Simultaneous localization and mapping mit dem Pioneer 3AT Robotor

Author: Eugen Bopp
Betreuer: Professor Dr. rer. nat. Wolfgang Ertel

31. Oktober 2011

Zusammenfassung

Ziel der Projektarbeit mit dem Titel “Simultaneous localization and mapping (SLAM) mit dem Pioneer 3AT”war es, anhand von Laserdaten einer Kinect, den Pioneer 3AT Roboter autonom in einem Gebäude fahren lassen zu können. Durch Vorgabe einer Zielposition in einem Gebiet soll der Roboter sich zur Zielposition bewegen können, ohne dabei gegen Hindernisse zu fahren. Des Weiteren soll der Roboter eine Karte seiner Umgebung erstellen. Dabei ist das Robot Operating System”(ROS) der Firma Willow Garage zu verwenden. ROS stellt Bibliotheken und diverse Programmkomponenten zur Verfügung um die Erstellung von Roboterprogrammen zu beschleunigen. Das folgende Dokument beschreibt die Installation, Konfiguration und die Benutzung des Robotersystems. Zusätzlich werden die installierten ROS Pakete kurz erläutert.

Inhaltsverzeichnis

Kapitel 1

Aufbau des Robotersystems

Das Robotersystem besteht aus zwei Computern auf denen jeweils ein Ubuntu Betriebssystem und ein ROS installiert ist. Einer der Computer wird an die Hardwarekomponenten, wie Kinect und Pioneer 3AT Roboter angeschlossen. Zusätzlich werden auf diesem Computer das SLAM Programmpaket(gmapping) sowie die Treiber für die Hardwarekomponenten(openni, p2os) installiert. Der zweite Computer wird zur Visualisierung der Umgebung, Erstellung der Karte und Definition der Zielpositionen eingesetzt. Es ist möglich alle Programmpakete auf einem einzelnen Computer einzusetzen, jedoch wäre das Setzen der Zielpositionen nicht so bequem da mindestens einer der Computer auf dem Roboter befestigt ist und dieser sich fortbewegt.

Das ROS Betriebssystem nutzt das Publish/Subscribe Modell zur Kommunikation der Prozesse und Programme untereinander. Die Daten werden in XMLRPC Format über den TCP/IP Stack gesendet, somit können Robotersysteme als verteilte Systeme realisiert werden. In Abbildung 1.1 ist der Aufbau des Robotersystems grafisch dargestellt.

1.1 Roscore

Das Roscore ist eine wichtige Komponente des ROS, es dient als Nachrichtenvermittler zwischen den ROS Programmen. Der Computer auf dem das Roscore ausgeführt wird, ist der ROS Master. Der ROS Master hat die Funktion, dass die Programme und Prozesse(nodes) sich gegenseitig finden. Sobald die Vermittlung abgeschlossen ist, kommunizieren die Programme und nodes per peer-to-peer miteinander. Der ROS Master kann per Konfigurationsdatei auf den jeweiligen Computern bestimmt werden. Wenn keine Verbindung zum ROS Master hergestellt werden kann, wird Roscore auf dem Localhost

gestartet, das bedeutet jedoch dass kein verteiltes System über mehrere Computer aufgebaut werden kann. Für weitere Informationen gibt es auf der ROS Wiki einen technischen Überblick.

1.2 Openni

Das Openni Kinect Paket beinhaltet den Treiber zur Ansteuerung der Kinect Kamera unter ROS. Die Daten des Sensors werden über das ROS Nachrichtensystem(topics) gesendet. Sensornachrichten von Openni sind z.B. RGB VGA Bildstream, sowie sog. PointCloud2 Nachrichten die eine Sammlung von N-Dimensionalen Punkten enthält. Aus diesen N-Dimensionalen Punkten wird eine Matrix generiert, die die Umgebung der Kinect mit Tiefeninformationen darstellt. In diesem Projekt wird nicht die gesamte Matrix benötigt, sondern lediglich eine 1xN Matrix. Die Kinect dient somit in diesem Anwendungsfall als Lasersensor. Das Transformieren der Nachrichten wird in ROS mit einer einfachen Konfigurationsanweisung realisiert.

1.3 P2OS

Das P2OS Paket dient zur Steuerung des Pioneer 3AT Roboters. Signale werden an den Roboter über die USB Schnittstelle geschickt. Der Roboter sendet Fehlerinformationen, Status des Akkus und sog. Odometry Nachrichten. Odometry Nachrichten repräsentieren die Drehung der Räder und werden von Gmapping benötigt um die Position des Roboters im Raum zu berechnen.

1.4 Rviz

Rviz ist ein Visualisierungswerkzeug dass verschiedene ROS Nachrichten abfangen und visuell darstellen kann, des Weiteren ist es möglich dem Roboter eine Zielposition zu schicken. Rviz kann verschiedene Konfigurationseinstellungen abspeichern und laden, somit lässt sich ein Profil für verschiedene Aufgaben definieren.

1.5 Gmapping

Gmapping ist ein Programm dass die Hindernisse, die Position des Roboters im Raum und den zu fahrenden Weg anhand der Sensordaten berechnet. Anschließend werden dem Pioneer Roboter Bewegungskommandos sendet.

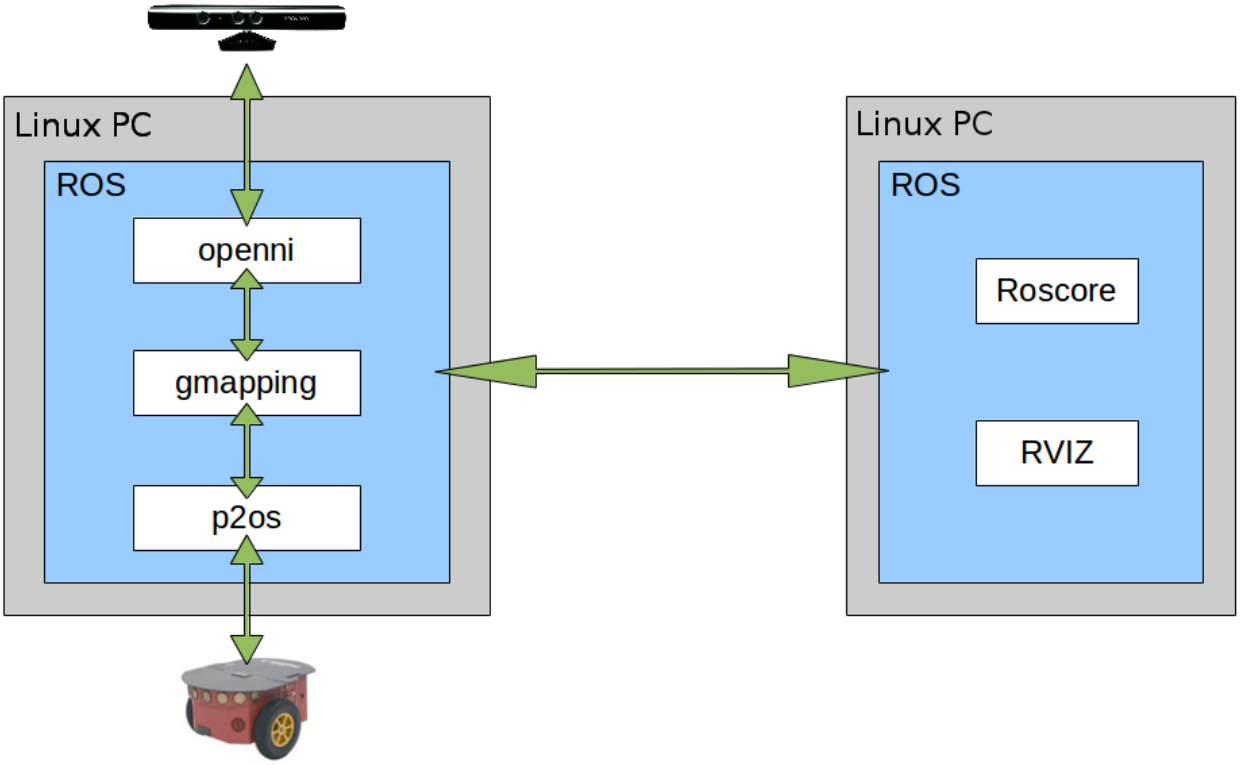

Abbildung 1.1: Aufbau des Systems

Kapitel 2

Installation

2.1 ROS

Installation des ROS auf einem Ubuntu 10.10 Betriebssystem.

Damit das APT Paketsystem von Ubuntu die ROS Pakete findet muss die ROS “apt source list”hinzugefügt werden.

```
$sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu maverick main" > /
    etc/apt/sources.list.d/ros-latest.list'
```

Als nächstes muss der öffentliche Schlüssel zur Authentifizierung der ROS Pakete importiert werden.

```
$wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Nun wird die Datenbank der Paketquellen neu geladen.

```
$sudo apt-get update
```

Zuletzt wird das ROS Betriebssystem installiert.

```
$sudo apt-get install ros-diamondback-desktop-full
```

envirement setup:

```
$echo "source /opt/ros/diamondback/setup.bash" >> ~/.bashrc
$. ~/.bashrc
```

2.2 Openni

Openni Kinect Treiber howto.

```
$sudo apt-get install ros-diamondback-openni-kinect
```

2.3 Gmapping

```
sudo apt-get install ros-diamondback-slam-gmapping
```

2.4 P2OS

Howto Installation, Konfiguration und Benutzung von P2OS.

Im Homefolder muss der Ordner ros/stacks erstellt werden.

```
$mkdir -p ros/stacks
```

Installationsdateien herunterladen:

```
$cd ~/ros/stacks/
$svn co https://usc-ros-pkg.svn.sourceforge.net/svnroot/usc-ros-pkg/trunk/
    usc-ros-pkg
```

Um Fehler mit rosmake für das p2os Paket auszuschließen, sollten folgende Pakete für die Gamepad Steuerung installiert werden.

```
$sudo apt-get install ros-diamondback-joystick-drivers ros-diamondback-
    joystick-drivers-tutorials
```

Als nächstes muss in der Datei "/opt/ros/diamondback/setup.sh"der Pfad zu anderen Softwarepaketen (stacks) erweitert werden.

```
export ROS_PACKAGE_PATH=/opt/ros/diamondback/stacks
```

Dazu wird die Zeile die den Ort der stacks angibt erweitert.

```
export ROS_PACKAGE_PATH=/opt/ros/diamondback/stacks:/home/user/ros/stacks
```

Nun muss die package/stack Liste neu geladen werden, da sonst der heruntergeladene stack von rosmake nicht gefunden wird.

```
$rospack profile && rosstack profile
```

Da der Roboter an einem usb-to-serial Adapter angeschlossen ist muss in einer Headerdatei der Parameter für den p2os Port von ttyS0 auf ttyUSB0 geändert werden.

```
$sudo vim /home/user/ros/stacks/usc-ros-pkg/p2os/p2os_driver/include/
    p2os_driver/robot_params.h
```

Zeile:

```
#define DEFAULT_P2OS_PORT "/dev/ttyS0"
#define DEFAULT_P2OS_PORT "/dev/ttyUSB0"
```

p2os kompilieren.

```
$rosmake p2os --rosdep-install
```

Kapitel 3

Konfiguration

3.1 P2os

Die nachfolgenden Konfigurationsdateien müssen auf beiden Computern identisch sein.
Hinzufügen des Computernamens zu /etc/hosts

```
$sudo vim /etc/hosts
```

Auf beiden Computern müssen diese Zeilen erweitert werden.

```
127.0.0.1    localhost
<ip des basis pc>    <name des basis pc>
<ip des pioneer pc>    <name des pioneer pc>
```

Bestimmung des ROS Masters.

```
$sudo vim /opt/ros/diamondback/setup.sh
```

Diese Zeilen müssen hinzufügt werden.

```
if [ ! "$ROS_MASTER_URI" ] ; then export ROS_MASTER_URI=http://<ip des basis
    pc>:11311 ; fi
export ROS_MASTER_URI=http://<ip des basis pc>:11311
```

¡ip des basis pc¿ ist die IP des Computers auf dem Roscore gestartet wird.

```
#!/bin/sh

export ROS_ROOT=/opt/ros/diamondback/ros
export PATH=${ROS_ROOT}/bin:${PATH}
export PYTHONPATH=${ROS_ROOT}/core/roslib/src:${PYTHONPATH}
export ROS_PACKAGE_PATH=/opt/ros/diamondback/stacks:/home/amser/ros/stacks
if [ ! "$ROS_MASTER_URI" ] ; then export ROS_MASTER_URI=http://<ip des basis
    pc>:11311 ; fi
export ROS_MASTER_URI=http://<ip des basis pc>:11311
```

3.2 Gmapping

Nachfolgend sind einige Konfigurationsdateien und Skripte aufgelistet die im Installationsorder von p2os also in “˜/ros/stacks/usc-ros-pkg/p2os/p2os_launch“ erstellt werden müssen.

3.2.1 Initiale Startpunkt

monteCarloSlam.launch ist der Initiale Startpunkt, das Skript startet das gmapping.launch und navigation.launch Skript.

```
<launch>
  <!--- start up launch files -->
  <include file="$(find p2os_launch)/gmapping.launch" />
  <include file="$(find p2os_launch)/navigation.launch" />
</launch>
```

3.2.2 Monte Carlo Localization

navigation.launch startet den Map server und übergibt diesem den Ort der gespeicherten Karte. Falls keine Karte verfügbar ist wird diese neu erstellt und kann später mit dem Mapserver gespeichert werden. Als Nächstes wird AMCL (Adaptive Monte Carlo Localization) für einen Roboter mit Differential Antrieb, in der amcl.launch konfiguriert. AMCL ist für die Lokalisierung des Roboters auf der Karte zuständig.

```
<launch>
  <master auto="start"/>

  <!-- Run the map server (you can run it here or in another terminal) -->
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
      p2os_launch)/maps/map.pgm 0.05"/>

  <!--- Run AMCL -->
  <include file="$(find p2os_launch)/amcl.launch" />

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
      output="screen">
    <rosparam file="$(find p2os_launch)/costmap_common_params.yaml" command=
        "load" ns="global_costmap" />
    <rosparam file="$(find p2os_launch)/costmap_common_params.yaml" command=
        "load" ns="local_costmap" />
    <rosparam file="$(find p2os_launch)/local_costmap_params.yaml" command="
        load" />
    <rosparam file="$(find p2os_launch)/global_costmap_params.yaml" command=
        "load" />
    <rosparam file="$(find p2os_launch)/base_local_planner_params.yaml"
        command="load" />
    <param name="base_global_planner" type="string" value="NavfnROS" />
    <param name="conservative_reset_dist" type="double" value="3.0" />
    <param name="controller_frequency" type="double" value="15.0" />
  </node>
```

```
</launch>
```

3.2.3 Slam gmapping

Das gmapping.launch Startskript dient dazu weitere Skripte und danach den slam_gmapping node mit den definierten Parametern zu starten. Für weiterführende Informationen über die Parameter ist das ROS Wiki zu empfehlen.

```
<launch>
  <include file="$(find p2os_launch)/p2os_driver.launch" />
  <include file="$(find p2os_launch)/static_transfomer.launch" />
  <include file="$(find p2os_launch)/kinect_laser.launch" />

  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" args="/scan
      " output="screen">
                <param name="map_update_interval" value="5.0"/>
                <param name="maxUrange" value="6.0"/>
                <param name="sigma" value="0.05"/>
                <param name="kernelSize" value="1"/>
                <param name="lstep" value="0.05"/>
                <param name="astep" value="0.05"/>
                <param name="iterations" value="5"/>
                <param name="lsigma" value="0.075"/>
                <param name="ogain" value="3.0"/>
                <param name="lskip" value="0"/>
                <param name="srr" value="0.01"/>
                <param name="srt" value="0.02"/>
                <param name="str" value="0.01"/>
                <param name="stt" value="0.02"/>
                <param name="linearUpdate" value="0.25"/>      <!-- param
                    name="linearUpdate" value="0.5"/-->
                <param name="angularUpdate" value="0.262"/>    <!--param name
                    ="angularUpdate" value="0.436"/-->
                <param name="temporalUpdate" value="-1.0"/>
                <param name="resampleThreshold" value="0.5"/>
                <param name="particles" value="300"/>
                <param name="xmin" value="-50.0"/>
                <param name="ymin" value="-50.0"/>
                <param name="xmax" value="50.0"/>
                <param name="ymax" value="50.0"/>
                <param name="delta" value="0.05"/>
                <param name="llsamplerange" value="0.01"/>
                <param name="llsamplestep" value="0.01"/>
                <param name="lasamplerange" value="0.005"/>
                <param name="lasamplestep" value="0.005"/>
        </node>
</launch>
```

3.2.4 Kinect als Laser

Mit der kinect_laser.launch wird der openni Kinect treiber gestartet und als Laserscanner interpretiert. Abbildung 3.1 zeigt eine grafische Darstellung der gesamten MxN Matrix, Abbildung 3.2 zeigt Lasernachricht nach Transformation.

```
<launch>
  <!-- kinect and frame ids -->
  <include file="$(find openni_camera)/launch/openni_node.launch"/>

  <!-- openni_manager -->
  <node pkg="nodelet" type="nodelet" name="openni_manager" output="screen"
      respawn="true" args="manager"/>

  <!-- throttling -->
  <node pkg="nodelet" type="nodelet" name="pointcloud_throttle" args="load
      pointcloud_to_laserscan/CloudThrottle openni_manager">
    <param name="max_rate" value="2"/>
    <remap from="cloud_in" to="/camera/depth/points"/>
    <remap from="cloud_out" to="cloud_throttled"/>
  </node>

  <!-- fake laser -->
  <node pkg="nodelet" type="nodelet" name="kinect_laser" args="load
      pointcloud_to_laserscan/CloudToScan openni_manager">
    <param name="output_frame_id" value="/openni_depth_frame"/>
    <remap from="cloud" to="cloud_throttled"/>
  </node>
</launch>
```

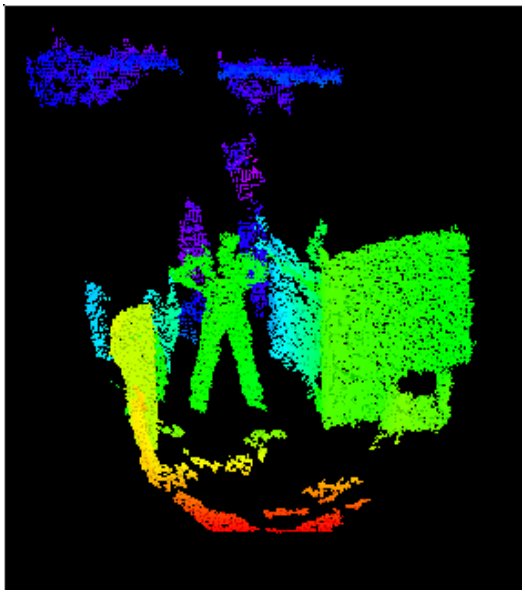

Abbildung 3.1: Kinect M x N Matrix

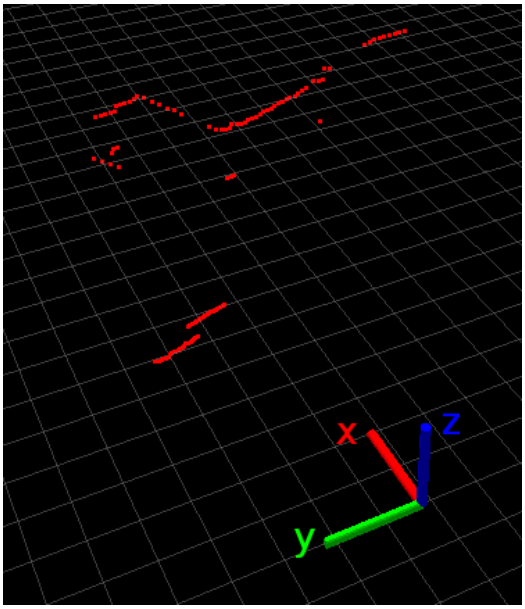

Abbildung 3.2: Kinect als Laser, 1 x N Matrix

3.2.5 P2os Treiber

Das p2os_driver.launch Skript startet den Treiber für den Pioneer Roboter. Sobald der Treiber gestartet ist, leuchtet die Kontroll LED auf dem Pioneer Roboter schneller. Nun muss der Motor des Roboters gestartet werden, dazu gibt es auf dem Roboter einen Knopf der betätigt werden muss. Nach dem starten des Motors muss die Kontrollleuchte des Roboters ein zweites mal schneller blinken.

```
1 <launch>
2   <!-- run p2os to control the pioneer -->
3   <node pkg="p2os_driver" type="p2os" name="p2os" />
4 </launch>
```

3.2.6 Transforms erstellen

erneut static_transfomer.launch dient dazu einen statischen Transform zwischen /openni_depth_frame und /base_link herzustellen:

```
1 <launch>
2   <node pkg="tf" type="static_transform_publisher" name="
        base_to_kinect_broadcaster" args="-0.115 0 0.226 0 0 0 base_link
        openni_camera 100" />
3   <!-- <node pkg="tf" type="static_transform_publisher" name="
        base_odom_broadcaster" args="0 0 0 0 0 0 /base_link /odom 100" /> -->
4 </launch>
```

3.2.7 Feintuning

In der costmap_common_params.yaml lässt sich der footprint des Roboters definieren, außerdem kann der minimale Abstand des Roboters zu Hindernissen mit dem Parameter inflation_radius eingestellt werden. Der inflation_radius sollte mindestens auf 0.6 Meter gesetzt werden da die Kinect keine Objekte näher als 0.6 Meter erkennen kann.

```
1  obstacle_range: 2.5
2  raytrace_range: 3.0
3  inflation_radius: 0.6
4
5  #---standard pioneer footprint---
6
7  #---(in meters)---
8  #footprint: [ [0.254, -0.0508], [0.1778, -0.0508], [0.1778, -0.1778],
       [-0.1905, -0.1778], [-0.254, 0], [-0.1905, 0.1778], [0.1778, 0.1778],
       [0.1778, 0.0508], [0.254, 0.0508] ]
9
10 #---pioneer AT footprint:---
11
12 #---(in meters)---
13 footprint: [ [0.3302, -0.0508], [0.254, -0.0508], [0.254, -0.254], [-0.254,
       -0.254], [-0.254, 0.254], [0.254, 0.254], [0.254, 0.0508], [0.3302,
       0.0508] ]
14
15 #transform_tolerance: 0.2
16 #map_type: costmap
17
18 observation_sources: laser_scan_sensor
19
20 expected_update_rate: 0.2}
21 laser_scan_sensor: {sensor_frame: openni_depth_frame, data_type: LaserScan,
       topic: scan, marking: true, clearing: true}
```

3.3 Rviz Konfigurationsdatei

Die für diese Aufgabenstellung erstellte Konfigurationsdatei kann unter diesem Hyperlink heruntergeladen werden. Die Datei muss die Endung .vcg haben und kann aus dem Menü von Rviz geladen werden. Für weitere Informationen und Einstellungen ist dieser ROS Wiki Eintrag hilfreich.

Kapitel 4

Anwendung

4.1 Basis PC Roscore und Rviz

Roscore starten

```
$roscore
```

Rviz starten.

```
$rosrun rviz rviz
```

4.2 Pioneer PC Slam

Auf dem Computer der mit dem Pioneer Roboter verbunden ist in den Installationsordner von p2os wechseln und das monteCarloSlam.launch Skript starten.

```
$cd ~/ros/stacks/usc-ros-pkg/p2os/p2os launch
$roslaunch monteCarloSlam.launch
```

Nun muss der Motor des Roboters gestartet werden, dazu gibt es auf dem Roboter einen Knopf der betätigt werden muss. Nach dem Starten des Motors muss die Kontrollleuchte des Roboters erneut schneller blinken.

4.3 Roboter steuern

Nachdem alle Komponenten erfolgreich gestartet sind, kann dem Roboter eine Zielposition mittels Rviz vorgegeben werden. Dafür klickt man auf den Button 2D Nav Goal, es erscheint ein Pfeil der in eine Richtung gezogen werden kann. Wenn der Roboter in dem Zielgebiet bereits eine Karte erstellt hat,

fährt dieser nach der Berechnung des Weges direkt los. Wenn die Zielposition in einem Gebiet liegt von dem der Roboter noch keine Karte erstellt hat, wird zuerst die Karte erstellt, danach ein Weg berechnet und zur Zielposition gefahren.

4.3.1 Karte speichern

Die Karte kann als Datei gespeichert werden sofern der mapserver aktiv ist. Abbildung 4.1 zeigt das Erdgeschoss des K-Gebäude der Hochschule Ravensburg/Weingarten.

```
1 rosrun map_server map_saver -f ./maps/map
```

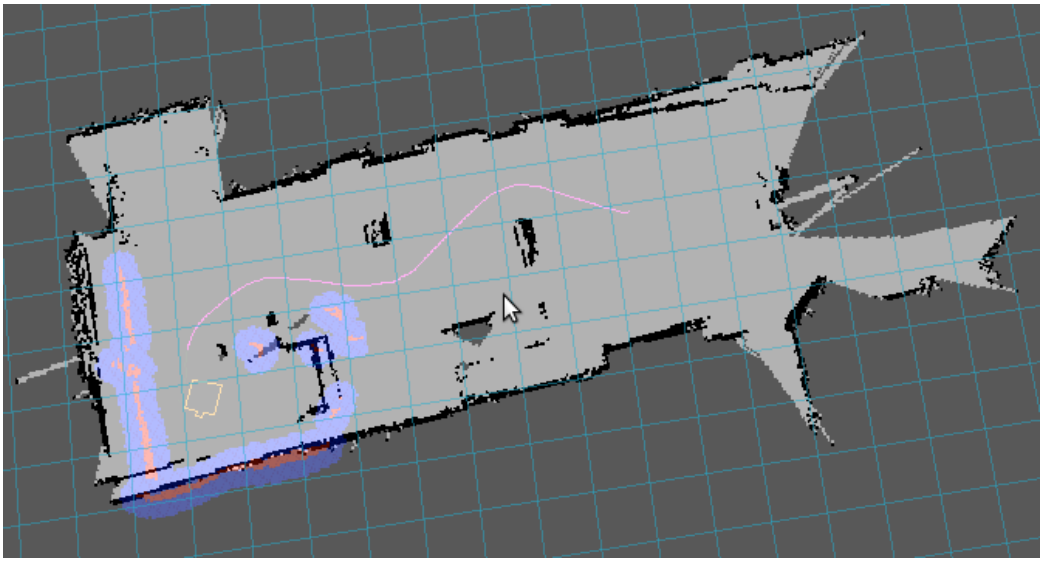

Abbildung 4.1: Karte K-Gebäude

4.4 Hilfreiche Hyperlinks

ROS Environment Variables
http://www.ros.org/wiki/ROS/EnvironmentVariables

Installation ROS Ubuntu 10.10
https://wiki.nps.edu/display/ thchung/ROS+–+Kinect+RGB-D+sensor

ROS openni Kinect Howto
http://www.ros.org/wiki/openni_kinect

ROS p2os Howto
http://robotics.usc.edu/ros/category/stacks/p2os

ROS Konzepte und Kommandoübersicht
http://www.scribd.com/doc/17726705/ROS-Concepts-and-Commandline-Overview

ROS Master Erklärung
http://www.ros.org/wiki/Master

Konfiguration Gmapping
http://www.hessmer.org/blog/category/robotics/