die Trajektorien der zwei Odometrievarianten in eine Datei schreibt.