SLAM und Navigation mit ROS auf einem Summit XL
SLAM and Navigation using ROS on a Summit XL

Gerald Hebinck - 201623221
Alina Heid - 201622289

8. Oktober 2020

Inhaltsverzeichnis

1 Einleitung

Es wurde mit einem Summit XL (`http://wiki.ros.org/Robots/SummitXL`) von Robotnik (`https://robotnik.eu/products/mobile-robots/summit-xl-en/`) gearbeitet (Abb. 1). Zunächst wurde ROS Melodic auf dem Summit XL installiert, da der Support für Kinetic 2021 ausläuft. Anschließend wurden auf Basis der Ausarbeitungen von ERO und ARO/Flores verschiedene Änderungen vorgenommen. So wurden ein neuer Laserscanner und eine Pixhawk eingebaut. Der Summit ist jetzt in der Lage, eine Karte zu erstellen, sich auf der Karte zu orientieren und zu gegebenen Wegpunkten zu navigieren.

Abbildung 1: Summit XL von Robotnik

2 Setup

Auf dem Summit XL wurde Ubuntu 18.04 und ROS Melodic installiert. Zudem wurde das ERO-Skript zur Einrichtung des Summit XL Workspace in ROS Melodic verwendet. Zusätzlich mussten Hardwaretreiber für die Motoren und den PS4-Controller installiert werden. Hier wurde die Installationsanleitung von Robotnik für ROS Kinetic verwendet und entsprechend umgesetzt.

2.1 toolchain

Da ein Großteil des Projektes auf der echten Hardware umgesetzt wurde, war es nötig, einen Remote-Zugriff zu schaffen. Ein direkter, physischer Zugriff auf das System ist bei einem Fahrzeug nicht sinnvoll. Auf den Einsatz einer Remote-Desktop Lösung

wurde verzichtet, um das Zielsystem nicht unnötig zu belasten. Stattdessen werden Befehle auf dem Zielsystem über eine ssh-Verbindung ausgeführt und auf die Verzeichnisstruktur über sftp (Abb. 2) zugegriffen. Dazu musste auf dem Zielsystem ein ssh-Server installiert werden

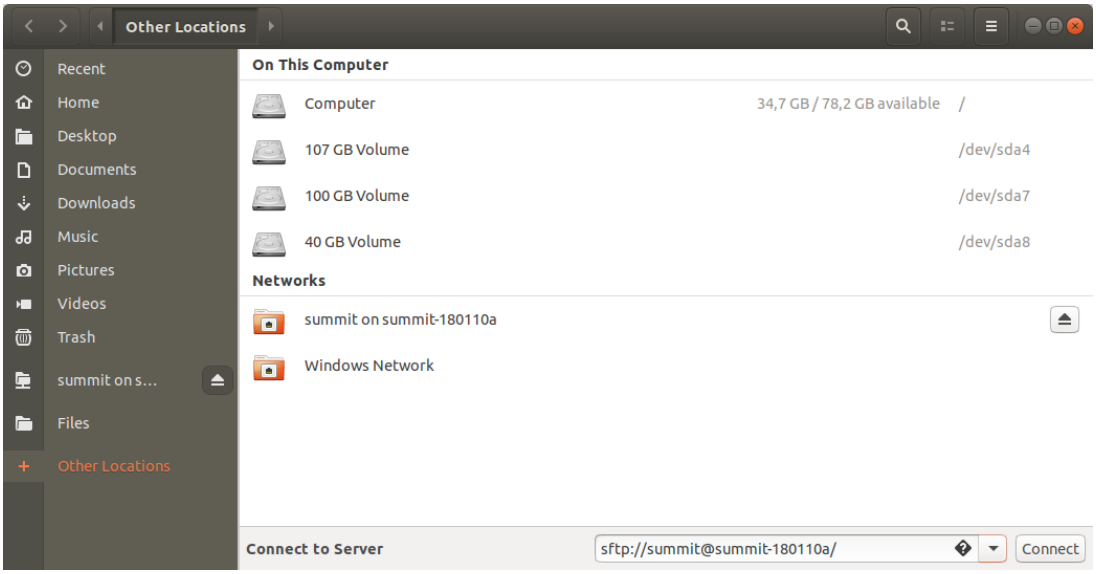


Abbildung 2: sftp Verbindung auf den Summit XL.

Durch die sftp-Verbindung ist es möglich, den catkin_ws auf dem Zielsystem direkt im Visual Studio Code zu bearbeiten, wodurch alle Vorteile der IDE zur Verfügung stehen (Abb. 3). Da es beim catkin_make zu Problemen mit einem Package kam, wurde auf catkin build umgestellt. Catkin build baut die Packages auf die selbe Weise wie catkin_make mit dem Parameter isolated. Dabei werden die Packages voneinander gekapselt. Ein Vorteil von catkin build ist dabei, dass mehrere Packages in verschiedenen Threads gebaut werden, was zu einer kürzeren Gesamtzeit führt.

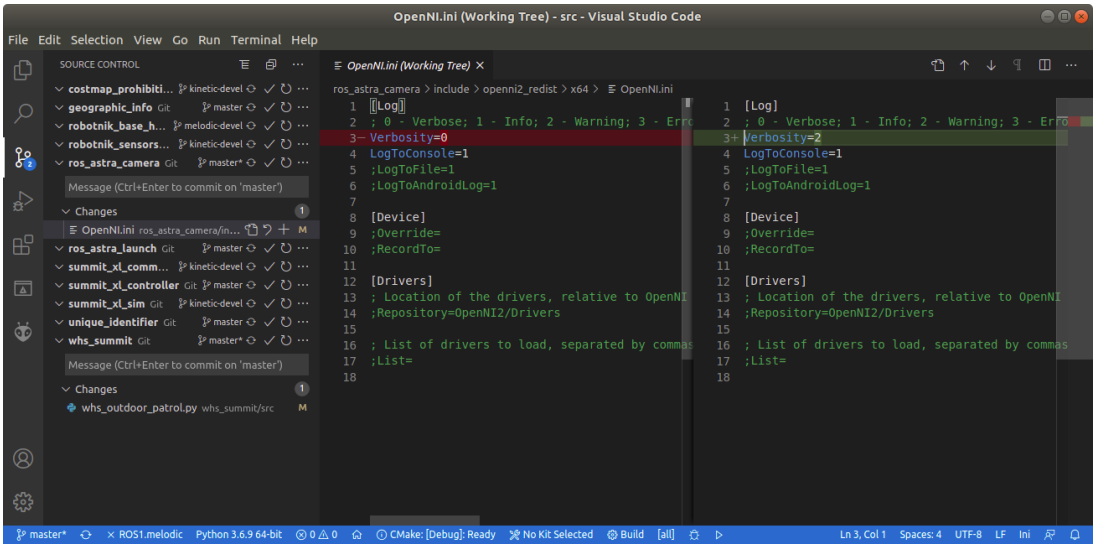


Abbildung 3: Versionskontrolle mit Visual Studio Code.

2.2 open-source

Das erforderliche Package robotnik_base_hw_lib gibt es nur als kompiliertes .deb Package. Die Quelle ist nicht offen zugänglich. Da die .deb Packages nur auf den passenden Systemen installiert werden können, ist der Einsatz von Ubuntu 20.04 mit ROS Noetic nicht möglich, da Robotnik noch keine Version für Noetic veröffentlicht hat. Das Package robotnik_base_hw_lib konnte zu Beginn des Projektes nur über die GitHub-Seite des robotnik_base_hw Packages heruntergeladen werden. Das robotnik_base_hw_lib ist von dem Package robotnik_msgs abhängig. Da das .deb Package robotnik_msgs nicht verfügbar war, konnte das robotnik_base_hw_lib zwar installiert werden, durch die fehlende Dependency blockierte es aber das apt-tool. Zum Installieren eines neuen Packages muss erst das robotnik_base_hw_lib entfernt werden, um es später wieder zu installieren. Dieses Problem wurde gelöst, indem das fehlende .deb Package aus dem ROS Package kompiliert wurde. Dazu wurde die Anleitung `https://gist.github.com/awesomebytes/196eab972a94dd8fcdd69adfe3bd1152` verwendet. Hierbei ist zu beachten, dass das Build-System dem Ziel System entsprechen sollte, damit alle nötigen Dependencies vorhanden sind.
Seit dem 18. September 2020 hat Robotnik die Packages als .deb veröffentlicht. Diese können jetzt über das apt Tool installiert werden. Das Package summit_xl_robot, in dem unter anderem die Launchfiles enthalten sind und das Package summit_xl_controller, welches für den Betrieb der realen Hardware erforderlich ist, wurden aus dem GitHub entfernt.

3 Sensoren

3.1 Hokuyo

Zur besseren Orientierung wurde ein neuer Laserscanner angebracht mit 30 m Reichweite und 270° Scanbereich (Hokuyo UTM-30LX `https://www.hokuyo-aut.jp/search/single.php?serial=169`). Dieser verwendet einen USB-Anschluss. Der vorherige Laserscanner wurde über Ethernet angesprochen. Da dieser Laserscanner größer als der vorherige ist, wurde ein neuer Halter (Abb. 4) gezeichnet, von der mechanischen Werkstatt angefertigt und angebracht. Durch die größere Reichweite gibt es mehr Überdeckungen bei aufeinander folgenden Laserscans.

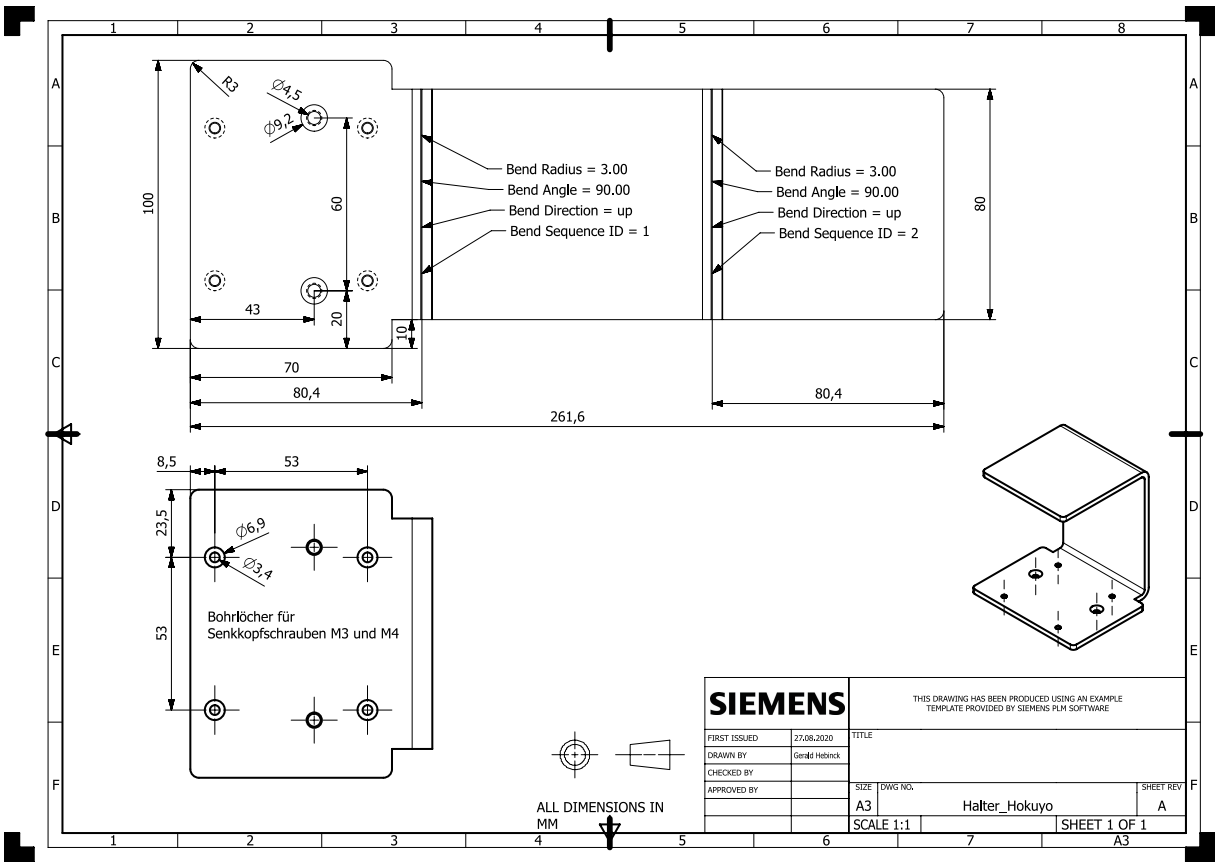
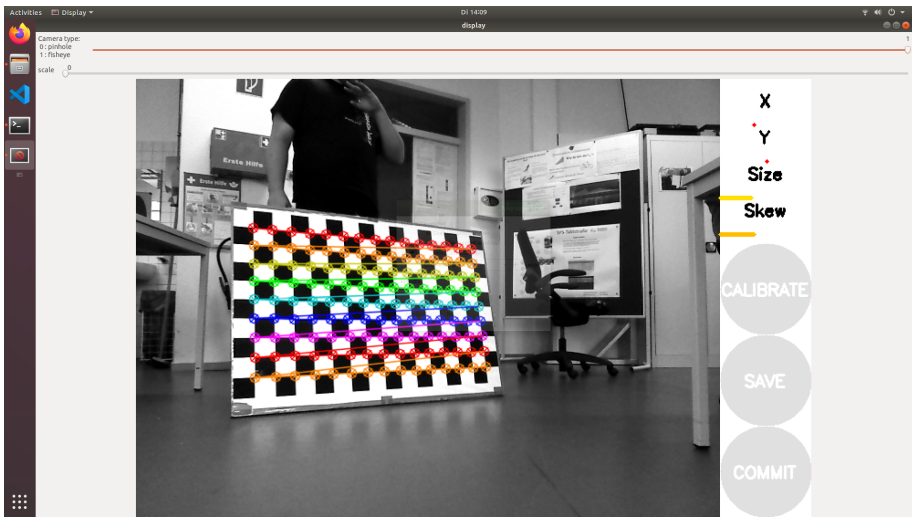


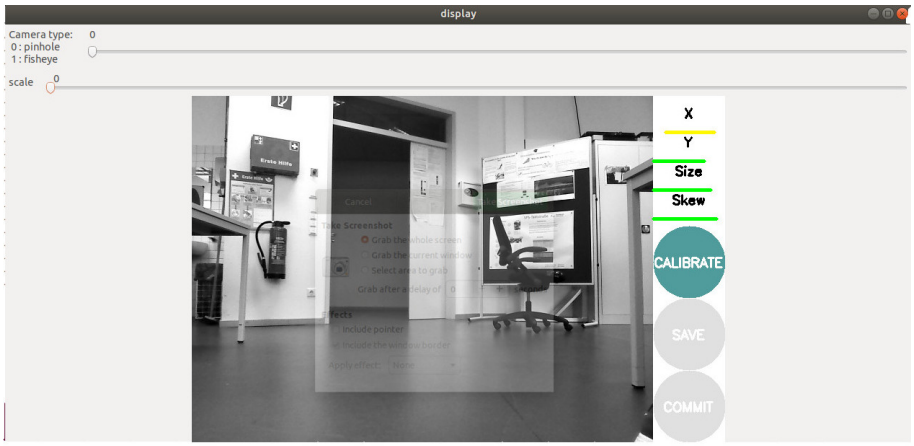
Abbildung 4: Fertigungszeichnung Halter Hokuyo.

3.2 Orbbec Astra

Die Orbbec Astra (`https://orbbec3d.com/product-astra-pro/`) wurde mithilfe eines ROS Tutorials mit camera_calibration kalibriert (`http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration`). Es wurde mithilfe eines Schachbretts (–size 15x9 –square 0.074) zunächst die rgb Kamera und anschließend die IR Kamera kalibriert. Bei jedem Schritt wird das Schachbrett still gehalten, bis das Bild im Kalibrierungsfenster hervorgehoben wird. Dabei wurde das Schachbrett in verschieden Abständen und Winkeln vor die Kamera gehalten, bis die einzelnen Orientierungen und der Knopf zum kalibrieren grün waren (Abb. 5).



(a) Die Knoten auf dem Schachbrett werden im Kalibrierungsfenster hervorgehoben.



(b) Die einzelnen Orientierungen sind ausreichend erfasst und das Kalibrierungstool ist fertig zum Kalibrieren.

Abbildung 5: Kalibrierung der Kamera mithilfe eines Schachbretts.

3.3 URDF

Das bestehende URDF musste entsprechend der Hardware-Änderung angepasst werden, da der neue Laserscanner etwas andere Maße hat (Abb. 6). Ohne diese Anpassung würde die Transformation zwischen dem base_link und dem Laserscan nicht passen. In dem Package robotnik_sensors ist bereits ein passendes Modell für den verwendeten Laserscanner hinterlegt. Das Modell wurde in das URDF eingefügt. Außerdem wurde der Schutzbügel eingefügt und die Kamera auf der Frontplatte entfernt, um eine genauere Abbildung des realen Summit zu erreichen.

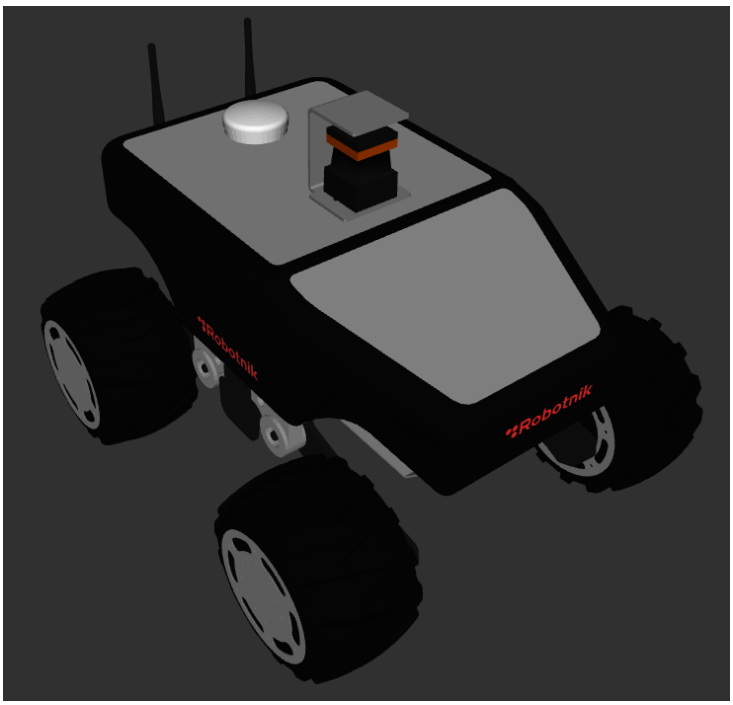


Abbildung 6: URDF des Summit XL.

Zusätzlich gab es an mehreren Stellen in den xacro-files Unstimmigkeiten in der Syntax. Es wurden xarco-tags ohne Namespace verwendet. Gemäß `https://wiki.ros.org/xacro` Abschnitt 13 - Deprecated Syntax sollte dies bei allen xacro-Dateien geändert werden. Das fehlen des Namespace führt zu Warnungen bei ROS Melodic. In ROS Noetic wird die resultierende URDF fehlerhaft. Diese Anpassungen wurden bei den Rädern und den Sensoren durchgeführt und im Github als Pullrequest bei Robotnik eingereicht (Abb. 7).

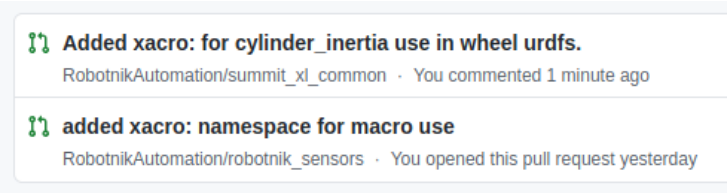


Abbildung 7: Pullrequest bei Robotnik.

3.4 Odometrie

Durch den differenziellen Antrieb des Summit XL und dem daraus resultierenden Schlupf kommt es zu Abweichungen zwischen der Odometrie und der echten Bewegung. Um die Genauigkeit der Odometrie zu erhöhen, ist im Summit eine Inertial Measurement Unit verbaut. Es handelt sich dabei um eine Pixhawk PX4. Über einen Kalmann-Filter werden die Daten mit den Daten der Odometrie verrechnet. Die ekf_localization_node ist Bestandteil des robot_localization Packages. Über die Parameter (Abb. 8) können verschiedene Sensoren hinzugefügt werden und es wird eine Odometrie mit Transformation erzeugt (`http://docs.ros.org/melodic/api/robot_localization/html/`).

```
<node pkg="robot_localization" type="ekf_localization_node"
    name="robot_localization_ekf_node_odom"
    clear_params="true">
    <param name="frequency" value="10" />
    <param name="sensor_timeout" value="0.2" />
    <param name="two_d_mode" value="true" />
    <param name="publish_tf" value="false" />
    <param name="map_frame" value="summit_xl_map" />
    <param name="odom_frame" value="summit_xl_odom2" />
    <param name="base_link_frame" value="summit_xl_base" />
    <param name="world_frame" value="summit_xl_odom2" />
    <param name="imu0" value="mavros/imu/data"/>
    <rosparam param="imu0_config">[false, false, false,
                                   false, false, true,
                                   false, false, false,
                                   false, false, true,
                                   true, false, false]</rosparam>
    <param name="odom0" value="summit_xl/robotnik_base_control/odom"/>
    <rosparam param="odom0_config">[true,  true,  false,
                                  false, false, true,
                                  true,  false, false,
                                  false, false, true,
                                  false, false, false]</rosparam>
</node>
```

Abbildung 8: Parameter der ekf_node.

Da die ekf_node die Transformation zwischen footprint und odom errechnet, diese aber auch von der robotnik_base_hw erzeugt wird, haben sich die Transformationen permanent überschrieben. Da ein Transformationsbaum als Wurzelelement die Map hat und jedes Element nur einen parent haben kann, musste ein Weg gefunden werden, die robotnik_base_hw so anzupassen, dass kein TF veröffentlicht wird (Abb. 9).

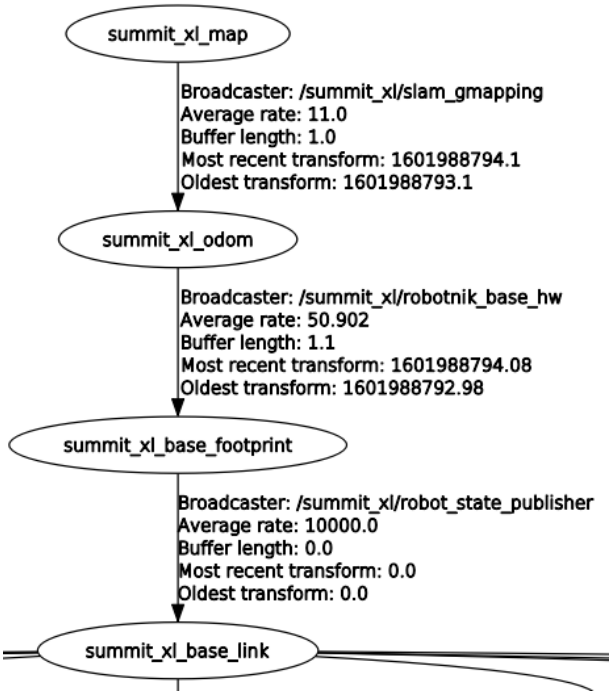


Abbildung 9: tf tree.

Im Sourcecode des Packages summit_xl_controller wird der Parameter odom_broadcast_tf ausgelesen. Wird dieser false gesetzt, veröffentlicht robotnik_base_hw keine Transformation mehr zwischen footprint und odom. Da ein entsprechender Parameter bereits im Package summit_xl_control für die Simulation verwendet wird, wurde die Launchfile entsprechend erweitert (Abb. 10).

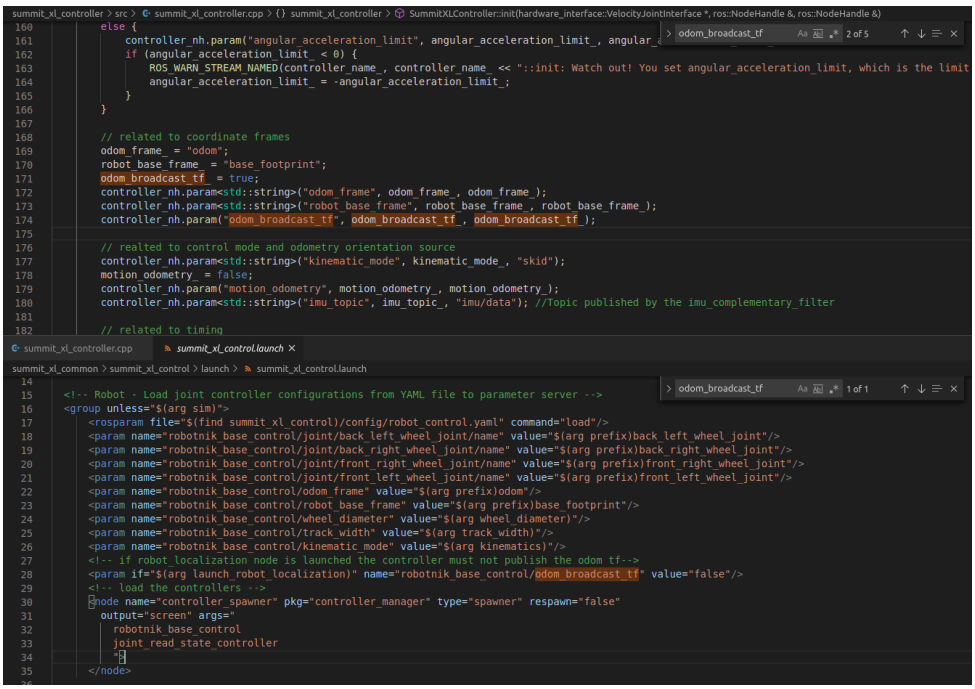


Abbildung 10: Änderung in der Launchfile zur Unterdrückung der Transformation.

4 Lokalisation und Mapping

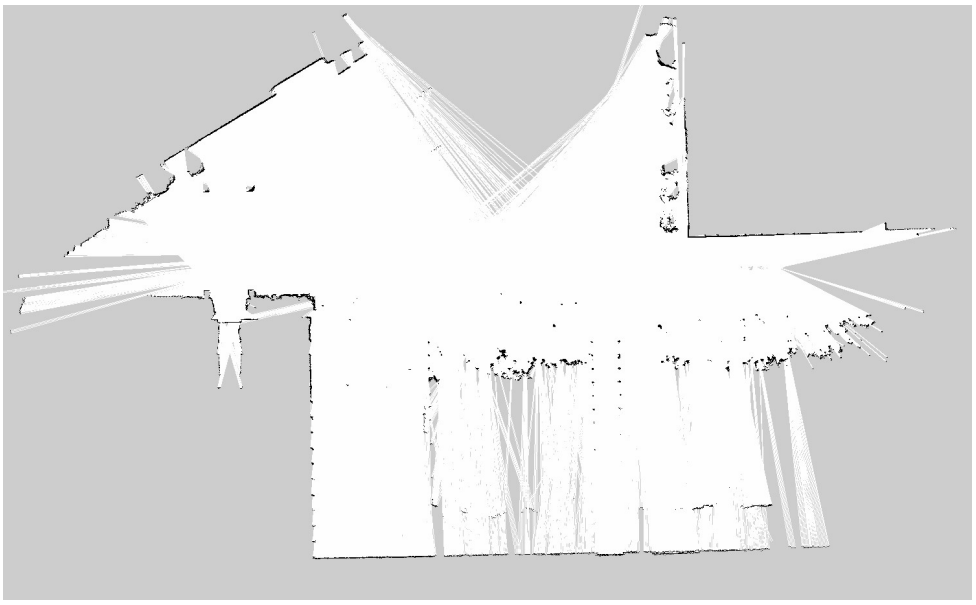
4.1 SLAM

Zur simultanen Erstellung einer Karte und der Lokalisation in selbiger kommt der gmapping Algorithmus zum Einsatz. Es werden aus Laserscan- und Odometriedaten Karten erstellt. Durch den Schlupf bedingten Odometriefehler kann es zu Fehlern beim Anfügen neuer Laserscans an die bestehende Karte kommen. Neben der Auflösung der Karte können Parameter zur Aktualisierungshäufigkeit sowie der Anzahl der Iterationen eingestellt werden. Hierbei muss ein Mittelweg gefunden werden zwischen Detail-Grad der Karte und Rechenzeit. Mit der verfügbaren Hardware konnte ein hoher Detail-Grad nur bei sehr langsamen Fahrgeschwindigkeiten umgesetzt werden.

4.2 Karten

Mit dem weiter reichenden Laserscan war es möglich, das fehlende Stück zwischen Mensa und Gebäude D auf der Karte abzubilden. Der Summit XL kann nun gleichzeitig beide Gebäude erfassen und sich so orientieren. Mit dem vorherigen Laserscanner mit geringerer Reichweite war dies nicht möglich. Der Summit XL wurde manuell sehr langsam zuerst von der Mensa zum Gebäude D (Abb. 11a) gefahren und anschließend wieder zurück zur Mensa (Abb. 11b), um die blinden Stellen aus der Karte zu entfernen, die er hinter sich gemappt hat. Die grauen Stellen in der Mitte der Karte sind nach dem zweiten Mapping nicht mehr vorhanden.

(a) Karte nach dem ersten Mapping.



(b) Karte nach dem zweiten Mapping.

Abbildung 11: Kartenerstellung zwischen der Mensa und dem Gebäude D.

Zudem wurde eine aktuelle Karte des Robotiklabors erstellt (Abb. 12). Der Summit XL wurde manuell sehr langsam durch das Labor gefahren. Diese Karte wurde mit erhöhter Auflösung und Partikeldichte erstellt.

Abbildung 12: Aktualisierte Karte des Robotiklabors.

Die Karten von der freien Fläche zwischen der Mensa und dem Gebäude D und dem Robotiklabor wurden zurecht geschnitten, skaliert und in die bereits bestehende Karte der Hochschule eingefügt (Abb. 13). Hierzu wurde das Programm GIMP verwendet.

Abbildung 13: Aktualisierte Karte der gesamten Hochschule.

5 Navigation

Die Navigation erfolgt mit dem ROS Navigation Stack. Es wird das Package move_base verwendet. Das Package ermöglicht die Auswahl eines dedizierten Planungsalgorithmus für den lokalen Weg. Im direkten Vergleich konnte sich teb gegen eband durchsetzen, da eband durch häufiges Stocken zur Neuberechnung der Fahrwege auffiel. Bei der Parametrierung von teb ist zu beachten, dass nicht verhindert werden kann, dass das Fahrzeug rückwärts fährt, da nur Gewichte verteilt werden können. Eine hohe Gewichtung für weight_kinematics_forward_drive (1000) sorgt dafür, dass das Fahrzeug regulär nur vorwärts fährt. Es kann aber zu Situationen kommen, in denen das Fahrzeug rückwärts fährt, um aus kleinen Lücken zu gelangen, in denen es nicht drehen kann. Deshalb wurde die Geschwindigkeit für das Rückwärtsfahren auf das Minimum begrenzt. Um die Genauigkeit des SLAM-Algorithmus zu erhöhen, wurde zudem die maximale Translations- und Rotationsgeschwindigkeit des Fahrzeugs begrenzt.

5.1 local_costmap

Die local_costmap wird durch das move_base Package erstellt. In der Costmap werden Hindernisse eingezeichnet und entsprechend der Parameter vom Planungsalgorithmus aufgeblasen. Dabei ist es möglich verschiedene Layer für die einzelnen Sensoren zu erzeugen. Dadurch entstehen überlappende Layer bei denen nur der entsprechende Sensor Hindernisse wieder entfernen kann. Dadurch können alle Sensoren auf maximalem Erfassungsbereich eingesetzt werden. Ein Nachteil ist dabei, dass ein bewegliches Hindernis nur aus der Costmap entfernt wird, wenn der Sensor auch die Bewegung wahrnehmen kann.

5.2 Visualisierung

Wird RViz mit der eigens erstellten Konfiguration geöffnet, sind die local_costmap, global_costmap, Sensordaten und die Pfadplanung zu sehen (Abb. 14). Das weiße Quadrat um den Summit XL stellt die local_costmap dar. In der lokalen Karte werden die von der Kamera und dem Laserscan erkannten Hindernisse eingezeichnet und aufgeblasen (dunkel grau). In der globalen Karte (global_costmap) werden die zuvor gemappten Hindernisse (schwarz) aus der Kartenerstellung aufgeblasen (helleres grau). Die farbigen Punkte stellen dar, was der Laser von der Umgebung erfasst. Die weißen kleinen Kästchen stellen dar, was die Tiefenbildkamera erkennt. Der gesamte geplante Pfad zu einem angegebenen Wegpunkt auf der globalen Karte wird mit einer blauen Linie angezeigt. Die lokale Pfadplanung in der lokalen Karte wird mit einer grünen Linie angezeigt.

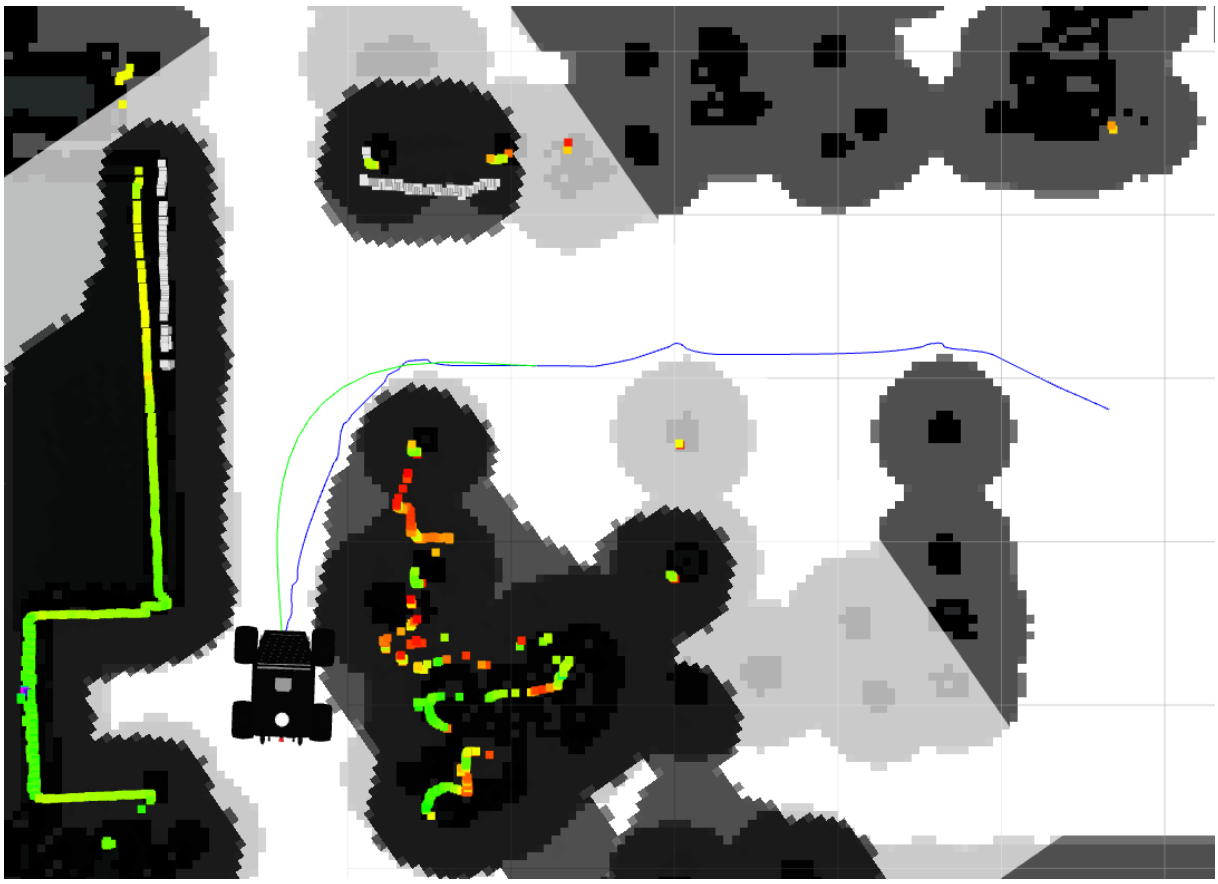


Abbildung 14: Orientierung, Navigation und Pfadplanung in RViz.

5.3 Navigation mit SLAM

Beim Start von SLAM und Navigation ist ein automatisches Fahren noch nicht möglich. Da die Karte nur den aktuellen Bereich des Laserscans umfasst, befindet sich der Mittelpunkt des Fahrzeugs im Sperrbereich der global_costmap. Es ist also nötig, das Fahrzeug manuell zu bewegen. Entweder durch Drehen, bis die Karte geschlossen ist, oder durch Vorwärtsfahren. Unmittelbar nach dem Verlassen des Sperrbereichs können mit move_base Wegpunkte angefahren werden. Um dieses Problem zu lösen, wurde der Roboter als Polygon mit den Maßen 750x640 mm vorgegeben. Dadurch war es nicht nötig, die Hindernisse aufzublasen. Es war möglich, die autonome Navigation direkt zu starten, allerdings wurde das Fahrverhalten sehr ruckartig und es kam vermehrt zu Fehlermeldungen bezüglich der Laufzeit. Deshalb wurden diese Änderungen wieder rückgängig gemacht.

6 Ergebnis

Als Ergebnis dieses Projekts wurde ein Repository auf Github angelegt (`https://github.com/alin185/whs_summit`). In dem Repository sind Installationsskripte hinterlegt, mit denen ROS und die nötigen Packages sowie Hardwaretreiber für den Summit installiert werden können. Des Weiteren enthält das Package die nötigen Konfigurations- und Kalibrationsdaten sowie die Änderungen am URDF und die erstellten Launchfiles. Um die verschiedenen Teilbereiche des Summits besser testen zu können, wurden mehrere Launchfiles erstellt.

- summit_amcl.launch
  startet das amcl und den map_server aus summit_xl_localization
- summit_bringup.launch
  startet die Hardware, den State Publisher und den PS4 Controller
- summit_move.launch
  startet move_base
- summit_sensors.launch
  startet Laser und Kamera
- summit_slam.launch
  startet slam_gmapping
- summit_imu.launch
  startet mavros und die ekf_node

Zusätzlich wurde ein Launchfile erstellt, mit dem alles gleichzeitig gestartet werden kann (summit_complete.launch). Bei parameterloser Ausführung wird der Summit mit SLAM gestartet. Zusätzlich besteht die Möglichkeit, mit dem Argument slam:=false den Summit mit AMCL zu starten und mit dem Argument map:=($mapname).yaml eine Karte aus dem Package für den map_server auszuwählen.

7 Ausblick

Durch den Einsatz eines weiteren Laserscanners am Heck des Fahrzeuges gäbe es einen 360 Grad Laserscan, der die Funktion des SLAM-Algorithmus' positiv beeinflussen würde. Außerdem würde dadurch die Karte um das Fahrzeug direkt geschlossen werden und ein Verfahren mit move_base direkt ab Start möglich. Der vorher verbaute Laserscanner könnte direkt an die Anschlüsse für Ethernet und 12V am Heck angeschlossen werden.

Die Pixhawk PX4 muss zum Einsatz der ekf_localization_node kalibriert werden. Bei den Tests ergab sich ein sehr großer Odometriedrift, da die Ausrichtung bei der Montage nicht genau genug eingestellt werden konnte. Durch die fehlerhafte Ausrichtung wird eine permanente Vorwärtsbewegung ermittelt.
Da das Fahrzeug bauartbedingt in der Lage ist, Schäden an der Umgebung anzurichten, sollte es mit einer Totmannschaltung ausgerüstet werden. Zur Zeit lässt sich die automatische Fortbewegung durch den Controller übersteuern. Hier wird aber nicht geprüft, ob der Controller überhaupt verbunden ist.
Die momentan verbauten Sensoren sind nicht in der Lage, Glasscheiben zu detektieren. Ein entsprechender Sensor an der Fahrzeugfront würde verhindern, dass eine Glastür beschädigt wird. Hier könnte ein einfacher Ultraschallsensor zum Einsatz kommen.