

# ROME2 - Praktikum 8

## Lokalisierung mit LIDAR

In diesem Praktikum wird ein einfacher LIDAR (Light Detection And Ranging) Sensor in Betrieb genommen und dessen Messungen grafisch dargestellt. Anschliessend soll ein Filter implementiert werden, um mit dem LIDAR die Position von Landmarken relativ zum Sensor (resp. zum Roboter) zu detektieren.

Der verwendete LIDAR ist ein 360° Scanner der Firma Slamtec. Die Distanzmessung basiert nicht wie bei den meisten LIDAR auf dem time-of-flight Prinzip eines Laserstrahls, sondern auf einer Triangulation: Ein Infrarot-Laser erzeugt auf entfernten Objekten einen Punkt, der mit einer seitlich versetzten Kamera detektiert wird. Aus der Position des Punktes im Kamerabild kann dann die Distanz bestimmt werden.

Der LIDAR führt alle 0.5 ms eine Distanzmessung durch und übermittelt diese Distanz, sowie den dazugehörigen Winkel via serieller Schnittstelle an den Mikrokontroller. Die Klasse `LIDAR` der Robotersteuerung empfängt diese Messungen und speichert sie in einem Array ab.

### 1 Inbetriebnahme des LIDAR und der Steuerungssoftware

Die erste Aufgabe besteht darin, den LIDAR und die gegebene Steuerungssoftware in Betrieb zu nehmen. Am LIDAR ist ein Adapter angebracht, um ihn direkt auf einen Mikrokontroller mit Arduino Uno kompatiblen Pin Headers aufzusetzen. Mit dem NUCLEO-F767ZI Mikrokontroller muss der LIDAR auf den vordersten Pins der äusseren beiden Pinreihen aufgesetzt werden, also soweit in Richtung des Debuggers mit der USB Schnittstelle zum Programmieren des Mikrokontrollers wie möglich.

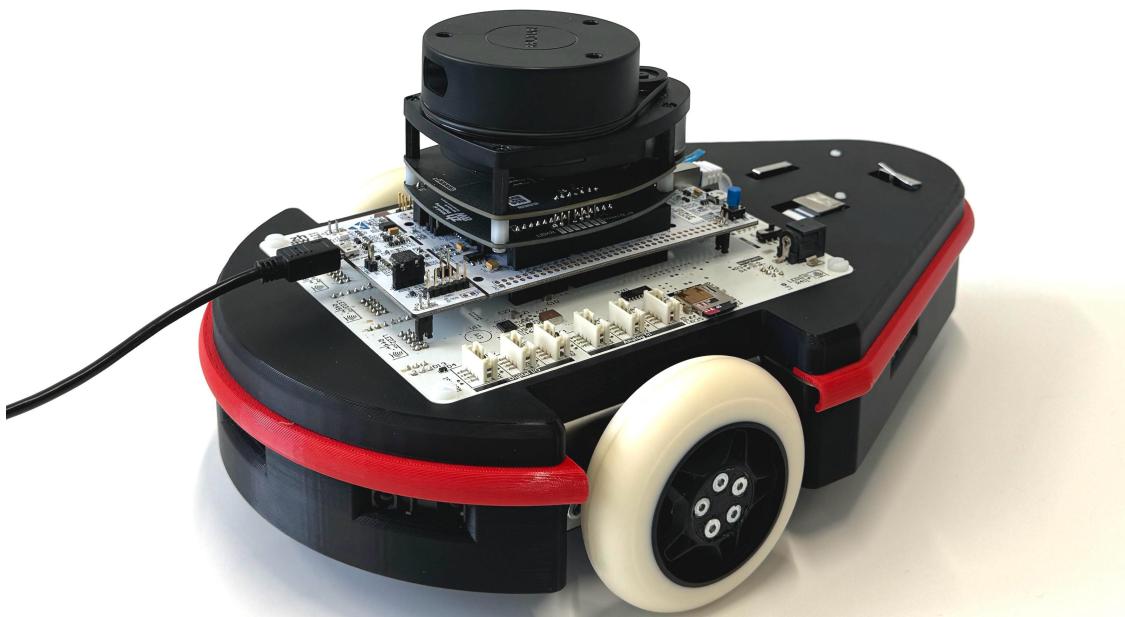


Bild 1.1: Slamtec LIDAR auf dem ROME2 Roboter

Die Steuerungssoftware beinhaltet eine Treiberklasse LIDAR. Ein Objekt dieser Klasse wird im Hauptprogramm, also in der Klasse Main wie folgt kreiert und gestartet:

```
PwmOut pwm(PE_9);
pwm.period(0.00005f);
pwm.write(0.5f); // 50% duty-cycle

ThisThread::sleep_for(500ms); // time for spin-up of motor

UnbufferedSerial* serial = new UnbufferedSerial(PG_14, PG_9);
LIDAR* lidar = new LIDAR(*serial);
```

Zum Betrieb dieses LIDARs wird ein PWM Ausgang benötigt. Mit diesem PWM Ausgang wird der Leistungstreiber des Antriebsmotors des LIDAR gesteuert. Mit der Duty-Cycle dieses PWM Ausgangs kann die Drehzahl des Scanners vorgegeben werden. Bei höheren Drehzahlen ist die Scanrate höher, aber dafür die Winkelauflösung der Distanzmessungen geringer. Im Beispiel oben wird die Duty-Cycle dieses PWM Ausgangs auf 50% gesetzt. Wählen Sie für Ihren LIDAR eine Duty-Cycle zwischen ca. 40% und 60%, so dass eine Scanrate von ca. 3 Scans/Sekunde erreicht wird, resp. eine Winkelauflösung der Distanzmessungen von ca. 0.5°.

Die Periode dieses PWM Signales wird wie bei den beiden Antriebsmotoren des ROME2 Roboters auch auf 50 µs gesetzt (dies entspricht einer PWM Frequenz von 20 kHz).

Der LIDAR Treiber selber benötigt eine serielle Schnittstelle für die Kommunikation mit dem Sensor, insbesondere für das Empfangen der Distanzmessungen. Diese serielle Schnittstelle wird im Konstruktor der LIDAR Klasse konfiguriert. Der Konstruktor sendet auch Steuerbefehle an den LIDAR, um den Scavorgang zu starten. Im Destruktor der LIDAR Klasse werden Steuerbefehle gesendet, um den Scavorgang wieder zu stoppen. Das Empfangen der Messungen ist in der Interrupt-Service-Routine receive() der LIDAR Klasse implementiert.

Der LIDAR Treiber speichert die empfangenen Distanzmessungen in einem Array mit 360 Werten. Das heisst, für jedes Winkelgrad wird die Distanz gespeichert. Dafür ist es wichtig, dass der LIDAR mindestens eine Distanzmessung pro Winkelgrad ausführt. Das bedeutet, dass die Scanrate des Sensors nicht zu hoch sein darf.

Um die Distanzmessungen auszulesen bietet die LIDAR Klasse die folgende Methode:

```
deque<Point> scan = lidar.getScan();
```

Damit erhält man eine Liste von 360 Punkten, die vom LIDAR gemessen wurden. Die Liste ist ein Vektor der deque Klasse von C++, und die einzelnen Elemente in dieser Liste sind Objekte von der Klasse Point. Dies ist eine Hilfsklasse, welche Punkte sowohl in Polarkoordinaten wie auch in Kartesischen Koordinaten abspeichert. Zudem bietet diese Hilfsklasse Point noch Methoden an, um die Distanz des Punktes zum Ursprung und diejenige zu einem anderen Punkt zu berechnen.

Um die LIDAR Scans zu visualisieren beinhaltet diese Steuerungssoftware auch einen Webserver, der eine statische Webseite liefert, welche anschliessend periodisch ein Skript aufruft, welches die Scans in eine XML Seite formatiert und an den Webbrower sendet. Dieser stellt die gemessenen Punkte des LIDAR dann grafisch dar. Kopieren Sie dazu das Dokument "lidar.html", welches im Verzeichnis "html" des Praktikums 8 liegt auf die Mikro-SD Karte der Robotersteuerung.

Um diesen Webserver zu nutzen müssen Sie auf einem Laptop, Tablet oder Smartphone zuerst eine WLAN Verbindung mit dem integrierten Access Point des Roboters aufzubauen. Der Access Point wird beim Starten der Software auf dem Mikrokontroller eingeschaltet und ist dann über einen Namen (SSID) sichtbar, der auf der Rückseite des Roboters aufgeklebt ist.

Der TCP/IP Stack der Ethernet Schnittstelle des Mikrokontrollers wird in der Main Klasse kreiert und konfiguriert. Die Einstellungen können mit der Methode `set_network()` vom Objekt `EthernetInterface` geändert werden. Die Einstellungen der gegebenen Software lauten:

IP Adresse: 192.168.0.10  
Netmask: 255.255.255.0  
Gateway Adresse: 192.168.0.1

Wenn Sie die Software kompiliert und auf den Mikrokontroller geladen haben, und wenn Sie das WLAN Ihres Roboters ausgewählt haben, können Sie einen Browser öffnen und die IP-Adresse Ihres Mikrokontrollers sowie die Webseite "lidar.html" eintippen. Dann sollten Sie im Browser einen LIDAR Scan ähnlich wie folgt sehen:

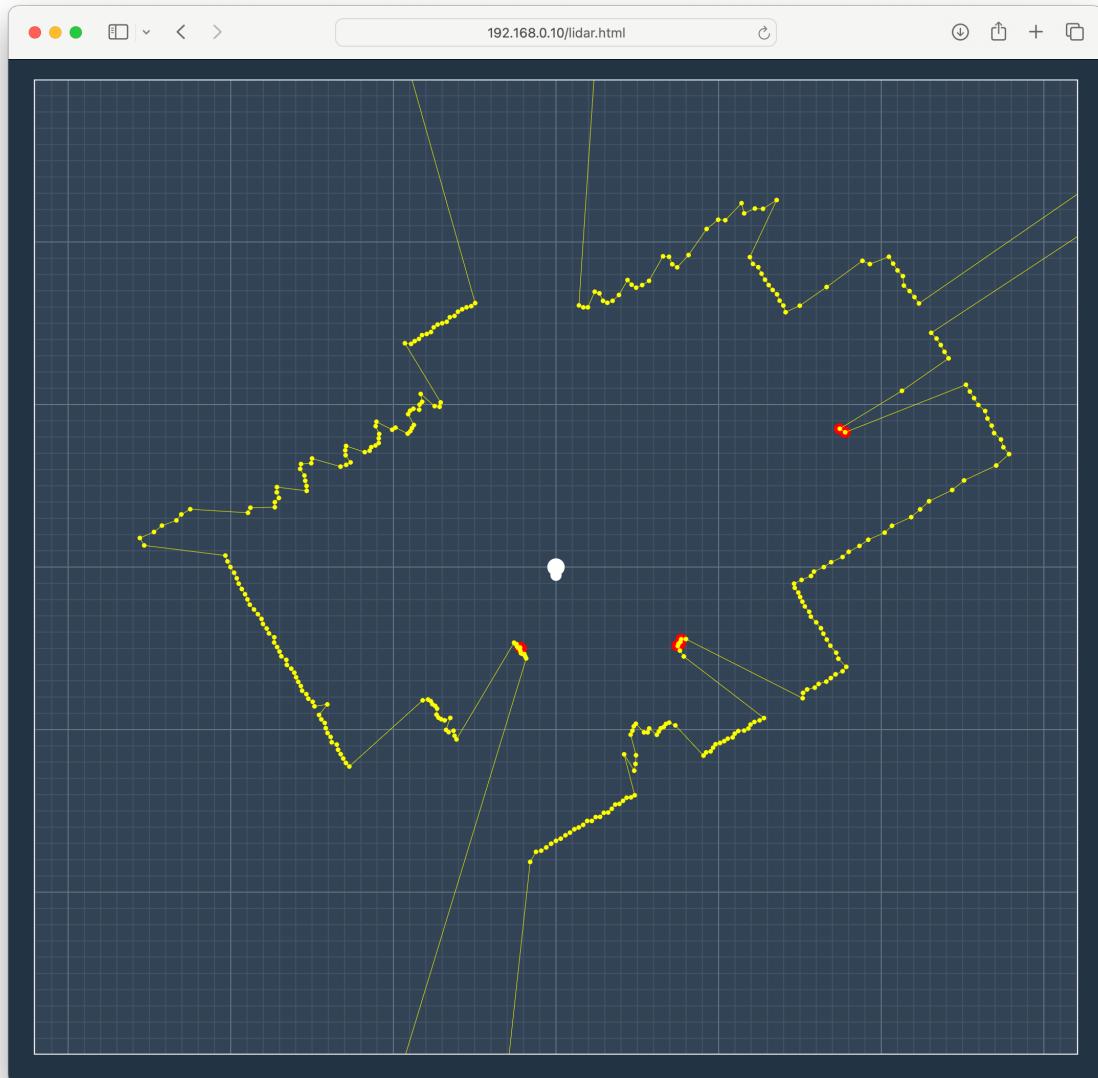


Bild 1.2: Scan des LIDAR in einem Raum, inkl. detektierten Landmarken

## 2 Algorithmus zum Finden von Landmarken

Dieser LIDAR kann zwar Distanzen zu Objekten in der Umgebung messen, er kann aber nicht den Reflexionsgrad der Objekte detektieren, wie dies viele andere LIDAR tun können. Damit ist es nicht

möglich, an ausgewählten Stellen Reflektoren anzubringen, um deren detektierte Position für die Lokalisierung eines mobilen Roboters zu verwenden.

Stattdessen müssen wir andere, im Scan eindeutig identifizierbare Strukturen finden, die wir für die Lokalisierung verwenden können. Eine einfache Möglichkeit ist nun, in einer Umgebung, in der sich der mobile Roboter bewegt Rohre auf den Boden zu stellen (z.B. in der Grösse von Getränkeflaschen), und zwar an Positionen, die bekannt sind. Dann muss man nur noch die Positionen der Rohre im Scan des LIDAR identifizieren, um anschliessend damit die Lage des Roboters korrigieren zu können.

Die LIDAR Klasse bietet bereits eine Methode an, mit der eine Liste von Punkten, die auf solchen Rohren liegen, abgefragt werden kann:

```
deque<Point> beacons = lidar.getBeacons();
```

Für dieses Praktikum muss diese Methode aber noch implementiert werden.

Für diese Methode soll also ein Algorithmus entwickelt werden, der aus einer gegebenen Liste von Punkten eines Scans diejenigen heraussucht, die auf vereinzelt aufgestellten Rohren liegen, und diese Punkte von Landmarken wieder in eine Liste speichert, die dann von der Methode zurückgegeben wird.

Um nun einen Punkt des Scans als mögliche Landmarke zu akzeptieren sollen die folgenden Bedingungen erfüllt sein:

- Der Punkt darf höchstens 3.0 Meter entfernt sein.
- Andere Punkte müssen entweder weniger als 0.1 Meter von diesem Punkt entfernt sein (dann befinden sie sich auf demselben Rohr), oder sie müssen weiter als 0.5 Meter von diesem Punkt entfernt sein (dann befinden sie sich ganz woanders).
- Es muss mindestens noch ein zweiter Punkt weniger als 0.1 Meter entfernt sein (das Rohr sollte also von mindestens 2 Punkten detektiert worden sein, einzelne Messungen werden verworfen).

Punkte, welche diese Bedingungen erfüllen, sollen in einer neuen Liste gespeichert und von der Methode getBeacons () zurückgegeben werden.

Sie können Ihren Algorithmus mit der Visualisierung im Webbrowser direkt prüfen. Die Methode getBeacons () wird nämlich bereits vom Skript des Webservers aufgerufen, und detektierte Landmarken im Browser als rote Punkte grafisch dargestellt (siehe Bild 1.2).

### 3 Optimierung der Performance des Algorithmus

Es ist oft nicht sehr schwer, einen Algorithmus zu programmieren, der eine gestellte Aufgabe löst. Aber oft sind dann die ersten Versionen solcher Algorithmen nicht sehr effizient, das heisst, sie brauchen verhältnismässig viel Zeit für deren Berechnung.

Messen Sie zuerst die Zeit, die für den Aufruf der Methode getBeacons () des LIDAR Objektes benötigt wird. Dazu steht in der Mbed Treiberbibliothek die Klasse Timer zur Verfügung. Diese Klasse kann wie folgt verwendet werden, um Zeitmessungen durchzuführen:

```
Timer timer; // kreiert ein Timer Objekt
timer.start(); // startet den Timer
...
timer.reset(); // setzt den Timer auf Null zurueck
lidar->getBeacons(); // Aufruf der Methode
int duration = timer.read_us(); // Lesen der Zeittdauer in Mikrosekunden
```

Diese Zeitmessung kann z.B. periodisch im Endlos-Loop des Hauptprogrammes "Main.cpp" durchgeführt und mit dem Befehl `printf()` in einer Konsole ausgegeben werden.

Eine einfache Optimierung des Algorithmus kann sein, wenn für die Berechnung von Distanzen von Punkten nicht die Methoden `distance()` der `Point` Klasse verwendet werden, sondern die Methoden `manhattanDistance()`. Diese Methoden verwenden zwar nur eine Approximation der Distanzberechnung, sind dafür aber um Faktoren schneller.