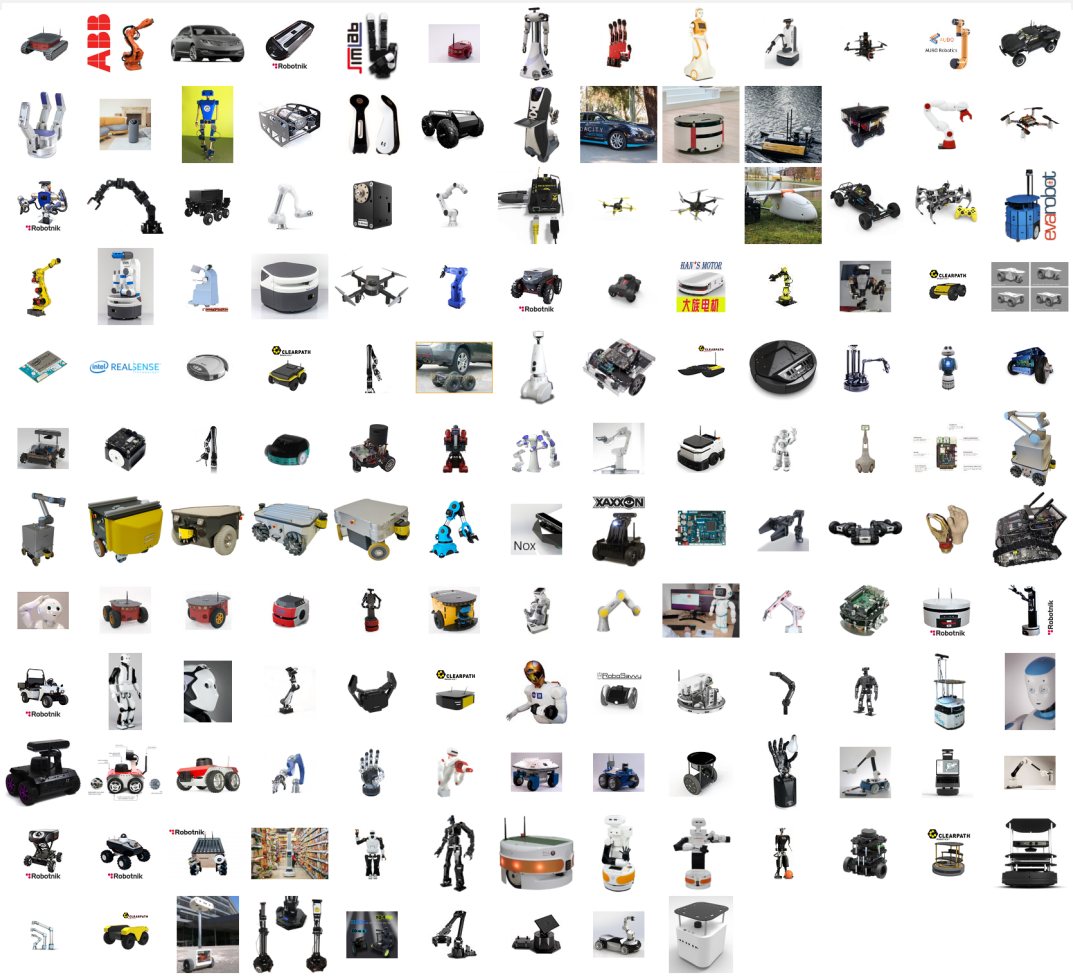
Introduction to ROS

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Technical Aspects of Multimodal Systems

October 14, 2022

Developing for Robots... Which One?



Source: https://robots.ros.org/

Should It Matter?

- Heterogeneity vs. Homogeneity
  - Hardware differs, but is often reused in different systems
- Abstraction
  - For many aspects the exact hardware does not matter
  - Robot Models, Navigation, Object Manipulation, Perception, ...
  - Generic algorithms can be reused
  - Avoid vendor lock-in
- Debug & Testing
  - Execution recording
  - Data visualization
  - Simulation

Framework Support

- We use Robot Operating System (ROS)
- Hardware-agnostic framework for generic robot programming
- **No** operating system
- OpenSource (mostly BSD-licensed)
- Highly Community-driven
- Support for a substantial number of robots
- One of many, many systems, but very popular
  Others: YARP, ArmaX, ROS2, MSRS, ROCK, OpenRave, ViSP, Orocos, ...[1]

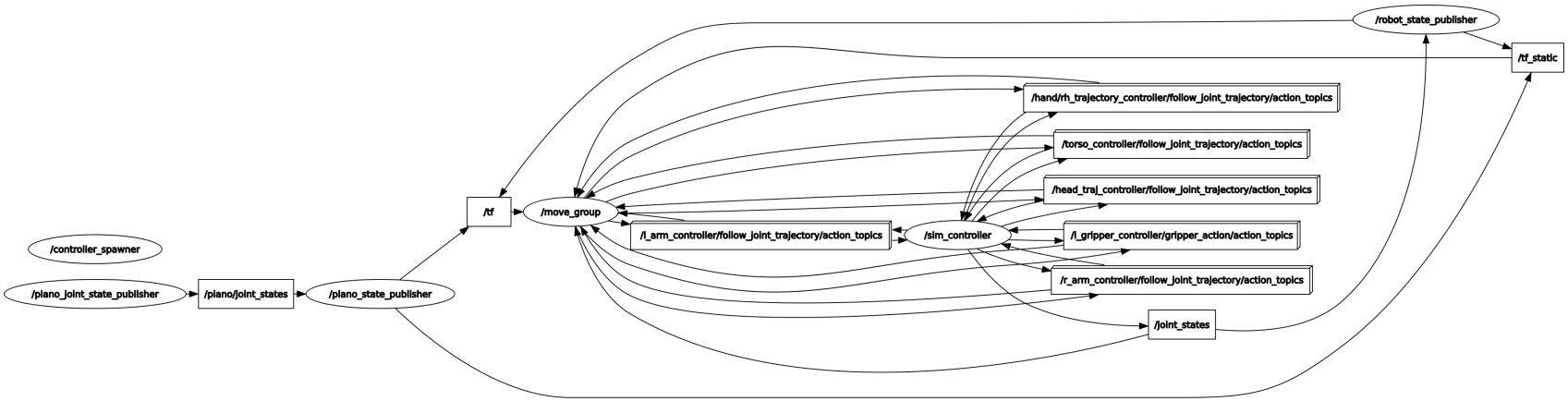
[1] Tsardoulias, Emmanouil & Mitkas, Pericles. (2017). Robotic frameworks, architectures and middleware comparison.

Current State

- Actively used and developed for 12+ years
- Mostly on Linux (Mostly Ubuntu)
- Multiple active versions (melodic, noetic)
  Supporting Ubuntu 18.04, 20.04 and other systems
- Supports C/C++, Python, Java, Lisp, Octave ...
- Various modules and algorithms are available in the community
  - Consider "use&improve" over reimplementing basics
- **ROS2** aims to succeed **ROS** eventually
  - Currently both coexist with different strengths

ROS Runtime System

- Modular
- Graph-based
- Message Passing (well, mainly)



ROS Core

```
$ roscore
```

Provide basic infrastructure

- ROS Master
  - Central XML-RPC server for communication
  - Global parameter server for easy configuration of *any* node
- rosout
  - Convenient message logging
  - one-line logging of debug/info/warn/error messages
  - fancy print/printf

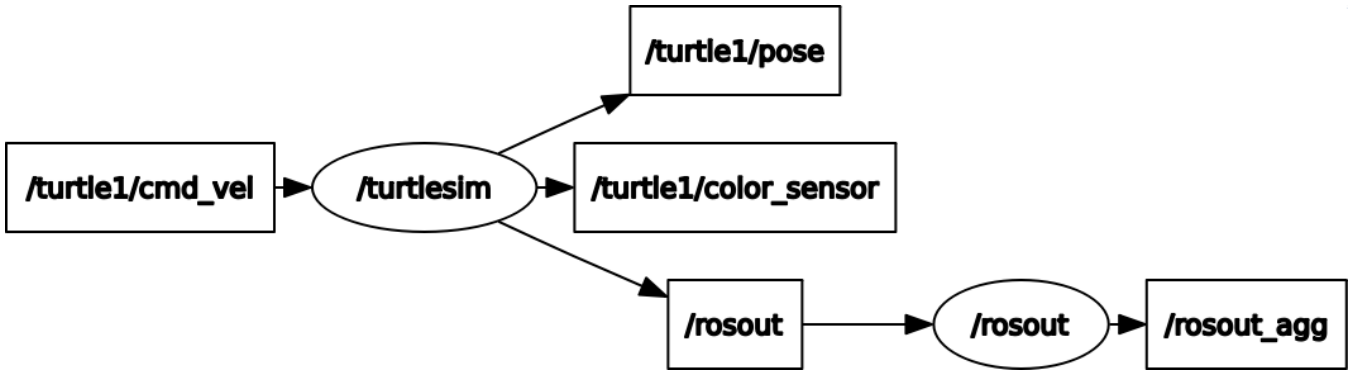
This is implicitly started with

```
$ roslaunch ...
```

ROS Nodes

- Basic unit of computation
- Just a system process
  - . . . with specific interfaces
- Could
  - talk to sensors, e.g., laser scanner, camera, force sensors
  - actuate robot, e.g., individual servos, wheels, whole robot arms
  - implement isolated logic

Communication



- Nodes pass **Messages** (e.g., Pose2D: $x/y/\varphi$)
- via **Topics** (e.g., `/turtle1/pose`)
  - anyone can publish, anyone can subscribe (m:n)
- Remote function calls are called **Service**
- *Asynchronous* calls are **Actions**
  - 1:1 communication

Messages

- Basic unit for data transmission
- Strongly-typed data structures
- General pattern with many use-cases (proto buffer)
- Possibly “Stamped” with a **Header**
  - Time stamp
  - Frame / reference system

```
$ rosmsg show -r geometry_msgs/Quaternion
# This represents an orientation in free space in quaternion form.
float64 x
float64 y
float64 z
float64 w
```

Messages

- Basic unit for data transmission
- Strongly-typed data structures
- General pattern with many use-cases (proto buffer)
- Possibly “Stamped” with a **Header**
  - Time stamp
  - Frame / reference system

```
$ rosmsg show -r geometry_msgs/Quaternion
# This represents an orientation in free space in quaternion form.
float64 x
float64 y
float64 z
float64 w
```

Topics

- Messages are **published** to **Topics**
- **Advertised** by Nodes
- Topics have unique names
- Anonymous publishers
- Anyone **subscribes** as needed
- Publishing triggers callbacks in subscribers

Services

- Two message types
  - **Request** and **Response**
- Synchronous protocol
  - client sends request
  - client waits for server
  - server replies

```
$ rossrv show AddTwoInts
int64 a
int64 b
- - -
int64 sum
```

Services

- Two message types
  - **Request** and **Response**
- Synchronous protocol
  - client sends request
  - client waits for server
  - server replies

```
$ rossrv show AddTwoInts
int64 a
int64 b
- - -
int64 sum
```

Actions

- Three message types
  - **Goal** and **Result**
  - optionally **Feedback**
- Asynchronous protocol
  - client sends goal
  - server may respond with feedback
  - server delivers result
- Interruptible

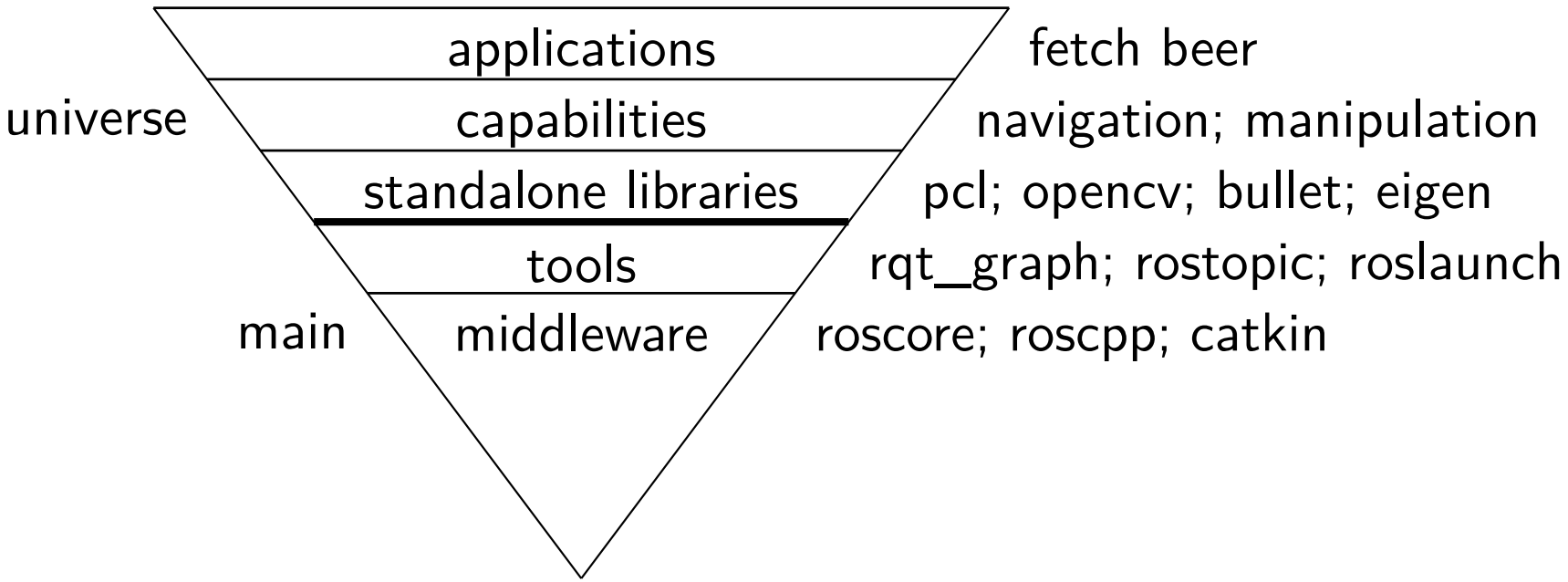
```
# Define the goal
uint32 dishwasher_id    # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```

Actions

- Three message types
  - **Goal** and **Result**
  - optionally **Feedback**
- Asynchronous protocol
  - client sends goal
  - server may respond with feedback
  - server delivers result
- Interruptible

```
# Define the goal
uint32 dishwasher_id    # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```

Userland



- universe → developed by community
- main → general tools, maintained by OSRF

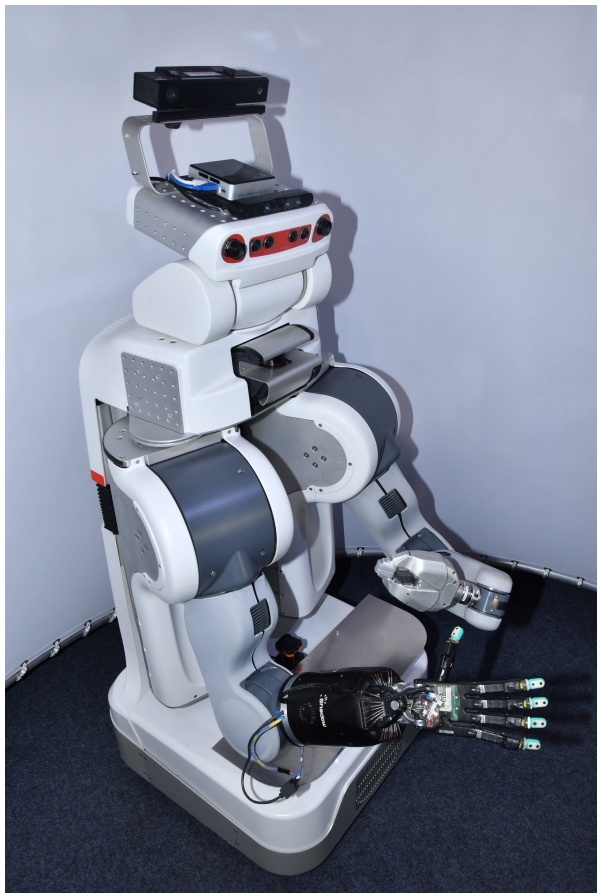
TAMS TurtleBots

- Four autonomous robots
- (Our) Hardware
  - Vaccum base
  - Kinect
  - Laserscan
  - Dell Laptop
- Capabilities
  - Navigation
  - Transport
  - Mapping
  - “Swarm” tasks
  - Laser Tag!

Source: http://wiki.ros.org/Robots/TurtleBot

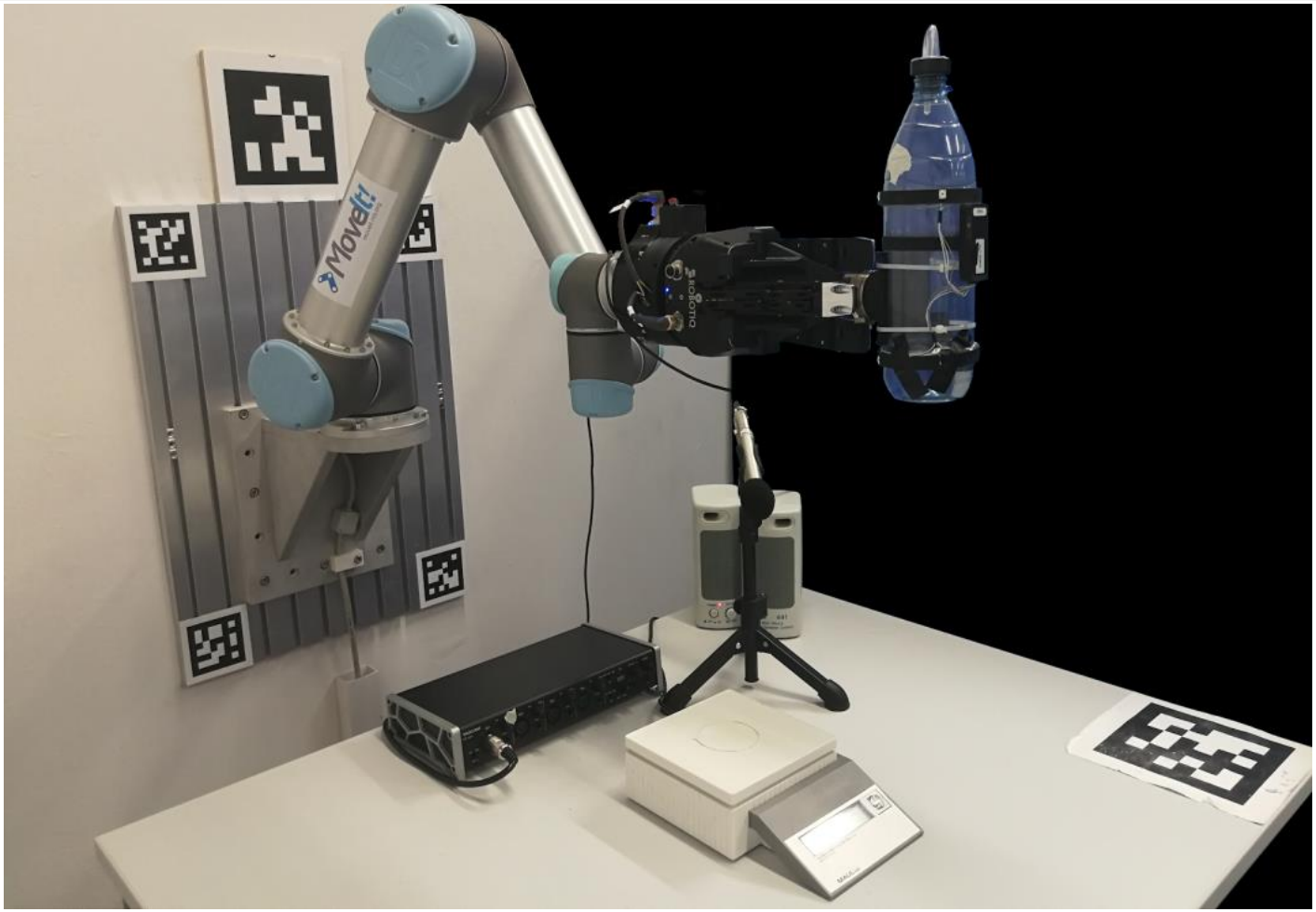
TAMS PR2

- Two server systems on-board
- Max. payload 1.8kg
- Sensors
  - IMU, Accelerometer
  - ASUS Xtion Pro Live / Kinect2
  - Two stereo camera pairs
  - Three laser scanner
  - Camera in forearm
  - Fingertip pressure sensor arrays
  - Shadow Dexterous Hand with BioTac fingertips



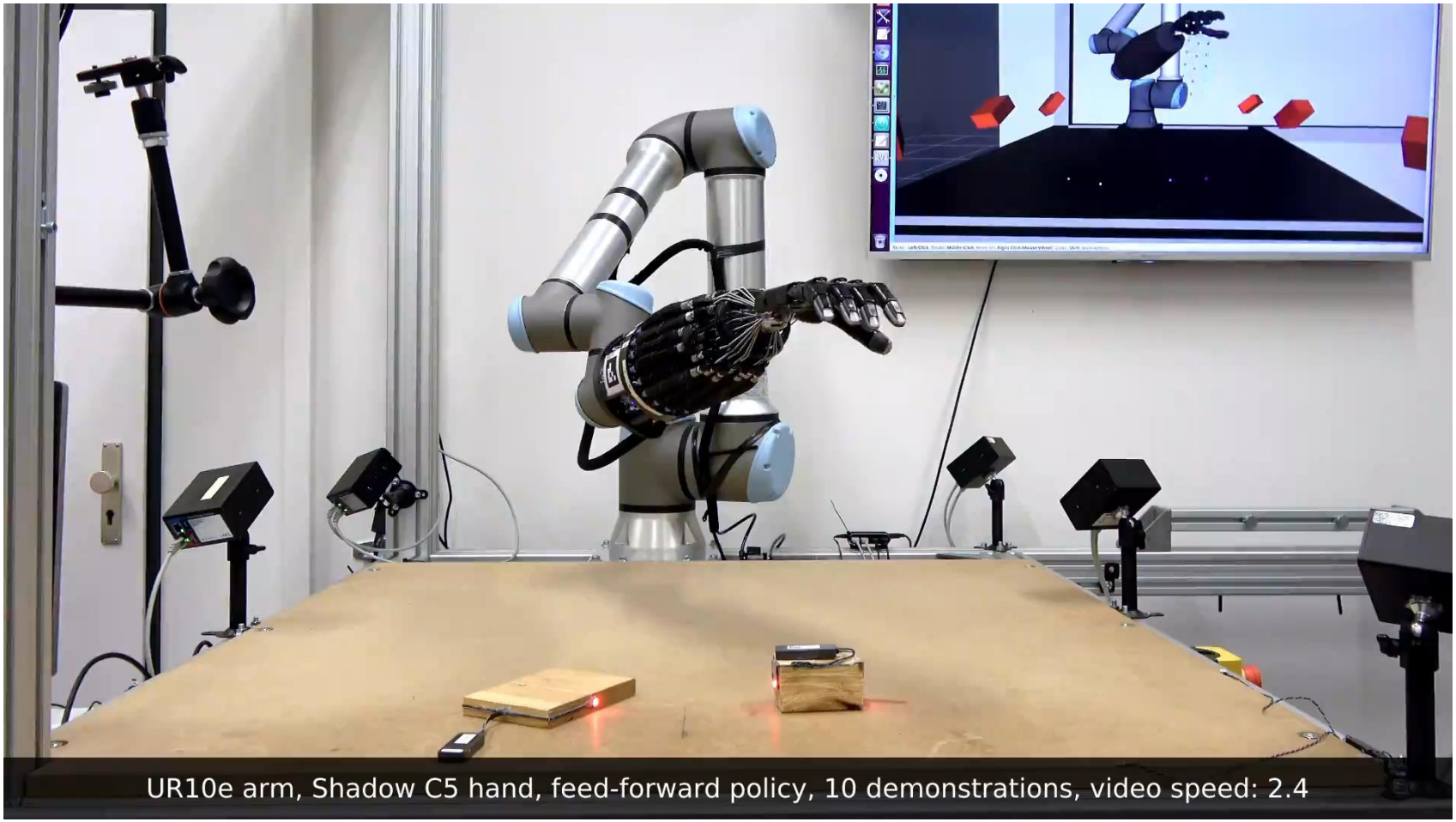
Source: Erik Strahl

UR5 & Robotiq Gripper



Source: Hongzhuo Liang

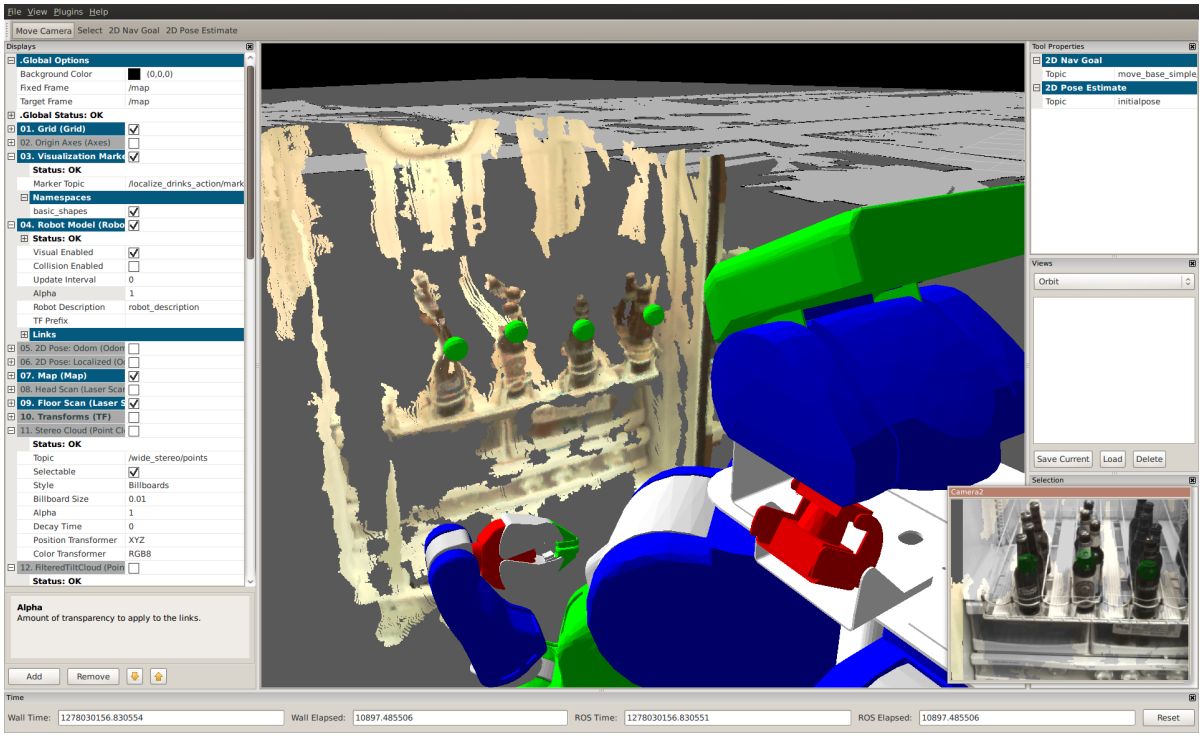
Tracking Cage with UR10e & Shadow Hand



Source: Philipp Ruppel

RViz

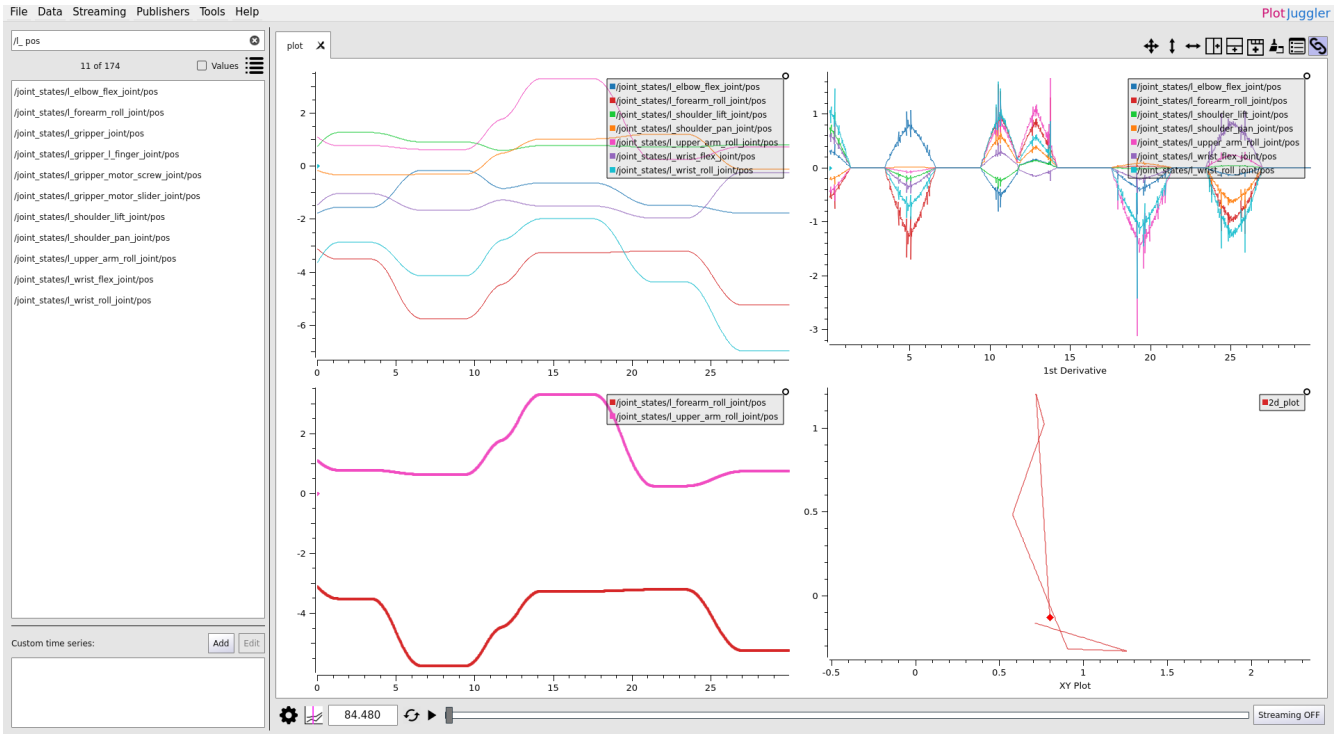
- 3D visualization environment
- Show various information sources online
  - Robot geometry, cameras, point clouds, detection results, maps...



Source: Jonathan Bohren

PlotJuggler

- Generic 2D plotting tool
- Analyze sensor signals, joint trajectories, etc.

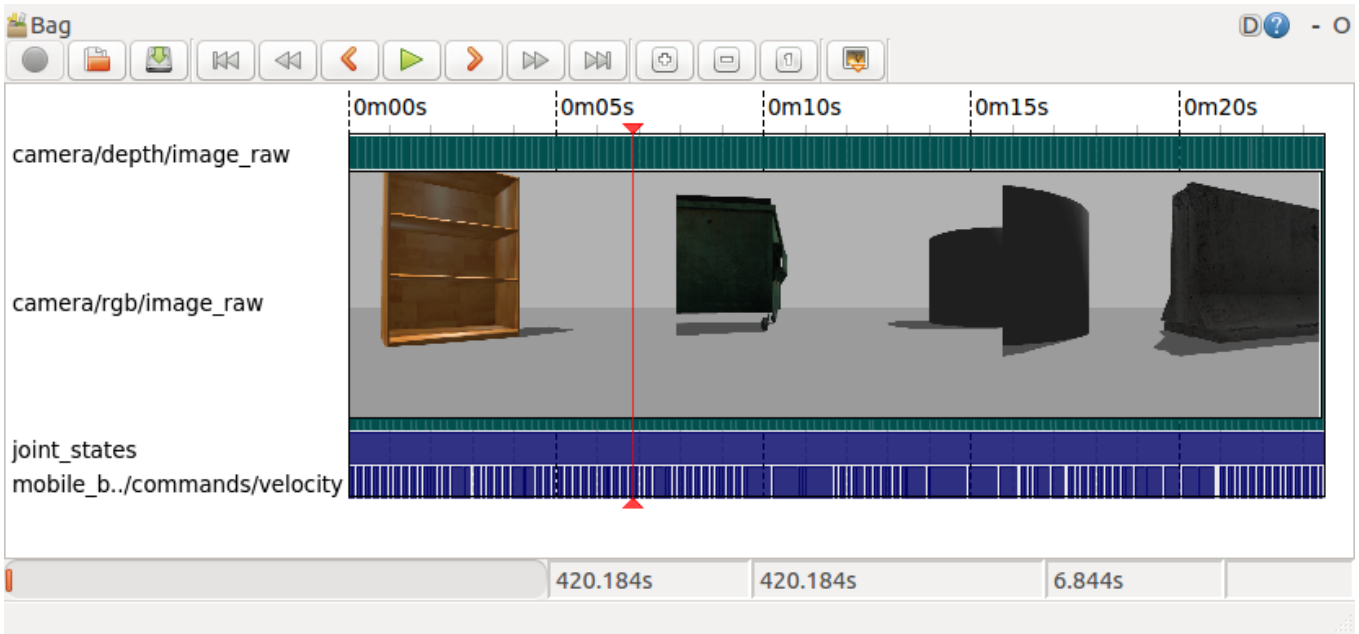


Source: Michael Görner

ROS Bags

```
$ rosbag record /topic1 /topic2
$ rosbag play file.bag
```

- Record and replay experiments via Topics



Source: https://wiki.ros.org/rqt_bag

Simulation

- Simulation is **not** real execution
- But it is an important development tool
  - Develop and test without robot
  - Can parallelize
  - Sim2Real training
- Simulates sensor data
  - Clean data / controlled noise

- Turtlesim
  - ROS learning tool
- MoveIt "demo mode"
  - Kinematic robot simulation
- Flatland
  - advanced 2D simulation

- Gazebo, Webots, CoppeliaSim, Isaac Sim, ...
  - Full-featured physics simulation
- MuJoCo, Isaac Gym, ...
  - Simulations for policy learning

Simulation

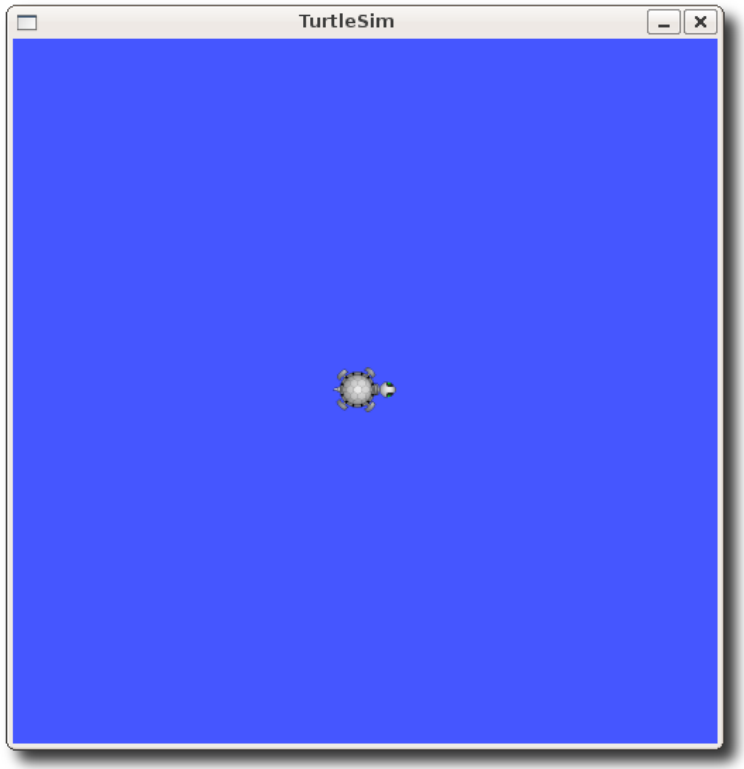
- Simulation is **not** real execution
- But it is an important development tool
  - Develop and test without robot
  - Can parallelize
  - Sim2Real training
- Simulates sensor data
  - Clean data / controlled noise

- Turtlesim
  - ROS learning tool
- MoveIt "demo mode"
  - Kinematic robot simulation
- Flatland
  - advanced 2D simulation

- Gazebo, Webots, CoppeliaSim, Isaac Sim, ...
  - Full-featured physics simulation
- MuJoCo, Isaac Gym, ...
  - Simulations for policy learning

Turtle Sim

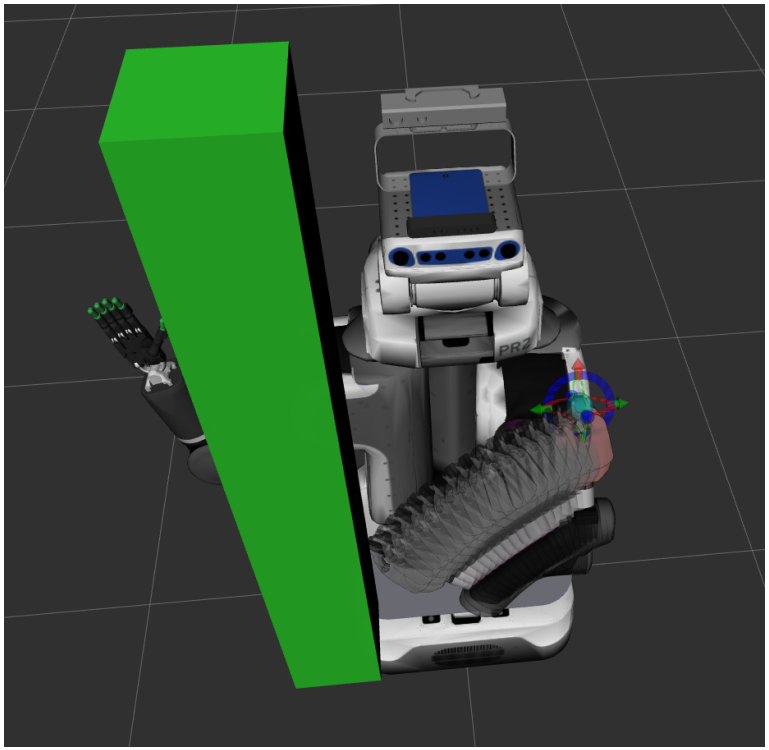
- Learning platform
- 2D turtle
  - move
  - turn
  - draw
  - sense color
- Topic & Service interfaces



Source: http://wiki.ros.org/turtlesim

MoveIt Demo Mode

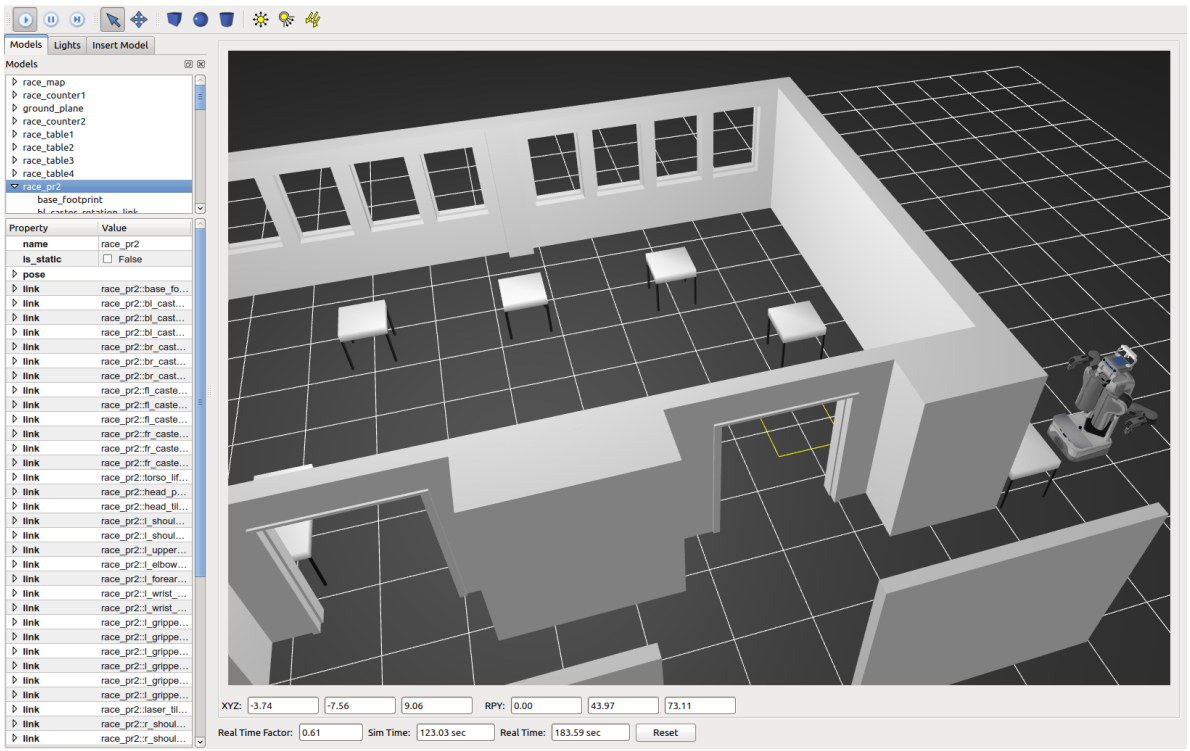
- Kinematic simulation only (no velocities!)
- Forward kinematics, state visualization
- Collision checking
- Fast
- Reachability testing / Easy integration without hardware



Source: Michael Görner

Gazebo

- 3D rigid body simulator
- Simulates robots, environment and sensor data
- Complex configuration & fragile behavior



Source: Lasse Einig