Leistungskurs C++

Einführung

Martin Gottwald, Stefan Röhrl und
Martin Knopp

16. Oktober 2018

Das Team

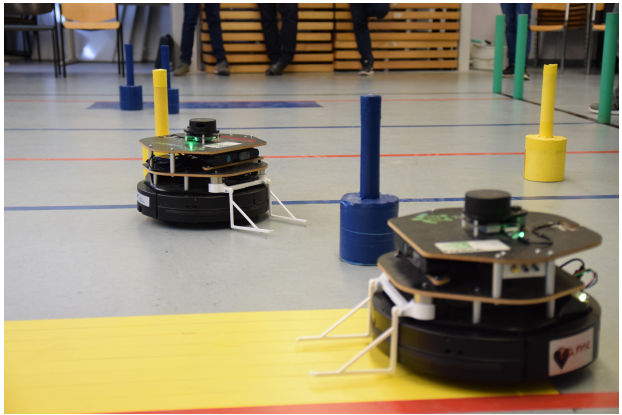
Martin Gottwald
Martin.Gottwald@tum.de
Büro Z943

Stefan Röhrl
Stefan.Roehrl@tum.de
Büro Z947

Martin Knopp
Martin.Knopp@tum.de
Büro Z934

Was machen wir im Leistungskurs?

- Roboter spielen „Hockey“
- Versionsverwaltung mit git
- Buildsystem cmake
- Ubuntu Linux (Bash!)
- Robot Operating System
- C++ ist Pflicht (kein „rospy“)
- Der ROS „navigation stack“ ist nicht erlaubt
  $\rightarrow$ es ist euer Projekt



Was machen wir im Leistungskurs?

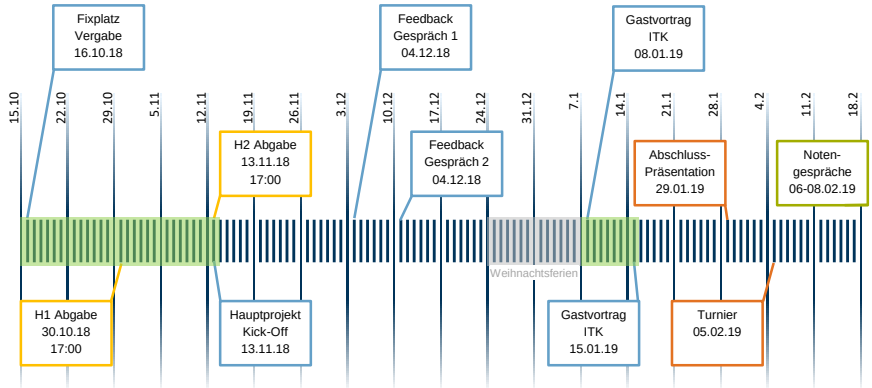
Videodemonstration

Bewertung

- Hausaufgaben: 30%
  - 2 Stück
  - 2 Wochen Bearbeitungszeit
  - Einzelarbeit
  - Diskussion in der Gruppe erwünscht!
  - Aber: kein Copy-n-Paste!
  - H1 im laufe des Tages verfügbar
- Abschlusspräsentation: 20%
  - am Ende des Hauptprojekts
  - Gruppenvortrag
  - verwendete Ansätze, Ideen, Probleme, …
- Hauptprojekt: 50%
  - Spielt der Roboter im Rahmen der Regeln?
  - Erfüllt der Roboter die minimalen Anforderungen?
  - Ist der Code wartbar?
  - Sind alle Gruppenmitglieder beteiligt ($\rightarrow$ git-Aktivität)?
  - …
- Was nicht zählt:
  - Performance, Speicherverbrauch, etc.
  - Ausgang des Turniers: 1. Platz: 1.0, 2. Platz: 1.3, …

Zeitplan (cf. Moodle)

Leistungskurs C++ WS 18/19



Gruppeneinteilung

Nicht die Folie anschauen sondern einen Gruppenplatz ergattern!

Nützliches

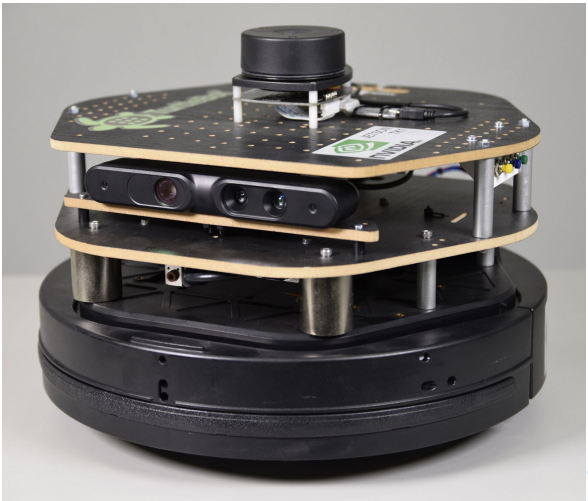
- LDV-Webseite zum Kurs http://www.ldv.ei.tum.de/lehre/leistungskurs-c/
  - Hausaufgaben
  - Vorlesungsfolien
  - Weitere Downloads (Virtuelle Maschine, etc.)
  - Buch „A Gentle Introduction to ROS“
- ROS-Wiki → Nutzt es, vorallem die Tutorials!
- Tutoren, die euch vor Ort bei Problemen weiterhelfen
- Tutoren findet ihr im Praktikumsraum -1981 (Zeitplan auf der LDV Seite)
- Schickt eure Fragen an cpp-tutor@ldv.ei.tum.de

Eure Verpflichtungen

- Meldet Probleme sobald sie auftauchen!
  - Wenn euer PC nicht will, geht zu den Tutoren!
  - Stimmt die Chemie in eurer Gruppe nicht dann informiert uns!
  - Teile werden brechen, wir haben 3D-Drucker und endlosen Ersatz!
- Verwendung von git
  - Wir müssen am Ende die Arbeitsverteilung bewerten können
  - Wenn ihr keine commits im Hauptprojekt habt ist das „verdächtig“
  - git ist Pflicht für die Abgabe der Hausaufgaben (später mehr)
  - Die Verwendung von git ist Teil der Note!
- Überprüft regelmäßig den Moodlekurs und lest euere Emails
  - Wir akzeptieren nicht als Ausrede ”Das habe ich nicht mitbekommen”
  - Wir gehen davon aus, dass Emails via Moodle / TUMonline von euch gelesen werden

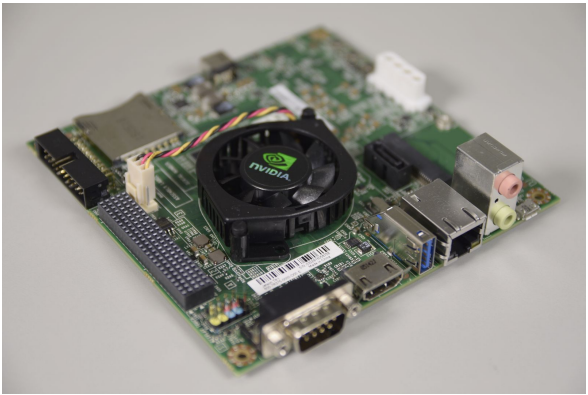
Die Hardware: Turtlebot

- Wäre fast ein Staubsauger geworden
- Nvidia Jetson TK1
- Asus Xtion Pro Live (Kinect Klon)
- LIDAR (360°, 1° Auflösung)
- Odometrie und Gyroskop
- Bumper vorne, links und rechts
- Klippensensor



Die Hardware: Nvidia Jetson TK1

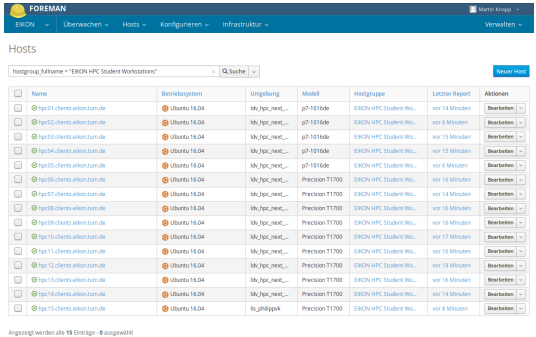
- Quad-Core ARM Cortex-A15
- 2 GB RAM, 16 GB Flash
- Nvidia Kepler GPU, 192 CUDA-Recheneinheiten
- USB 3.0, WLAN, …



Entwicklungsumgebung

Ubuntu Linux (z.Z. 16.04 LTS)

- (bis zu) 15 Praktikumsrechner
- zentral administriert
- Für High Performance Computing (HPC) geeignet
- Gerne auch eigene Notebooks
- Support nur für Probleme, die auf den Praktikumsrechnern reproduzierbar sind



Warnung: Die HPC Rechner werden vom Lehrstuhl verwendet, wer einen Neustart verursacht fliegt aus dem Kurs!

ROS – The Robot Operating System

Was ist ROS *nicht*?

- eigenständige Programmiersprache
- Entwicklungsumgebung

Was ist ROS dann?

- Bibliothek für C++ oder Python
- Erweiterung für CMake
- Bietet Abstraktion und Modularisierung
- Erlaubt die einfache Erstellung Verteilter Systeme mit Message Passing (aber kein „echtes“ MPI!)
- Diverse vorgefertigte Module für Hardware und Algorithmen
- Tool für Visualisierung

ROS: Warum nicht was anderes?

- Open Source
- Der de-facto-Standard
  - Industriell genutzt / unterstützt
  - sehr viele Algorithmen sind verfügbar und getestet
  - viele Hersteller (z.B. Robopeak vom Lidar) stellen ROS Code zur Verfügung
  - teilweise auch aktuelle Veröffentlichungen
  - gute Unterstützung verschiedener Plattformen
- Modularisierung
  - Multi-Agent Systeme
  - Fernsteuerungen
  - Austausch einzelner Teile

ROS: Testmöglichkeiten

- Austausch von Teilen (z.B. Navigationsalgorithmen, Simulatoren)
- Aufnahme und Wiedergabe von Sensordaten (ROS-bags)
- beides ist nahtlos eingebunden
- keine Codeänderungen nötig

ROS-Konzepte: Der Laufzeitgraph

- Peer-to-Peer Netzwerk von Prozessen
- Zentraler Master (roscore)
- Knoten (nodes)
- Synchrone Kommunikation (services)
- Asynchrones Streaming (topics)
- Parameterspeicher (Teil des Masters, z.B. für Konfigurationsdaten)

ROS-Konzepte: Die Knoten (Nodes)

- Konten machen die eigentliche Arbeit
  - Ansteuerung von Hardware
  - Navigation
  - Bilderkennung
  - Entscheidungsbäume
  - ...
- ROS ist sehr fein-granular angelegt $\rightarrow$ typischerweise viele Knoten
- Ein ROS-Knoten nutzt immer eine ROS-Clientbibliothek, z. B. roscpp oder rospy
- Eigenständiges kompiliertes Programm im Sinne von C++: Jeder node besitzt genau ein `int main(int argc, char *argv[]){ ... }`

ROS-Konzepte: Der Masterserver (roscore)

- Namensauflösung
- Informationen über den Laufzeitgraphen
- Ohne Master könnten sich die Knoten nicht finden und keine Nachrichten austauschen
- Speichert angebotene Topics, Services und registrierte Knoten

ROS-Konzepte: Nachrichten

- Alle Kommunikation erfolgt über Nachrichten (messages)
- Datenstruktur aus Primitivtypen oder anderen messages
- Primitivtypen umfassen Integer, Gleitkommazahlen, Boolean, etc.
- Arrays sind möglich
- Im Prinzip wie structs in C++

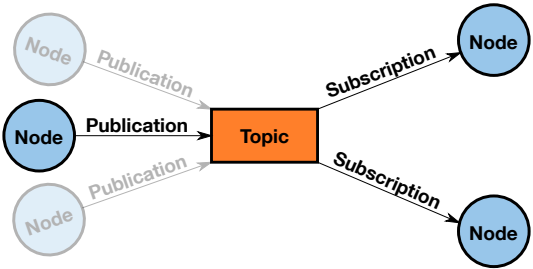
ROS-Konzepte: Asynchrones Streaming (topics)

- Nodes verschicken Nachrichten durch Publikation unter einem bestimmten Thema
- Thema (topic) spezifiziert den Inhalt der Nachricht
- Nodes, die bestimmte Daten benötigen, abonnieren das entsprechende Thema (subscribe)
- Vergleichbar mit einem stark typisierten Kanal
  - Jeder Kanal hat einen Namen
  - Jeder kann sich mit dem Kanal verbinden und Nachrichten empfangen oder versenden
  - Aber nur wenn er sich an die Typvorgaben hält
- Nicht-blockierend!

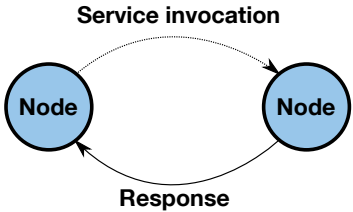
ROS-Konzepte: Synchrone Kommunikation (services)

- Request-/Reply-Modell
- Nachrichtenstruktur ist paarweise definiert
- Vergleichbar mit einem Remote Procedure Call
- Knoten A bietet einen Dienst an
- Knoten B benutzt diesen Dienst indem er eine Anfrage an A schickt und auf die Antwort wartet
- Blockierend!

ROS-Konzepte: Topics vs. Services



- m:n-Kommunikation
- Nicht-blockierend
- Keine Garantie das Nachricht verarbeitet wird



- 1:1-Kommunikation
- Blockierend
- Es gibt definitiv einen Zuhörer
- Feedback falls Kommunikation dennoch scheitert

Hausaufgabe 1 (cf. Moodle)

Ziel: Schreibt in „turtlesim“ mit der Schildkröte eure Initialen auf den Boden

- Deadline: 30. Oktober, 17:00
- Nutzt die ROS-Tutorials
  - „Creating a ROS Package“
  - „Simple Publisher and Subscriber (C++)“
  - „turtlesim tutorials“
  - …
- Legt einen neuen *branch* an
  - wählt als Name „h1_vorname_nachname“
  - nach der Deadline wird eine Kopie des *branch* erstellt
  - dies ist die offizielle ”Abgabe”
- Bei Problemen: Fragt nach!

Bis nächste Woche