# HSP

**A**utonomes **L**aser **F**ahrzeug - ALF

# Lidar


Hokuyo URG-04LX

- Scanbereich von 240°
- Winkelauflösung von 0,35°
- Verwendung des **hokuyo_node** ROS Treiber
- Stellt die Daten im Topic **/scan** zur Verfügung

# Verwendetes Betriebssystem

- **Ubuntu Mate** aufgrund einer sehr großen Ubuntu Community
- Es mussten kleine Konfigurationen stattfinden um es auf einem **Raspberry 3b+** lauffähig zu bekommen (Kernel Update, WLAN-Treiber ersetzen)
- Das Framework **ROS** wurde installiert und eingerichtet
- Roscore Master, Hokuyo Treiber sowie die SLAM Map stehen sofort nach einem Boot-Vorgang zur Verfügung. Können über Systemd Services kontrolliert werden.

→ Detaillierte Informationen sind im PDF Dokument beschrieben

# SLAM

Verwendet wurde der Hector-SLAM

Dieser benötigt zum einen ein Topic ***/*scan** auf dem die Laserscans zur Verfügung stehen, zum anderen muss der Transformationsbaum zwischen den Frames

- map
- scanmatcher_frame
- base_link
- laser

korrekt sein

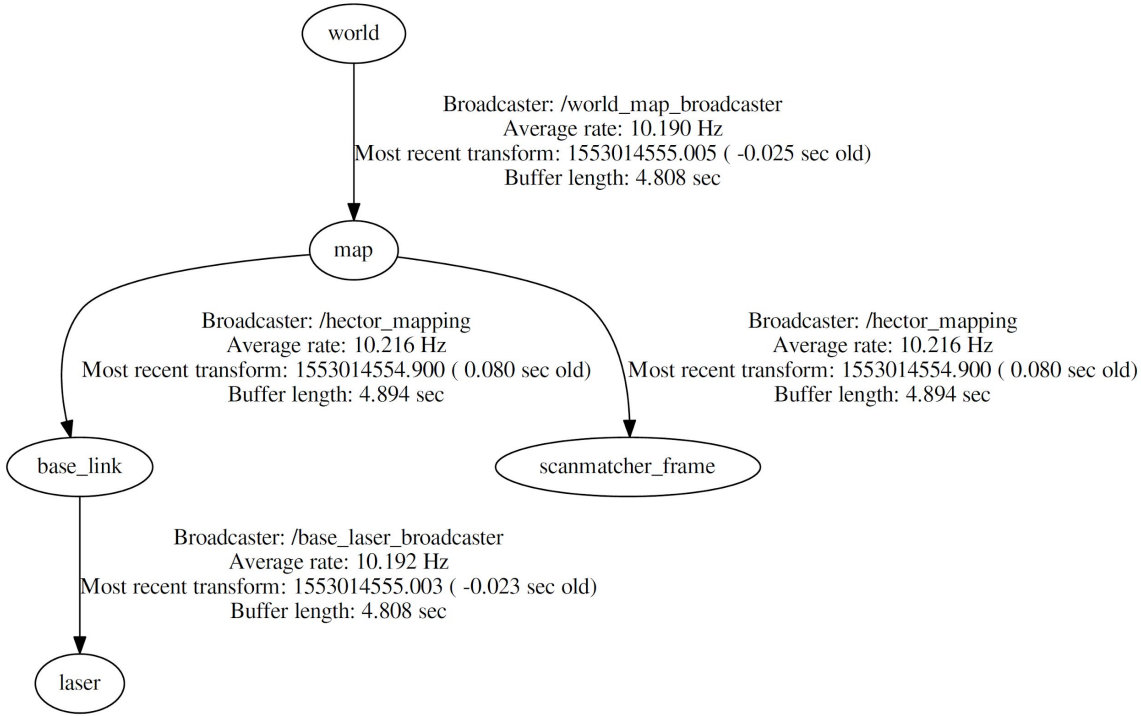
# SLAM

**Interface Modul für den SLAM:**

Wofür?

Um auch ohne das Framework ROS die erstellte Karte zu erhalten und kann somit jederzeit abgefragt werden

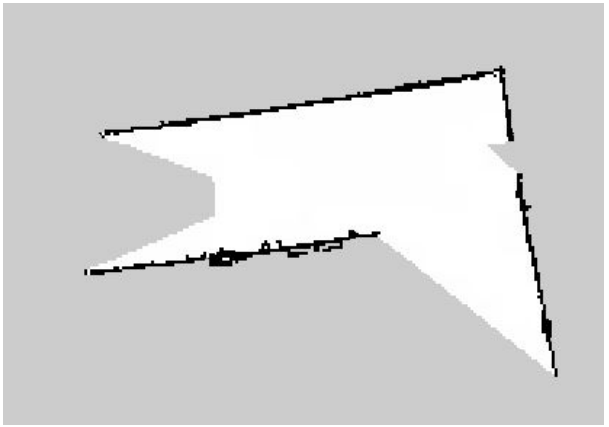
## Transformationsbaum



# Kartenarithmetik

Ziel:

- basierend auf erstellter Karte einen Weg zu allen “unbekannten” Flächen finden

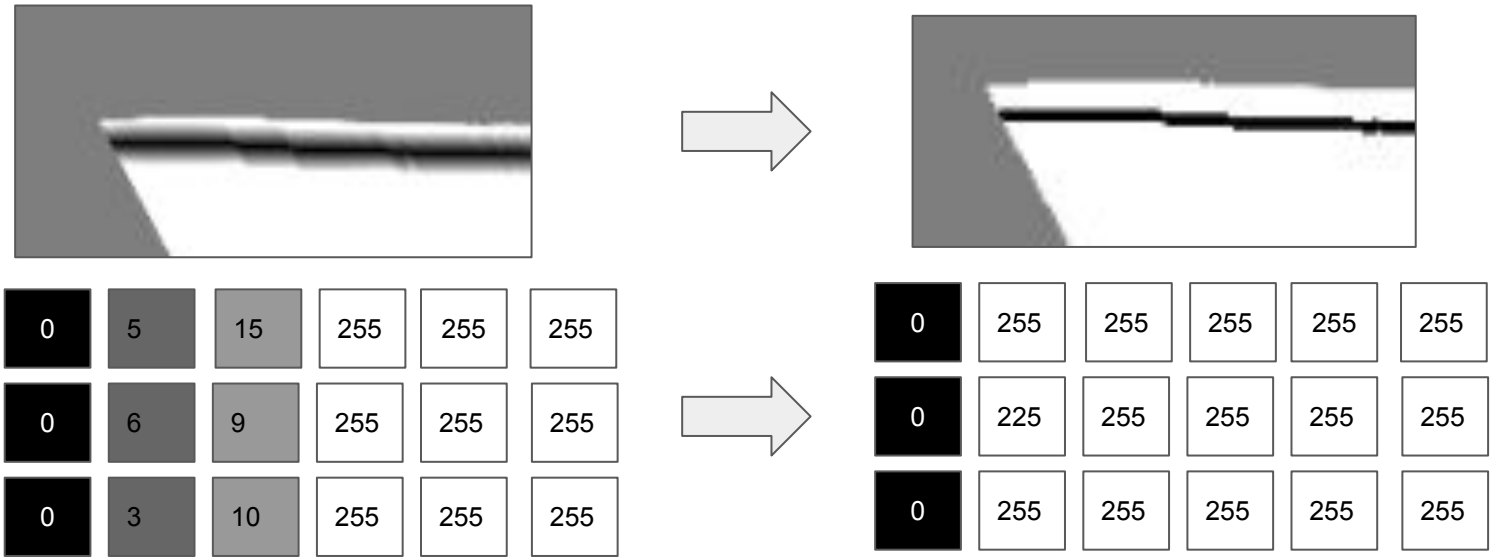


| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 255 | 255 | 127 |
| 0 | 255 | 255 | 127 |

- Idee: jeder Pixel ist Knoten; nebeneinander liegende Knoten haben Kanten

# 1) Karte bereinigen

- abhängig von SLAM
- Karte enthält evtl. “unsaubere Attribute”
- können bei späterer Weg-Suche zu Problemen führen



# 2) Gewichtung

- 90° Winkelfahrten mit Fahrzeug nicht möglich
- Fahrten nahe an Gegenständen oder Wänden kritisch

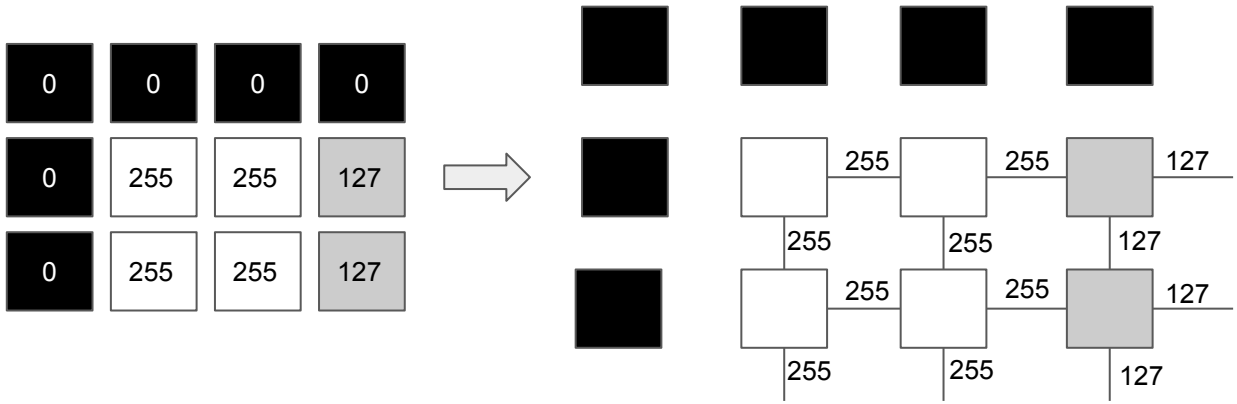
⇨ Weg sollte immer von Objekten weg führen

| | | | | | |
|---|---|---|---|---|---|
| 0 | 50 | 100 | 50 | 0 | 127 |
| 0 | 50 | 100 | 50 | 50 | 127 |
| 0 | 50 | 100 | 100 | 200 | 255 |

Lösung:
Punkte in der Nähe von Objekten: schlecht (niedriger Grauwert)
Punkte in Freiflächen: gut (hoher Grauwert)

# 3) Vorbereitung Wegfindealgorithmus

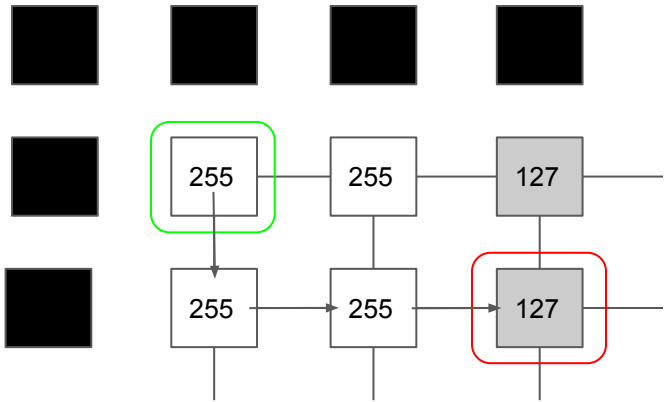
- Karte kann in Graphen gewandelt werden




- Pixel sind Knoten
- nebeneinander liegende Pixel haben Kanten
- Gewicht ist Pixel-Wert

# 4) Weg berechnen

- Dijkstra zur Wegfindung
- Start: Ego-Position
- Ziel: Flächenübergänge von Grau zu Weiß




Ausgabe:

1;1
1;2
2;2
3;2
...

# Positionskontrolle - Motoransteuerung

- Ansteuerung der Motoren und Sensoren mit echtzeitfähiger Firmware und dem Controllerboard STM32 Nucleo F334R8
- Aufgaben:
  - Signalgenerierung und Verarbeitung
  - Ansteuerung des Fahrmotors
  - Ansteuerung des Lenkservos
  - Auslesen der Ultraschallsensoren
  - Auslesen des Beschleunigungssensors per I²C (IMU MP6050)
  - Kommunikation über SPI (+ DMA) mit Raspberry Pi Board
- echtzeitfähigkeit zur Regelung der Position gewährleistet

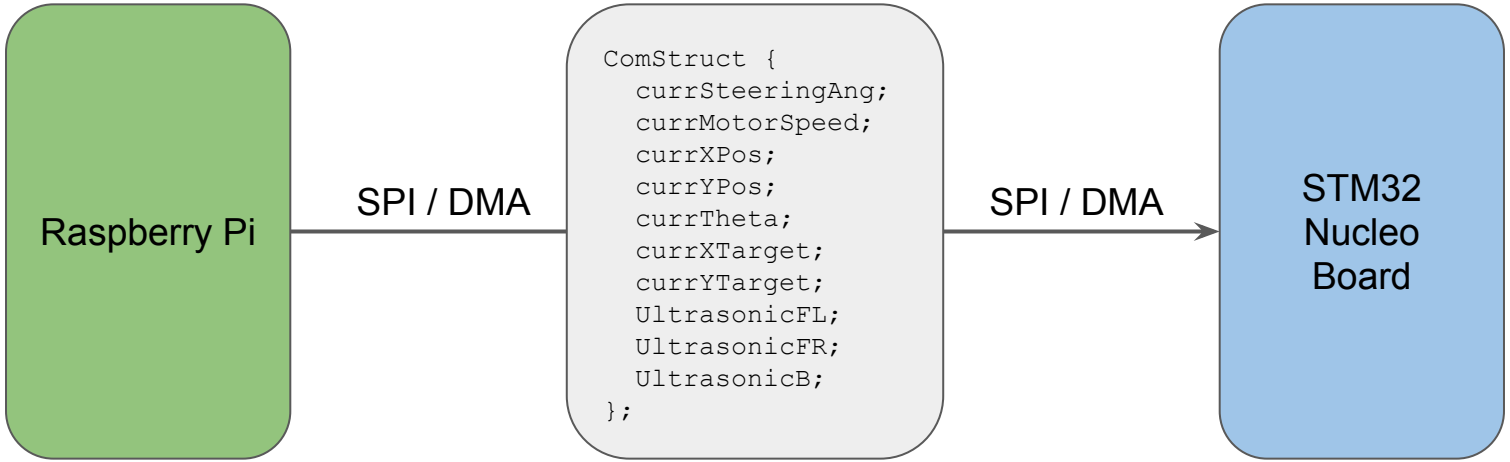
# Positionskontrolle - Motoransteuerung

Kommunikationsstruktur des STM32 Boards mit dem Raspberry Pi Board:

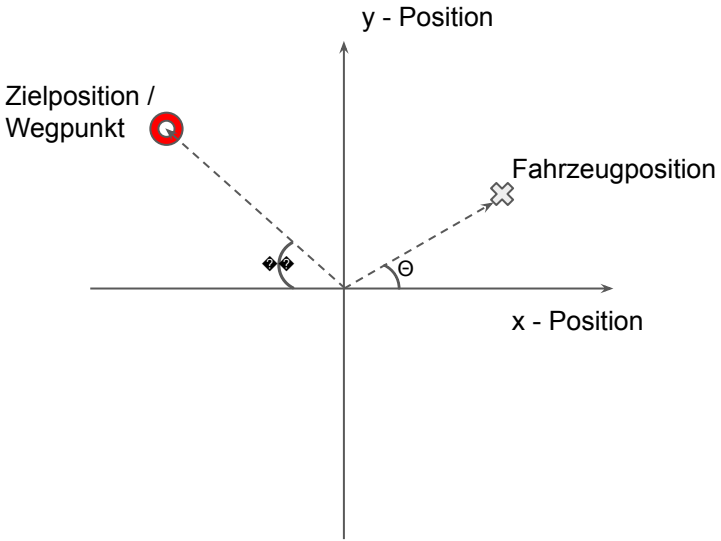
- Austausch der aktuellen Messwerte und Regelparameter über den SPI Bus
- aktuell übermittelte und verwendete Parameter:
  - Fahrmotorgeschwindigkeit
  - Lenkwinkel des Lenkservos
  - Messwerte der Ultraschallsensoren
- sämtliche Parameter werden über eine einfache C - Datenstruktur angelegt, und über SPI gebündelt übertragen / synchronisiert

# Positionskontrolle - Motoransteuerung

Kommunikationsstruktur des STM32 Boards mit dem Raspberry Pi Board:



# Positionskontrolle - Zielwinkelberechnung



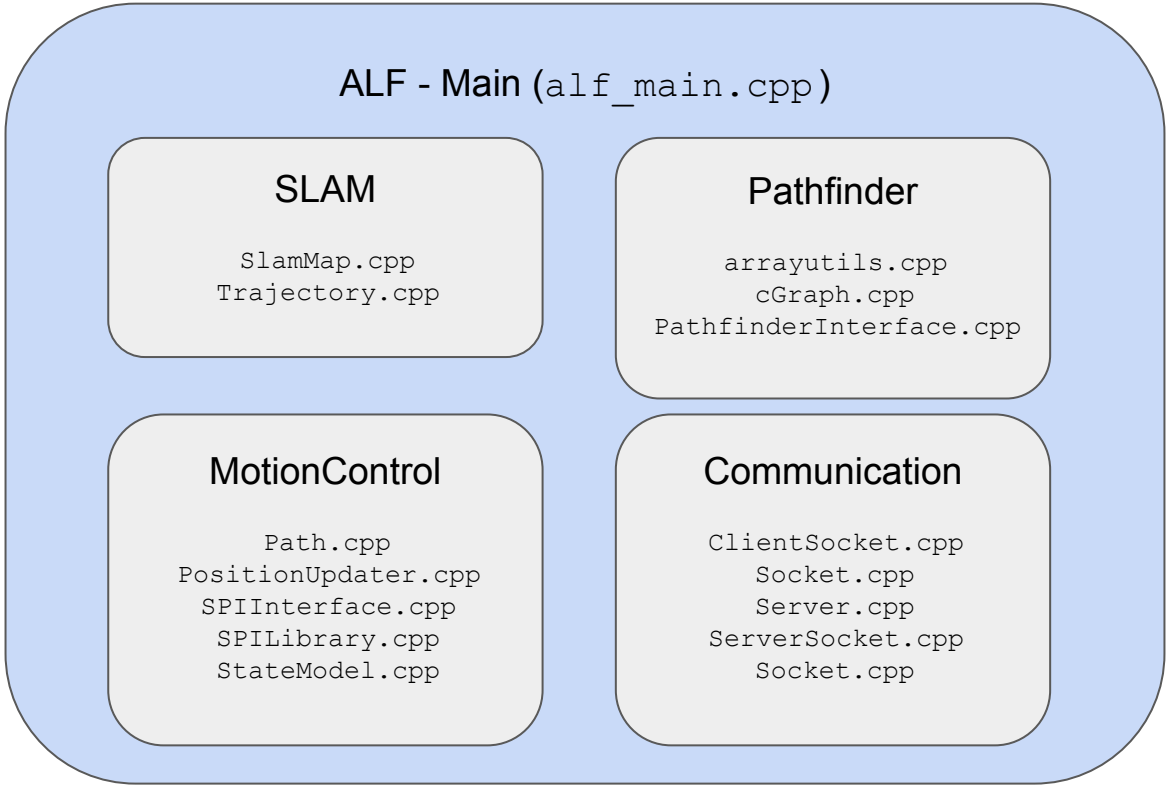

- Berechnung des aktuellen Lenkwinkels unter Berücksichtigung von:
  - aktueller Fahrzeugkoordinaten
  - aktueller Fahrzeug Ausrichtung Θ
  - Koordinaten des nächsten Zielwegpunkts $\alpha$
- Ermittlung des kürzesten Winkels zum Ziel (=> Links- oder Rechtsdrehung)
- Lenkwinkel wird per Software auf $\mp 45°$ begrenzt (maximaler Fahrzeuglenkwinkel)

# Positionskontrolle - Zustandsautomat

## Grundlage: Einfacher Zustandsautomat zur Ablaufsteuerung

- Aufgaben:
  1. Anfahren der einzelnen Zielwegpunkte (in der Raspberry Pi ALF - Softwarekomponente)
  2. Ansteuerung / Kommunikation mit dem STM32 Controllerboard (Übermittlung der Befehle)
- wesentliche Zustände:
  1. 360° Scannen der Umgebung zum Programmstart (vollständige Karte des aktuellen Standpunkts erstellen)
  2. Berechnung des Pfades
  3. Anfahren der einzelnen Wegpunkte
  4. mit Schritt 2) fortfahren

# ALF - Software - Architektur



# Offene Todos

- Ultraschallsensoren
  - Anschluss und Auslesen aller Ultraschallsensoren (momentan nur FrontLeft)
  - Berücksichtigung der Ultraschallsensoren während der Fahrt, Berücksichtigung im SLAM
- Positionsregelung optimieren
  - Regelung mit PID Controller verbinden (bereits im STM32 Board implementiert, noch nicht aktiv)
  - Rückführung der aktuellen Position an PID Regler (evtl. über Beschleunigungssensor)
  - optional: Zielkoordinaten komplett über externes STM32 Board anfahren, ohne Raspberry
- LIDAR
  - alle Messpunkte, die weiter als 8m entfernt sind, als Freifläche in die Karte einzeichnen (da LIDAR diese als unplausible Werte erkennt)

# Offene Todos

- Wegfindung
  - manchmal kann kein Pfad gefunden werden (Bugfix)
  - Performanceoptimierung
- Kartendarstellung
  - z.B. Webinterface für die Kartendarstellung erstellen (Raspberry Pi als kleinen Webserver aufsetzen)
- ROS
  - manchmal startet der ROS Node nicht korrekt, und muss manuell neugestartet werden