ROS: Topics, Services und Actionlib
Was man wie wann und warum benutzt

Kurzfassung ROS-Backend

- Roscore vermittelt TCP-Verbindungen zwischen Prozessen
- Interprozesskommunikation wird immer Serialisiert
- Nutzung von Shared-Pointern bei Nodelets verhindert Kopien
- Intern wird alles über Topics und Messages gelöst

Topics/Messages

- Anonymes Unidirektionales System
- Publisher meldet Bereitschaft an Roscore
- Subscriber erhält vom Roscore Direktverbindung zum Publisher
- Wenn ein Topic keine Subscriber hat werden Messages verworfen
- Nachdem die Verbindung aufgebaut ist gehen Messages nicht verloren, außer der Subscriber-Buffer wird überschritten
- Buffer funktioniert nach FIFO-Prinzip, älteste Daten fallen raus
- Eigene Messages können einfach definiert werden

Topics/Messages (Nachrichten-Definition)

- Im Ordner msg

- Syntax:

  std_msgs/Header header

  std_msgs/Int8 Wert

  ...

Services

- Bidirektionales System
- Eine Message als Anfrage, eine als Antwort + Bool
- Ros::ServiceClient client = nh.serviceClient<Typ>(„Service“)
- Jede Service-Message hat
  - .request (Anfrage)
  - .response (Antwort)
- Aufruf des Services über client.call(Service-Message)
  - Blockiert bis der Service ausgeführt wurde
  - Boolean Rückgabewert ob der Call erfolgreich war oder nicht

Services (Client-Seite)

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv){
  ros::init(argc, argv, "add_two_ints_client");
   ros::NodeHandle n;
  ros::ServiceClient client =
n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
  if (client.call(srv))  {
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
  else  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

Services (Server-Seite)

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"

bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
  ros::NodeHandle n;

  ros::ServiceServer service = n.advertiseService("add_two_ints", add);
  ROS_INFO("Ready to add two ints.");
  ros::spin();

  return 0;
}
```

Services (Message-Format)

std_msgs/Int32 Anfrage

---

std_msgs/Int32 Antwort
std_msgs/String Immer_noch_Antwort

Action-Lib

- Services ++
- Fügt Status-Message und die Möglichkeit des Abbruchs hinzu
- Gedacht für einbindung zeitintensiver und fehlerbehafteter Funktionen in ROS (z.B. Objektmanipulation oder Navigation)
- Ein Action-Server arbeitet seine Ziele immer nacheinander ab
- Über Status-Message wird der Zustand vom Client eingesehen
- Intern läuft auch eine State-Machine die vom Client auslesbar ist
- Blockiert nicht wie ein Service
- Move_base als Erklärbeispiel

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
EIT FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Server State Transitions