USI/SUPSI IDSIA

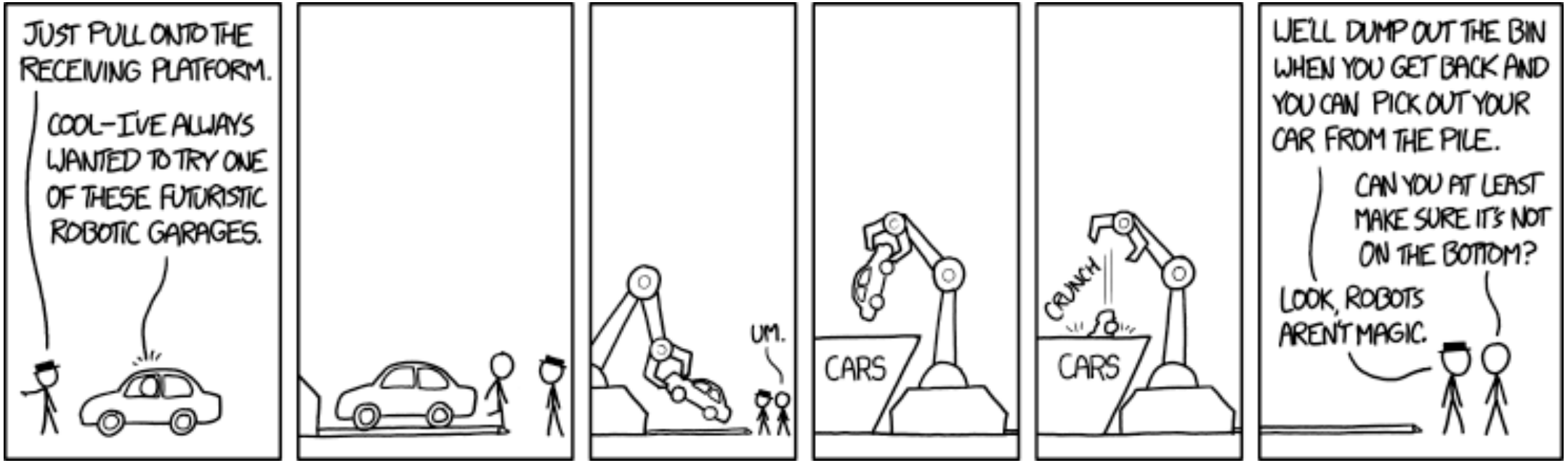
Università della Svizzera italiana

Part 1

Introduction to ROS

Elia Cereda, Simone Arreghini

Real robots



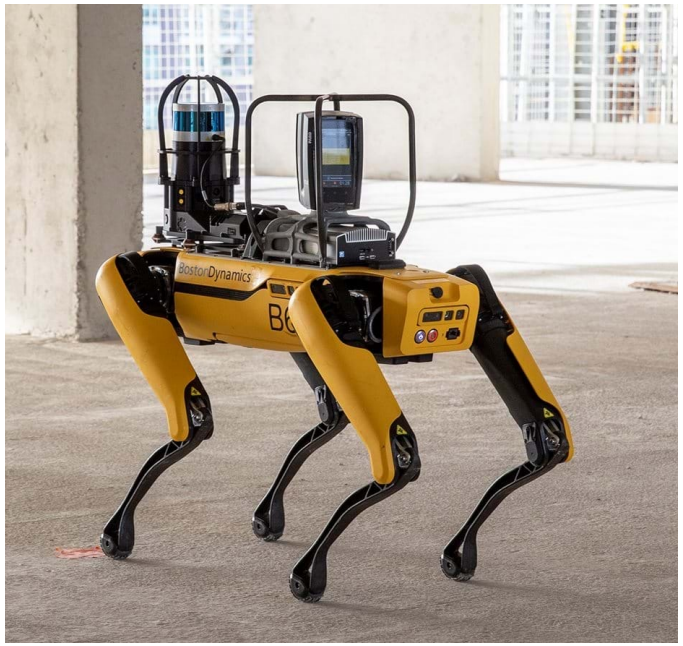
https://xkcd.com/1651/

Real challenges

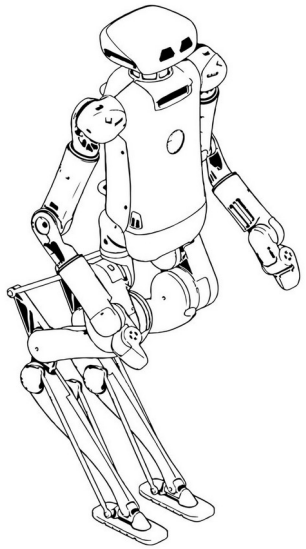
- The world is inherently asynchronous
- Events may occur at different times
- Current actions may need to be preempted
- Example: odometry data arrives at the same time with range-sensor data
- The robot should change its control actions with respect to the sensed environment, e.g., perform obstacle avoidance, and initial goals

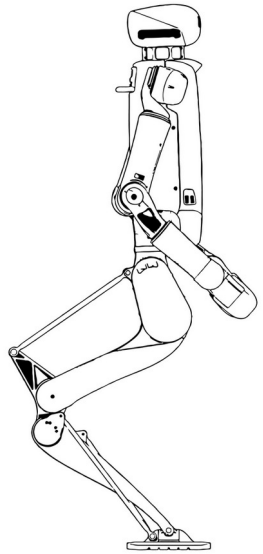
Real challenges

- Real robots are complex: dozens of sensors and actuators



Spot (Boston dynamics)





Digit (Agility Robotics)

Real challenges

- Robot's hardware is very different (sometimes even custom), e.g., different sensors measuring the same physical property can represent data differently
- Software expects data of certain types in certain formats, used by many algorithms
- Non-compatible software interfaces, e.g., proprietary drivers

LIDARs: Hokuyo, Sick, Velodyne.

Introduction to ROS - Part 1

What is ROS?

- Meta-operating system
  - Hardware abstraction
  - Package management
  - Commonly used functionality, e.g., filesystem navigation
  - Message-passing between processes

- A **software framework** for robotics applications, i.e., a set of libraries and tools targeted at robotics software development

Why ROS?

- ROS2 is the **de facto standard** for research on robotics
- Open-source framework compatible (officially or through community) with almost 350 different robotic platforms (2022 data)
- Compatible with **C++** and **Python 3**
- Contemporary and in continuous **updates**
- **ROS1** is still used in a variety of situations both industrial and in academia. One core difference with ROS2 is that ROS1 acted as a centralized network with a master that handled communication

Useful resources

- https://www.ros.org
  - official website
- https://docs.ros.org/en/humble/
  - documentation
- https://answers.ros.org/questions
  - community supported Q&A

(like StackOverflow)

- https://docs.ros.org/en/humble/Tutorials.html
  - introduction to ROS and some basic tutorials
- https://docs.ros.org/en/humble/Concepts.html
  - Conceptual overviews about key aspects of ROS 2.
- https://design.ros2.org/
  - a nice doc that describes the design choices

behind ROS 2

- https://index.ros.org/
  - registry of all ROS packages

A very brief history

- 2007: Originally developed by Stanford Artificial Intelligence Laboratory
- 2008-2013: Primarily developed by Willow Garage where it took its current shape
- 2013: Development transitioned to Open Source Robotics Foundation (OSRF)
- 2014: Initial development of ROS2 with native Python 3 compatibility
- 2020: ROS Noetic is the final release of ROS1
- June 2020: First stable version of ROS2 Foxy (current LTS)
- May 2021: First ROS2 release supporting rosbags: ROS2 Galactic

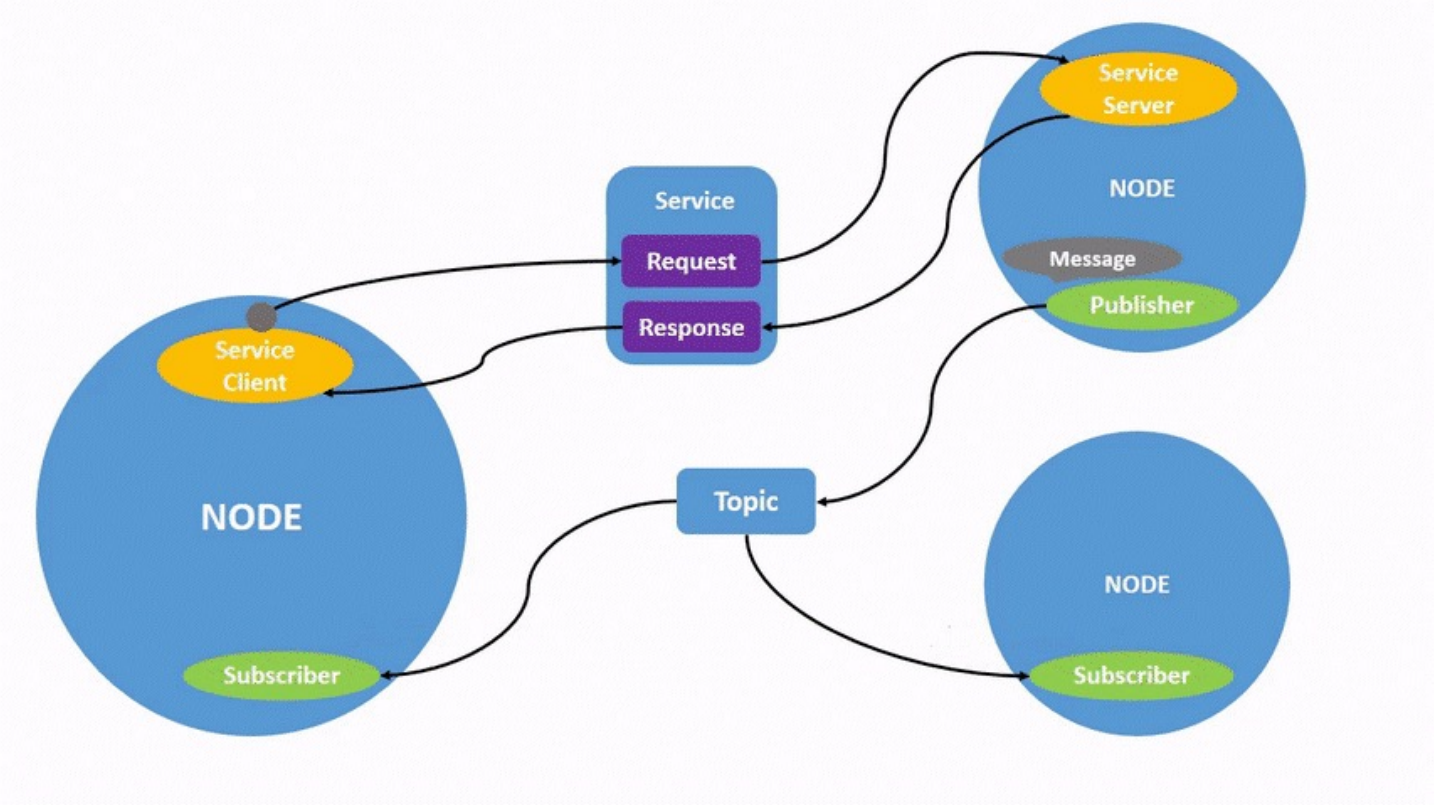
Key features

- Provides high-level **hardware abstraction** layer for sensors and actuators
- Provides extensive set of **standardized message types** and **services**
- Utilizes distributed **peer-to-peer network** architecture
- Forms a **computation graph**, i.e., a distributed data processing system
- Fault-tolerant: computing units (**nodes**) are separate OS processes
- Lots of **community packages**: +11k (incl. 3rd party)
- **Open-source** software: Most of the packages are under BSD-license
- **Ubuntu** and **Windows** officially supported; **macOS** supported by the community

Concepts

- **Filesystem**
  - Workspaces
  - Packages: Manifest, Message description, Service description, Launch files
- **Computation Graph**
  - Nodes: Topics, Messages, Services, Actions, Parameters
  - ROS client libraries (language bindings): `rclpy`, `rclcpp`
- **Build System**: Colcon
- **Command-line Tools**: `ros2 node`, `ros2 topic`, `ros2 service`, ...

Computation graph



Nodes

- A single executable files can contain more nodes
- Fine grained processes that perform computations
- Focused on a single purpose
- Communicate with other nodes using **Topics**, **Services, Actions** and are configured with **Parameters**
- Usually publish and subscribe to several topics/services
- May run on different hosts of a network
- Uniquely identified by graph resource names (like filesystem paths)

Messages

- The means of node communication
- Data structures comprising typed fields
- Supported field types:
  - Primitive types: **integer**, **boolean**, **string**, **double**
  - Complex types: **list** (array), **dictionary** (map)
  - Compound types (incl. other message types)
- Routed via a peer-to-peer system with publish and subscribe architecture
- Communications happens on a DDS transport system (not only limited to TCP)

https://github.com/ros2/common_interfaces
https://docs.ros.org/en/humble/Concepts/About-ROS-Interfaces.html

Topics

- Follow **many-to-many** paradigm
- **Named**, **strongly typed** and **unidirectional** data channels / message buses
- Any node can “connect” to a topic in order to send or receive data
- Nodes **subscribe** to a topic to receive **messages** and **advertise** a topic to send **messages**

Services

- **Named** and **typed** data channels that follow request / reply model
- There can be more Service Clients but only one Service Server
- Unlike topics the services are a way of sharing data on request
- Defined by a pair of **messages:** one for request (input) and one for reply (output)
- Not cancelable

Actions

- Another comunication method in ROS2
- Used for long running tasks. Can be cancelled. Actions are **typed**
- Built upon topic and services, actions consist of three parts: **Goal, Feedback** and **Result**
- View them as a more complex service, actions can be called, feedback can be received in addition to the result
- Actions work as a client-server model. There are Action clients and servers. A client sends a goal to the server and it respond with a feedback and the result.
- Both the server and client can cancel an action. If it happen server side the action is it said to be “aborted”

Parameters

- Configuration **Values** for a node
- Consider them as Node's settings, can be getted and setted
- It is possible to dump a node's parameters into a file and load them later
- Parameters are **typed**
- We can attach callbacks to a parameter to know when a parameter is going to change and after it has changed.

Name resolution

- Approach is somewhat similar to path resolution on UNIX-systems
- Four types of names: **base**, **relative**, **global** and **private**
  - *base*
  - *relative/name*
  - */global/name* (preferred one)
  - *~private/name*
- By default, resolution is done **relative** to the node's namespace
- Names without namespace qualifiers are the **base** names
- **Global** names are fully resolved
- **Private** names convert node's name into a namespace

Introduction to ROS - Part 1

Command-line tools

- `ros2 run <package> <executable_file>` — runs an executable file of a package
- `ros2 launch <package> <launch_file>` — launches multiple nodes from a single batch Python or XML file
- `ros2 node <sub–command>` — displays information about nodes
  - `list` print all nodes
  - `info <node_name>` print information about precise node
- `ros2 topic <sub–command>` — displays information about topics
  - `list` print all active topics
  - `echo <topic_name>` print messages to screen
  - `pub` publish data to topic
  - `info <topic_name>` print information about active topic
  - `hz <topic_name>` display publishing rate of topic
- `ros2 <interface> <sub–command>` — General ros2 cli structure for communication interfaces
  - Interfaces:
    - `topic`
    - `service`
    - `action`

Let’s try!

Exercise 1: Student topic

- Open a **terminal** and type `ros2 topic list` and press Enter
- You will see all the available topics in ROS at this time
- In **this terminal** try also `ros2 topic info /rosout`
- This will display relevant information about the topic `/rosout`
  - Message type
  - Publisher
  - Subscribers

Exercise 1: Student topic

Let's try publishing a message

- In the **last terminal** type:

```
ros2 topic pub /student example_interfaces/msg/String 'data: your_name'
```

- In **another terminal** type `ros2 topic list`
- You can see the */student* topic now!!!
- Let's check the messages inside this topic with:

```
ros2 topic echo /student
```

Final Remark

While we understand the appeal of tools like Bing, ChatGPT or similar, you should know that these tools are **almost useless with ROS2** (and most of the times wrong!) since the documentation is still in the writing and most of the robot tools you will use are custom made from us (IDSIA Intelligent Robotics Group)

If you have questions, please write us on Teams or via email!

Our emails are

elia.cereda@usi.ch

simone.arreghini@usi.ch

Send your emails to the both of us or even better write on the iCorsi forum