Upper-body motion planning on the REEM robot

Current state and future perspectives

Adolfo Rodríguez Tsouroukdissian
Presenter: Jordi Pagès

PAL ROBOTICS

Overview

- **First steps and usecases**
- Retrospective
- Next steps

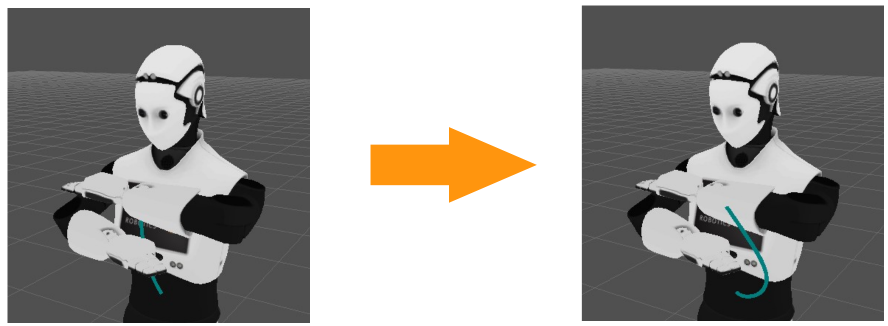
First steps

Q2 2010 Boxturtle

- Add **dual-arm** planning to **REEM-H1**
- Call **arm_navigation** from in-house codebase
- Did not use **move_arm**
  - Why? we needed **decoupled** planning and execution

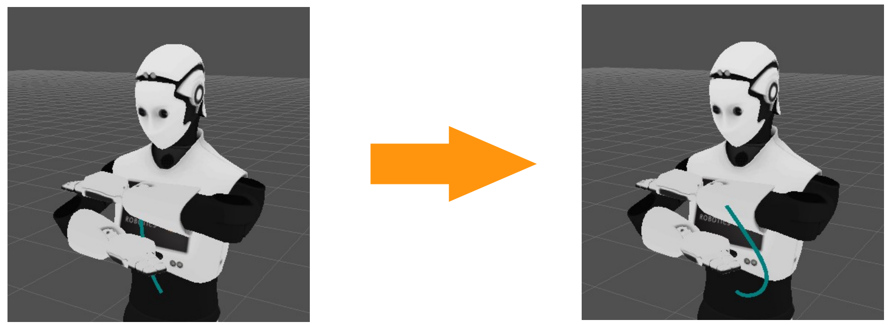
Usecase: Trajectory playback

- Plan collision-free motions between **postures**



Usecase: Trajectory playback

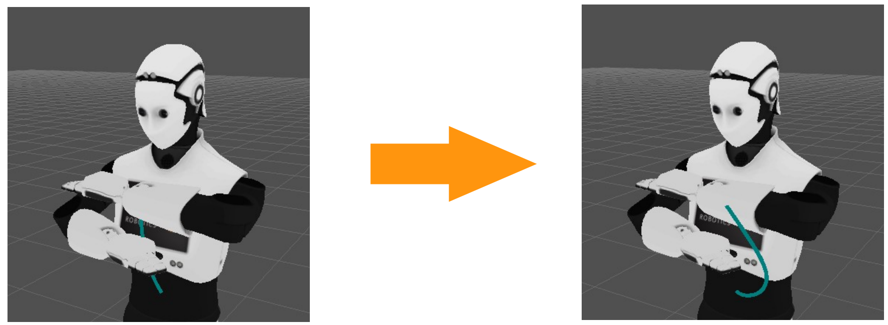
- Plan collision-free motions between **postures**



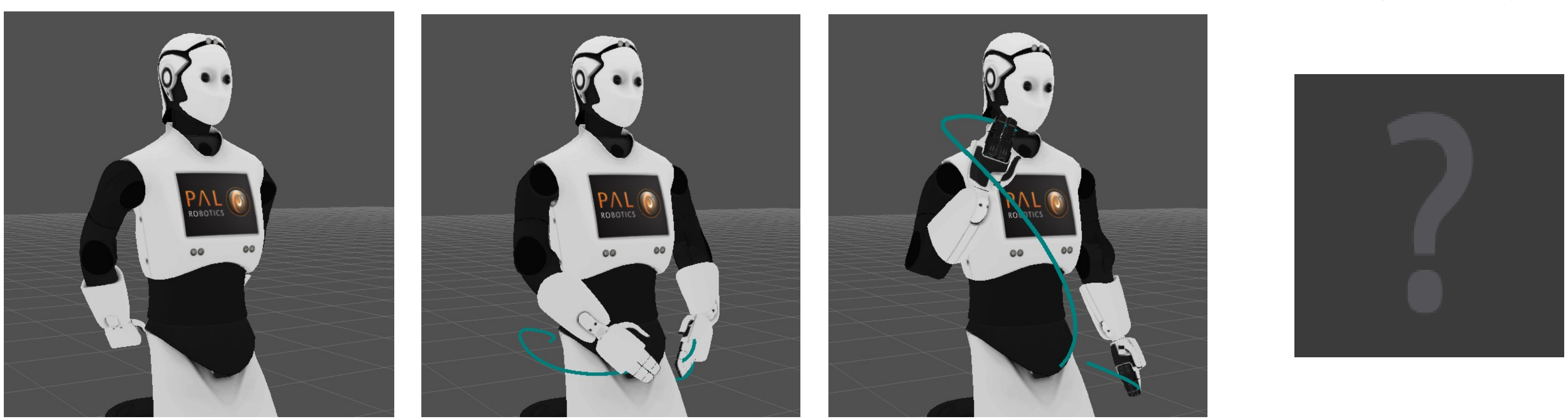
- **Pre-recorded** trajectories
  - **Collision-check** recorded trajectory
  - **Prepend collision-free approach** from current state to trajectory start

Usecase: Trajectory playback

- Plan collision-free motions between **postures**



- **Pre-recorded** trajectories
  - **Collision-check** recorded trajectory
  - **Prepend collision-free approach** from current state to trajectory start



current state | **approach** trajectory | **pre-recorded** trajectory | complete execution

motion planning | *collision checking*

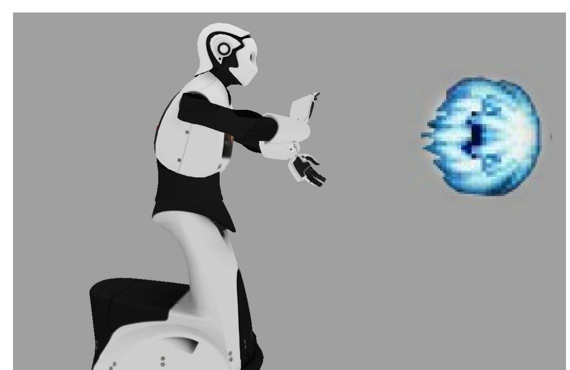
Usecase: Trajectory playback

- Where was this used?
  - Task **transitions,** interruption **recovery**

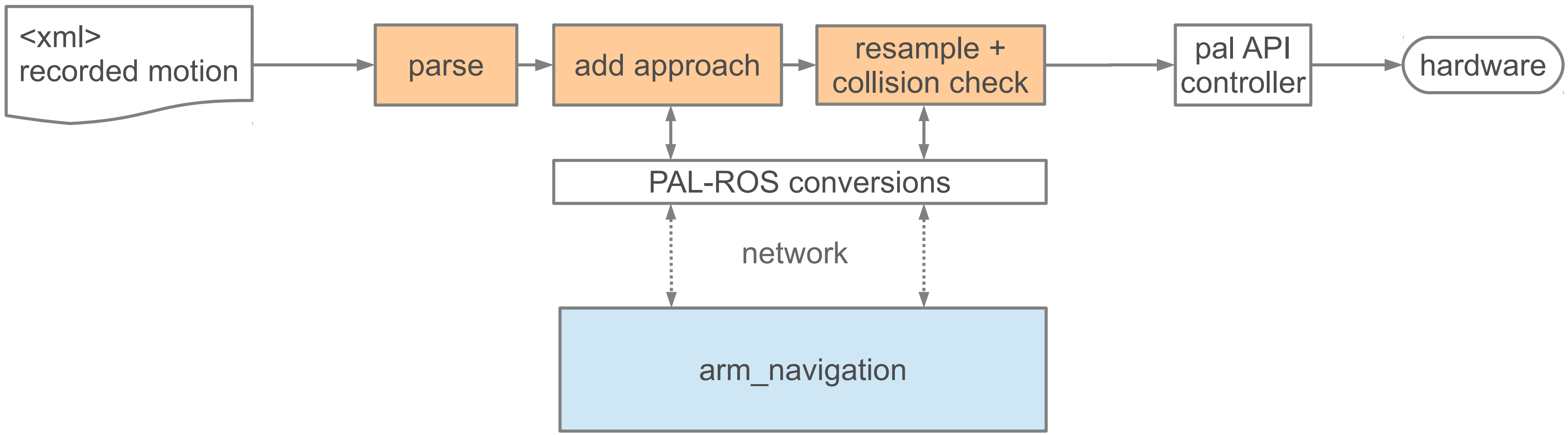
task *A* end

task *B* start

  - **Interactive motion triggers** using joystick / tablet



Usecase: Trajectory playback



- Lessons learned:
  - Setting up **arm_navigation** was **“easy”**, only config files
  - Interface with **in-house** codebase was a **considerable effort**
  - We fixed bugs twice in **move_arm** and **our code**

Usecase: Online trajectory generation

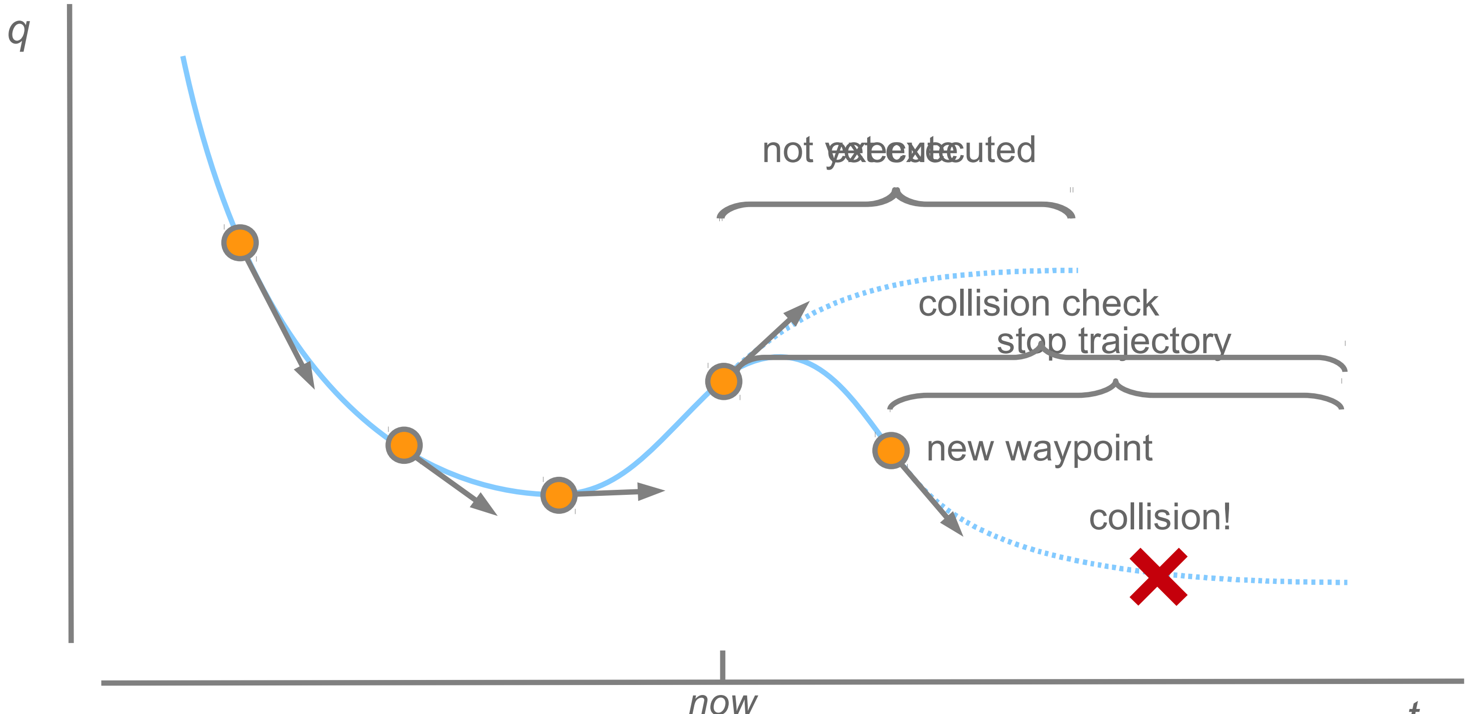
- **Online** joint trajectory sources (through IK)
  - Object tracking / visual servoing
  - Upper body teleoperation



- Constraints
  - Executed trajectories should be **collision-free**
  - Avoid **abrupt stops**, ie. *'klunk'*

Usecase: Online trajectory generation

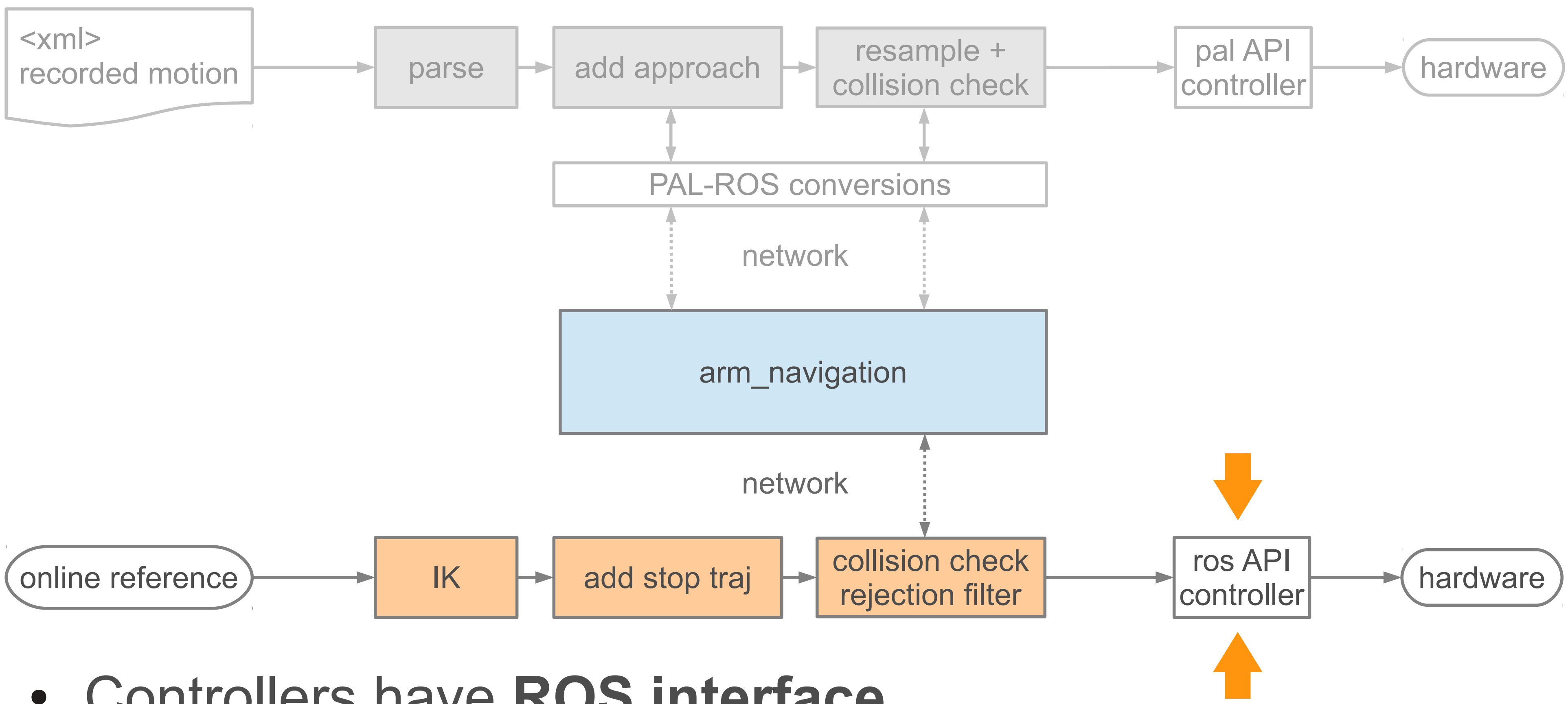
- **Smooth-stop** rejection filter (an effective hack)
  - **Input:** Next trajectory waypoint *(pos, vel)*
  - **Append stop trajectory:** first order dynamics
  - **Collision check:** input + stop trajectory
  - **Rejection filter:** Do not send command if in collision



Usecase: Online trajectory generation

- **Smooth-stop** rejection filtering (an effective hack)
  - **Input:** Next trajectory waypoint *(pos, vel)*
  - **Append stop trajectory:** first order dynamics
  - **Collision check:** input + stop trajectory
  - **Rejection filter:** Do not send command if in collision

Usecase: Online trajectory generation



- Controllers have **ROS interface**
  - Feature parity with **JointTrajectoryActionController**
  - Hard-realtime **Orocos RTT** implementation

Overview

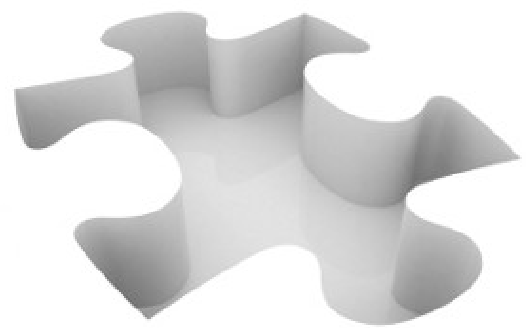
- First steps and usecases
- **Retrospective**
- Next steps

Retrospective

arm_navigation

- **3+ years** using it
- **Very satisfied** users
- Looking forward to **MoveIt!**

Let's review **open issues...**



Retrospective

- **Motion planning** usecases
  - **Similar** problems encountered **over and over**
  - **Task context** is often known by high-level coordinator
  - **Hard problems** are **infrequent** (eg. narrow passages)
- **Currently** used tools
  - **Always** plan from scratch, do not exploit experience
  - **Agnostic** to task context
  - Great for solving **hard problems**

not a great match!

Retrospective

Additional constraints

- **Problem solved:** for engineer ≠ for client

Retrospective

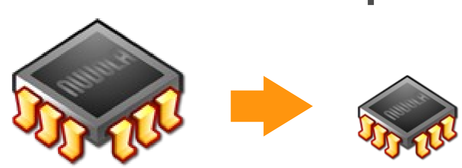
Additional constraints

- **Problem solved:** for engineer ≠ for client
  - **Continuous task execution:** Less *move-**stop-think***, *move-**stop-think***
  - **Determinism:** ~same problem → ~same solution

credit: Pastor et. al 2012

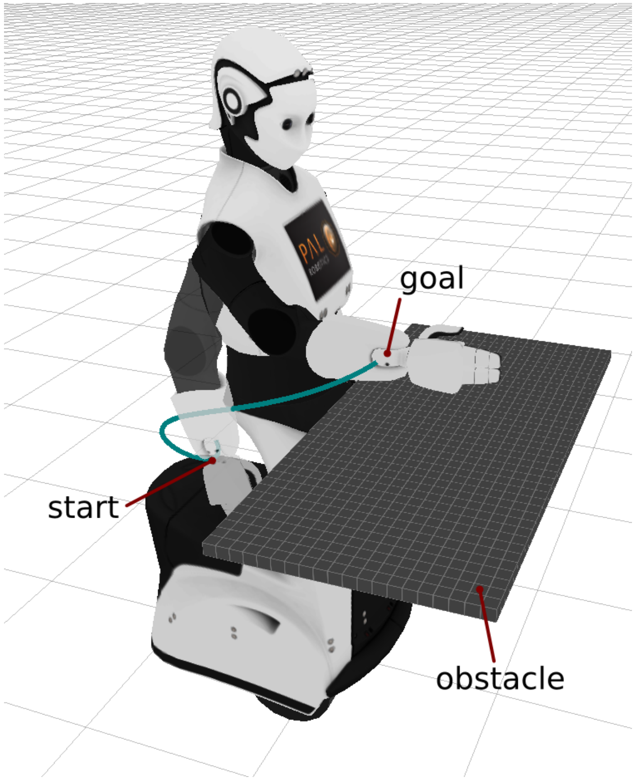
credit: Honda research 2011

- **Resource footprint:** as small as possible



Motion generation, motion recall

- Evaluation (Lopera et.al., Humanoids 2012)
  - **Motion generation:** LazyRRT
  - **Motion recall:** DMP
  - **Criteria:** variability, computational load, generality



Motion generation, motion recall

Motion generation, motion recall

TODO: Exploit **complementarity!**

- Use motion **recall** when
  - Task context is known
  - Solution to similar problem is available
- Use motion **generation** otherwise
- Motion library (long term goal)
  - Self maintaining
  - Sparse

Overview

- First steps and usecases
- Retrospective
- **Next steps**

Next steps

- Motion planning with **MoveIt!**
  - Embrace planners with **optimality** guarantees, **faster** trajectory filters
  - Leverage runtime **switching** of planning/control joint groups

- Online trajectory generation
  - **Stack of Tasks** (Escande et.al., 2012, in review - integration **work in progress**)
    - Multi priority, multi end-effector IK
    - Constraints: equality and **inequality** (joint limits, collision avoidance)
  - **Local collision avoidance:** Leverage MoveIt! proximity query alternatives

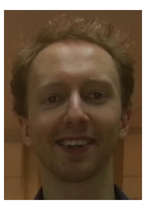
- Control
  - Unify hardware access, enter **ros_control** (work in progress)

Acknowledgements

- Intern power!

Marcus Liebhardt: teleoperation

Carmen Lopera: motion generation/recall



David Butterworth: ROS tabletop grasping

Hilario Tomé: all of the above plus more (now staff member)

- **Jordi** here, for presenting in my stead :)