Final consideration and project presentations

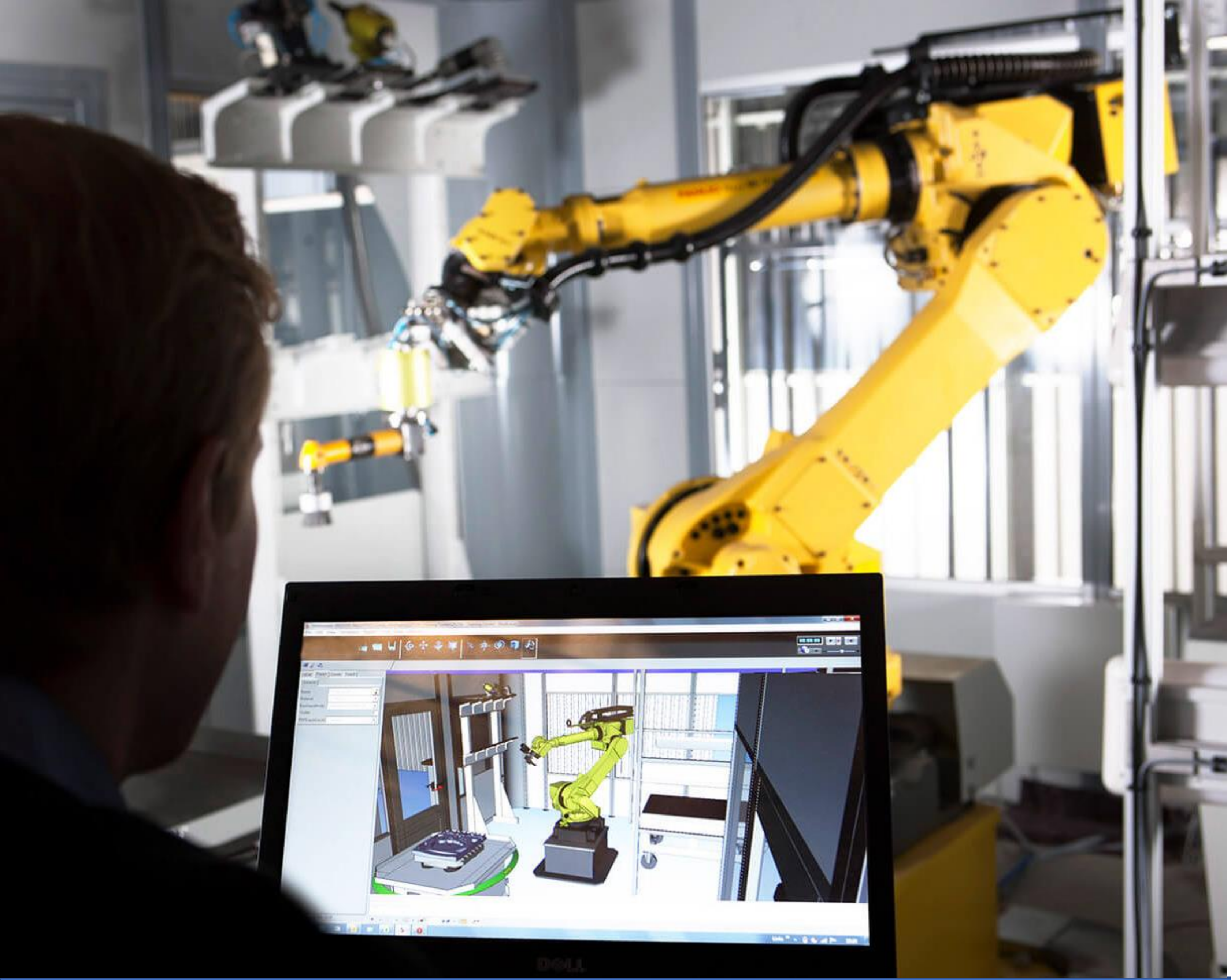
Robot Programming and Control

Accademic Year 2021-2022

Diego Dall'Alba diego.dallalba@univr.it

Department of Computer Science – University of Verona

Altair Robotics Lab



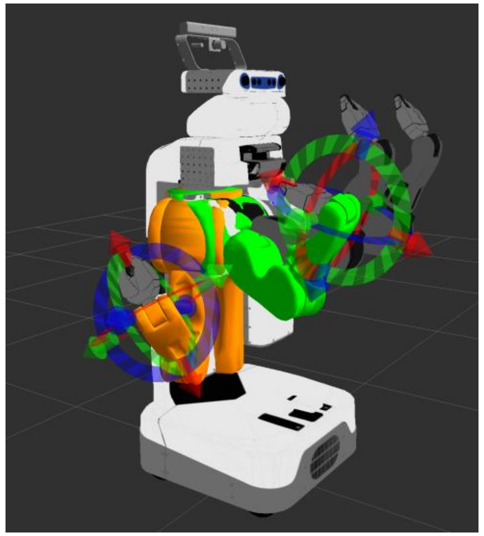
Overview

- ROS 1 design consideration
- ROS 1 case study: ANYmal
- Best practice when using ROS in complex projects
- ROS2 brief description

- Examples of Module B final projects
- Final project rules
- Final project proposals

Some last reminders about ROS design (1)

- ROS began life as the development environment for the Willow Garage PR2 robot around 2007.
- The primary goal was to provide the software tools that users would need to undertake novel research and development projects with the PR2.
- The original developer wanted ROS to be useful on other robots → Defining levels of abstraction (usually through message interfaces) that would allow much of the software to be reused elsewhere.



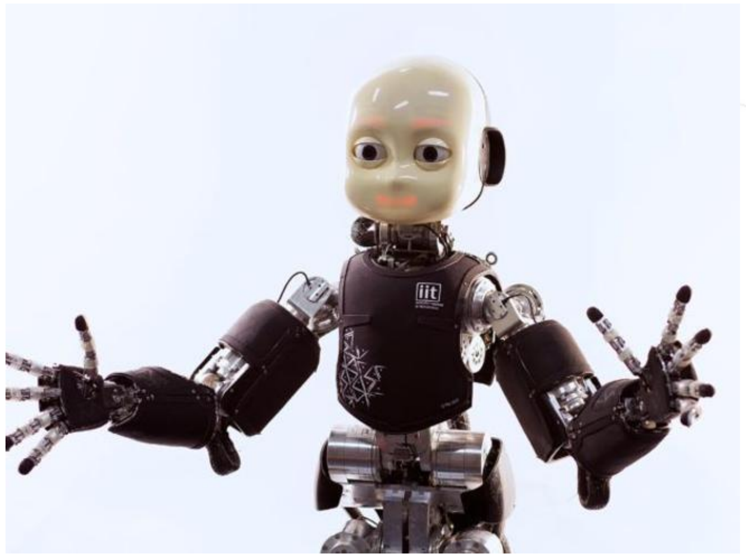
Some last reminders about ROS design (2)

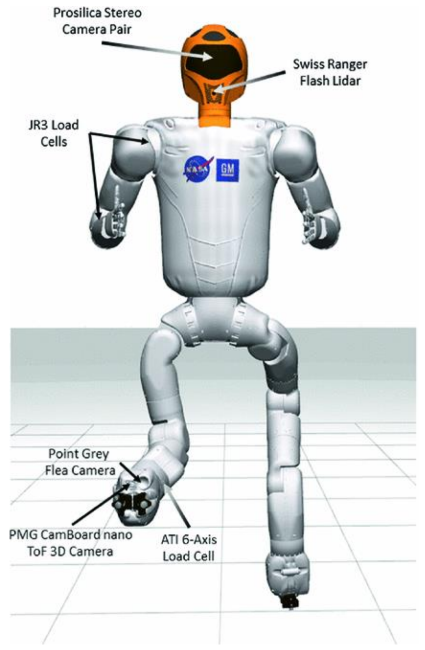
Considering the PR2 use case, the salient characteristics of which included:

- a single robot;
- workstation-class computational resources on board;
- **no real-time requirements**
- excellent network connectivity (either wired or close-proximity high-bandwidth wireless);
- applications in research, mostly academia;
- maximum flexibility, with nothing prescribed or proscribed (e.g., “we don’t wrap your main()”).

Some last reminders about ROS design (2)

- ROS satisfied the PR2 use case, but also overshot by becoming useful on a surprisingly wide variety of robots.
- Today we see ROS used not only on the PR2 and robots that are similar to the PR2, but also on wheeled robots of all sizes, legged humanoids, industrial arms, outdoor ground vehicles (including self-driving cars), aerial vehicles, surface vehicles, and more.





ROS Best practice (1)

You should follow all the rules and recommendations learned in previous software programming/engineering courses and projects!

For example:

- Check for available solutions (packages, nodes, …),
- Understand design pattern underneath successful and working packages/stack
- Define common units, please refer to Standard Units of Measure and Coordinate Conventions.
- Use a version control system
- Test your code (push only tested code on the shared repository)

ROS Best practice (2)

Messages

- Check if common messages are already available: https://github.com/ros/common_msgs
- Create separate packages that contain only messages, services and actions (separation of interface and implementation).
- Do not define a new msg/srv/action definition for each topic/service/action!
- Complex messages are built through composition (e.g. geometry_msgs/PoseWithCovarianceStamped).
- Try to avoid building messages that tend to not get completely filled out.

ROS Best practice (3)

Package Organization

- The overhead of a ROS package is not large. Define separate packages wherever they make sense. Pay attention to avoid over
- The package dependency graph must be acyclic, i.e. no package may depend on another that directly or indirectly depends on it. → Avoid combining nodes that pull in mutually unneeded dependencies and are often used separately (to eliminate unnecessary build overhead).
- **Create separate packages that contain only messages, services and actions**
- Group related packages in stacks.
- Package Names → Choose the name carefully:
  - They are messy to change later.
  - Package names are global to the entire ROS ecosystem.
  - Try to pick names that will make sense to others who may wish to use your code.

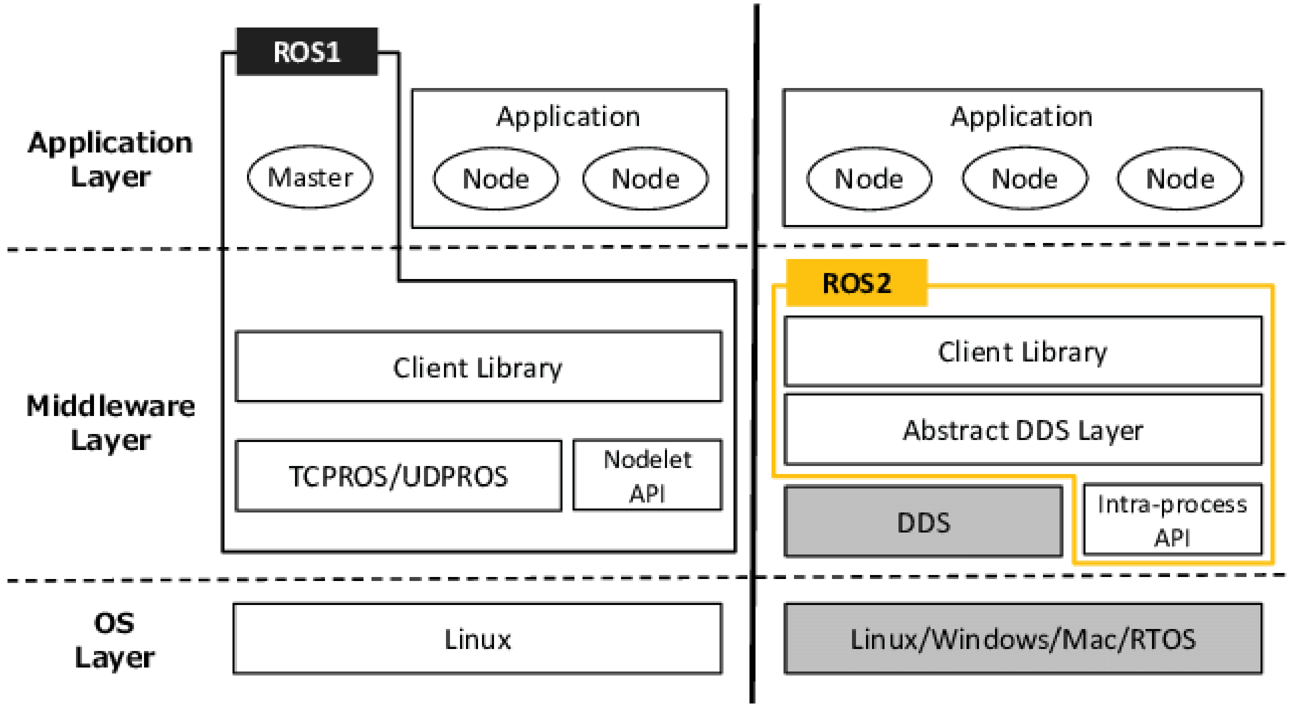
Please check also: https://github.com/leggedrobotics/ros_best_practices/wiki

New use cases for ROS

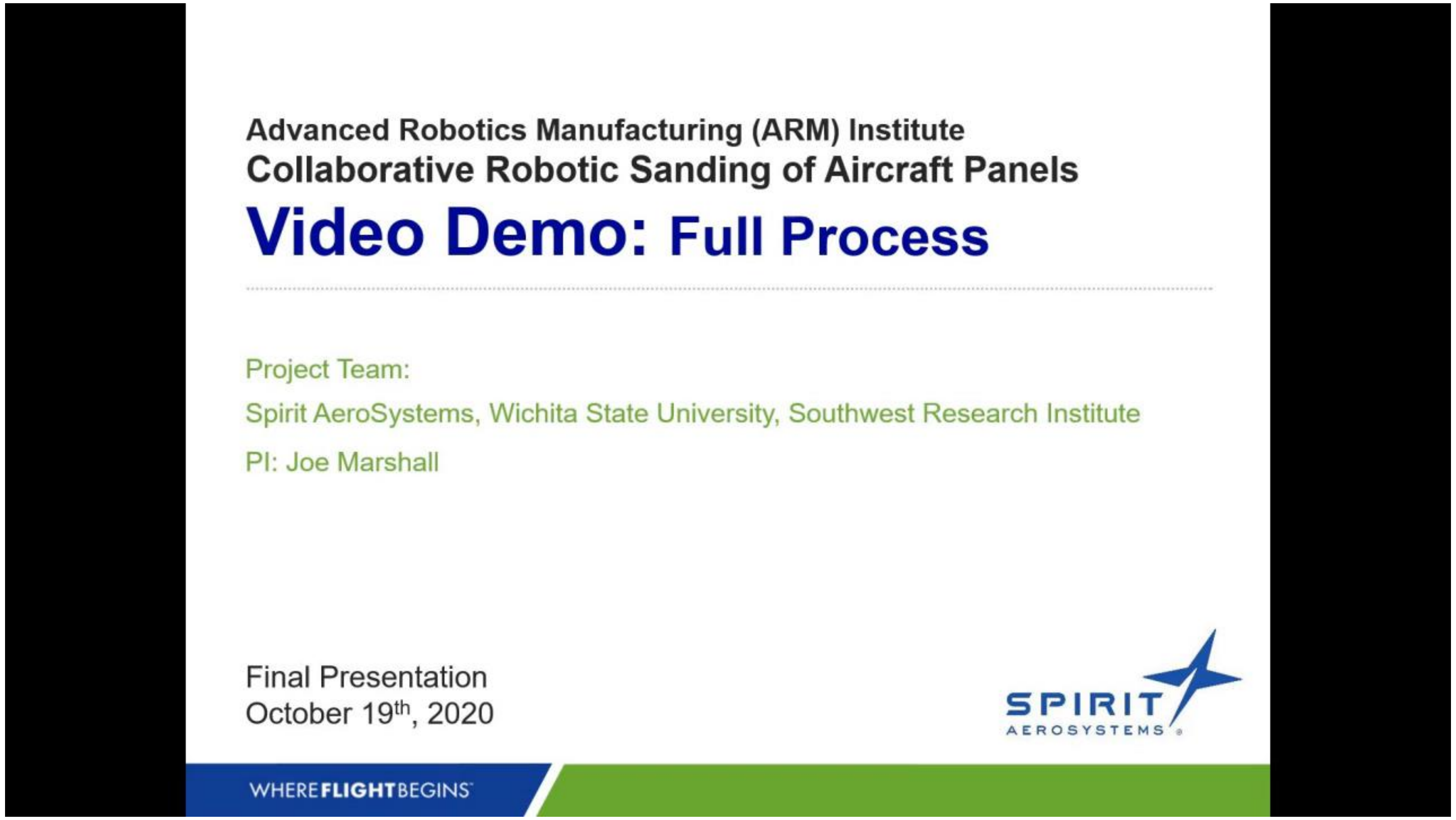
- **Teams of multiple robots**: while it is possible to build multi-robot systems using ROS today, there is no standard approach,.
- **Small embedded platforms**: we want small computers, including “bare-metal” micro controllers, to be first-class participants in the ROS environment, instead of being segregated from ROS by a device driver.
- **Real-time systems**: we want to support real-time control directly in ROS, including inter-process and inter-machine communication.
- **Non-ideal networks**: we want ROS to behave as well as is possible when network connectivity degrades, from poor-quality WiFi to ground-to-space communication links.
- **Production environments**: while it is vital that ROS continue to be the platform of choice in the research lab, we want to ensure that ROS-based lab prototypes can evolve into ROS-based products suitable for use in real-world applications.
- **Prescribed patterns for building and structuring systems**: while we will maintain the underlying flexibility that is the hallmark of ROS, we want to provide clear patterns and supporting.

ROS 2 Characteristics

- **Realtime support possible**
- Multi-platform support: Linux, Windows, OSX
- Production-ready framework based on industry feedback of ROS 1
- DDS (Data Distribution service): open communication standard



ROS 2 Example



Advanced Robotics Manufacturing (ARM) Institute

Collaborative Robotic Sanding of Aircraft Panels

Video Demo: Full Process

Project Team:

Spirit AeroSystems, Wichita State University, Southwest Research Institute

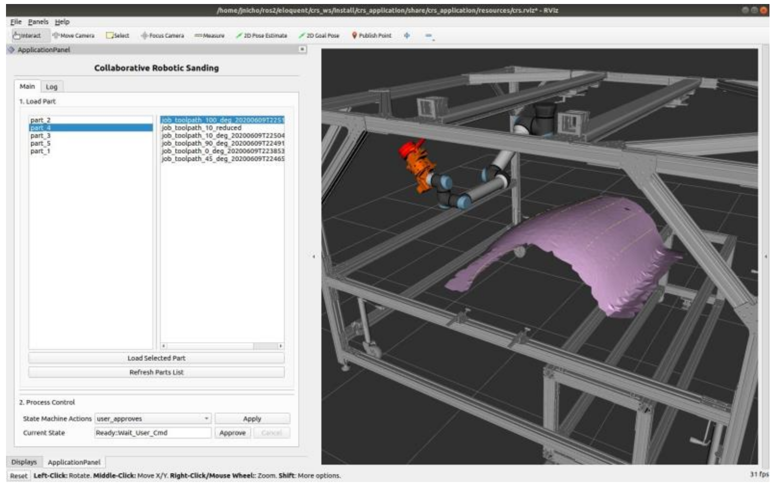
PI: Joe Marshall

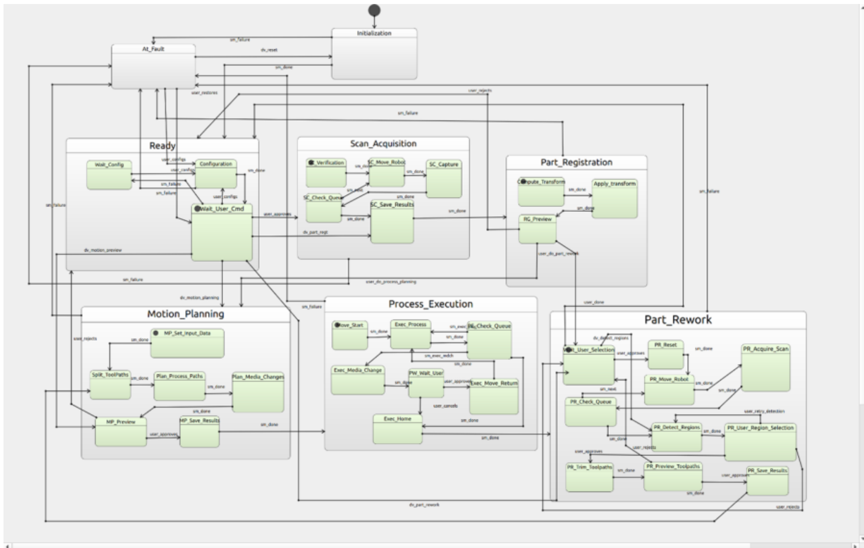
Final Presentation

October 19th, 2020

SPIRIT AEROSYSTEMS

WHERE FLIGHT BEGINS





https://github.com/swri-robotics/collaborative-robotic-sanding

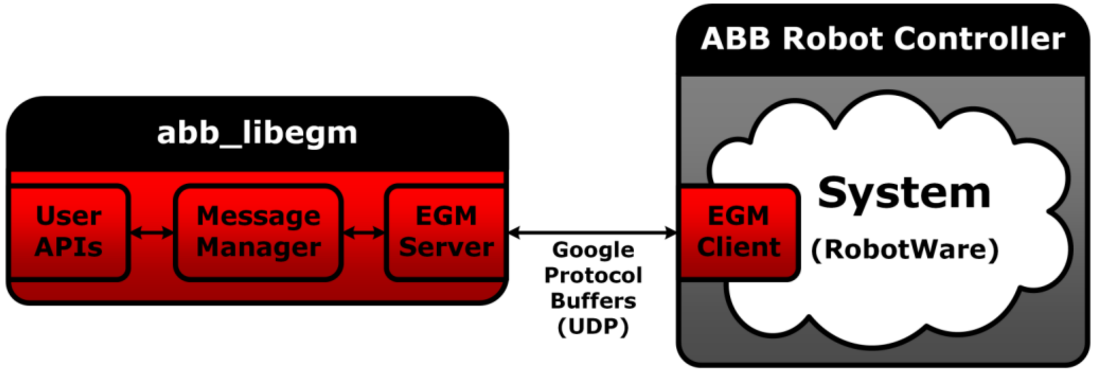
Previous project examples

ALTAIR robotics lab

ICE Lab INDUSTRIAL COMPUTER ENGINEERING

Previous project example: Using ABB EGM library (1)

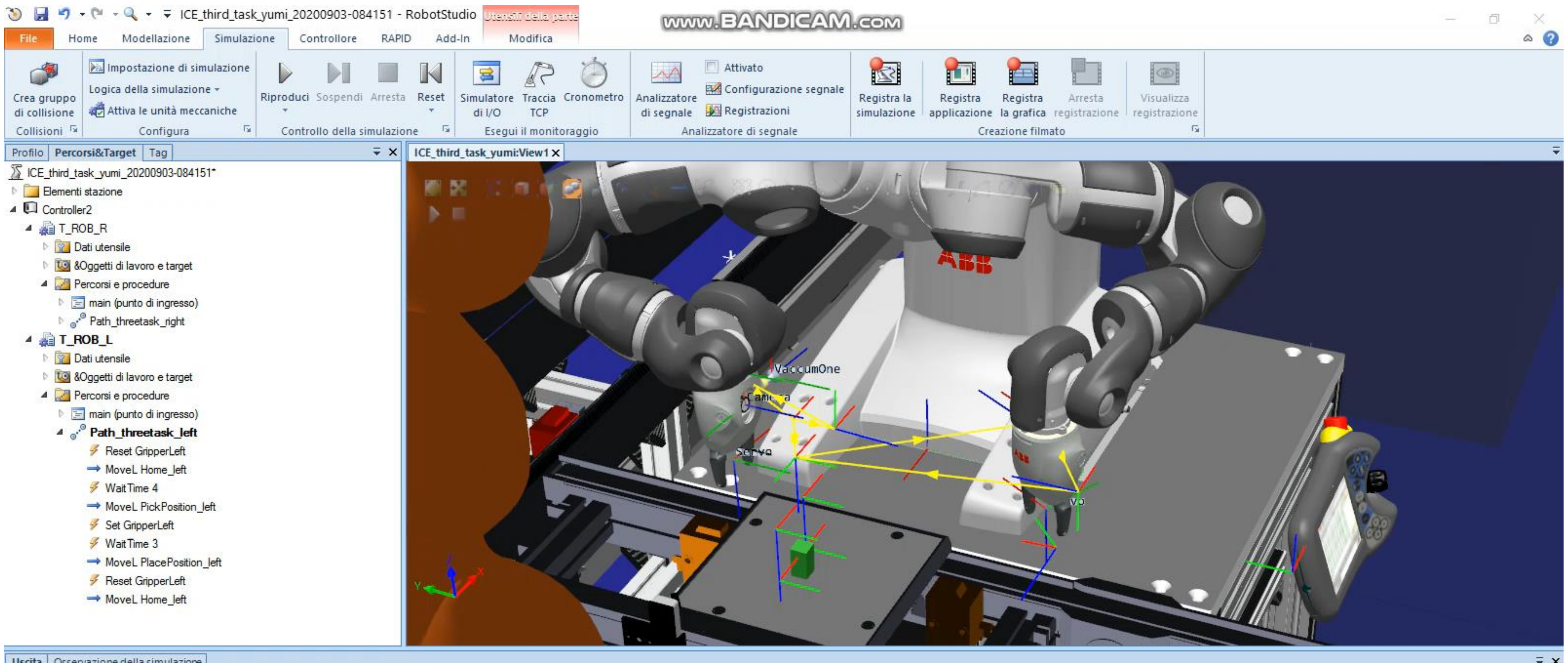
- abb_libegm is a C++ communication library for facilitating, and easing, the use of the Externally Guided Motion (EGM) interface of ABB robot controllers.
- The library helps with setting up communication channels, parsing EGM messages, as well as providing user APIs



Courtesy of Enrico Sgarbanti and in collaboration with Nimbo

Previous project example: Using ABB EGM library (2)

Previous project example: Offline Programming of a Pick and place task with ABB Yumi



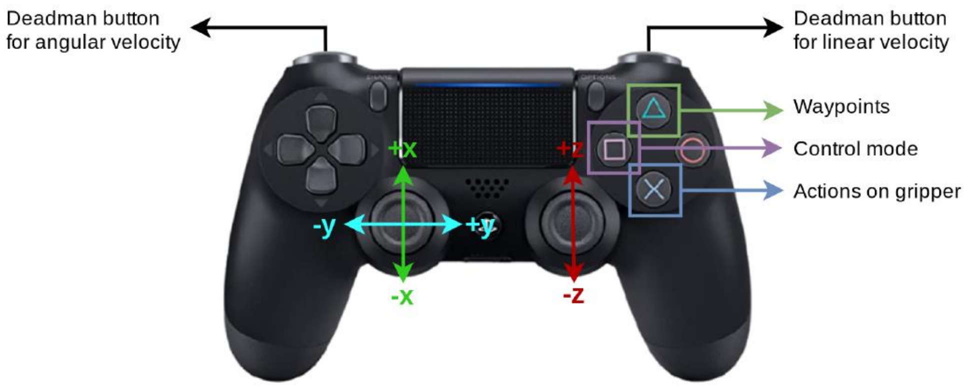
```
ICE_third_task_yumi_20200903-084151*
  Elementi stazione
  Controller2
    T_ROB_R
      Dati utensile
      &Oggetti di lavoro e target
      Percorsi e procedure
        main (punto di ingresso)
        Path_threetask_right
    T_ROB_L
      Dati utensile
      &Oggetti di lavoro e target
      Percorsi e procedure
        main (punto di ingresso)
        Path_threetask_left
          Reset GripperLeft
          MoveL Home_left
          WaitTime 4
          MoveL PickPosition_left
          Set GripperLeft
          WaitTime 3
          MoveL PlacePosition_left
          Reset GripperLeft
          MoveL Home_left
```

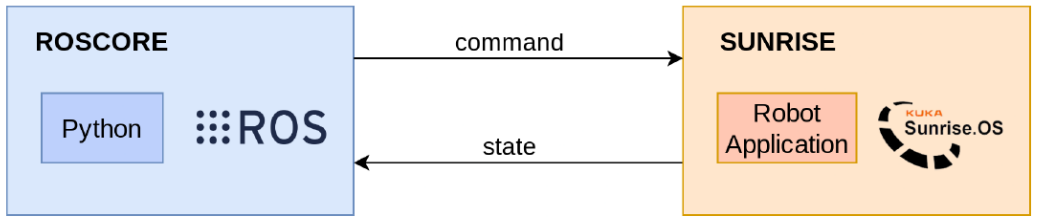
Courtesy of Leonardo Testolin and in collaboration ICE lab

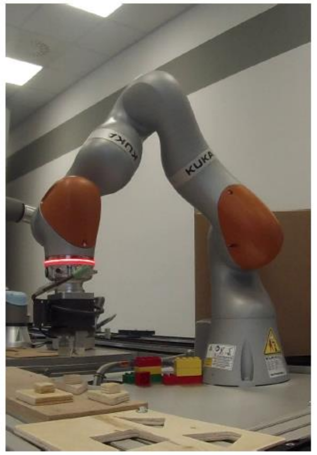
Previous project example: Offiline Programming of a Pick and place task with ABB Yumi

Courtesy of Leonardo Testolin and in collaboration ICE lab

Previous project example: Using KUKA IIWA

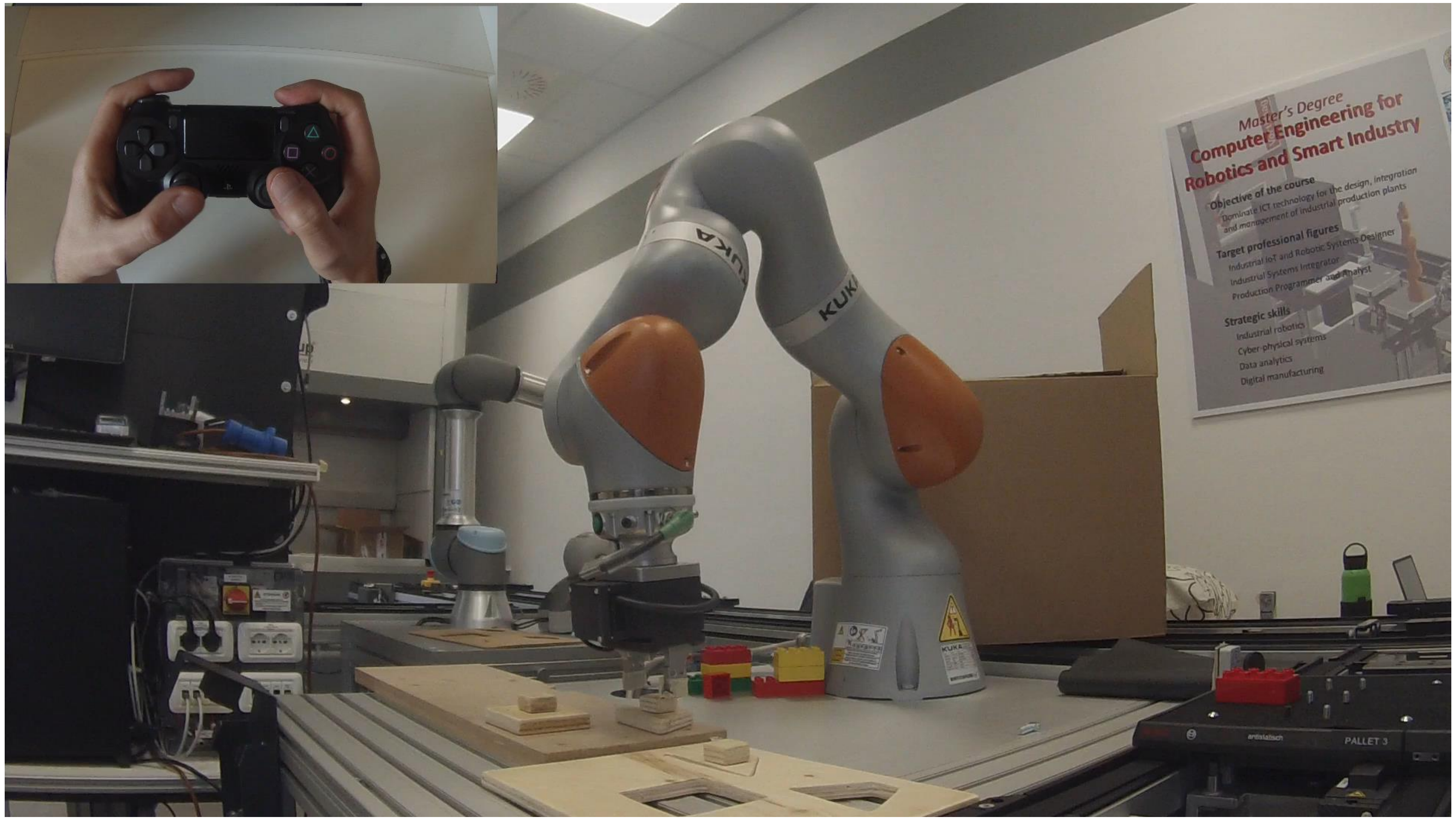

Deadman button for angular velocity
Deadman button for linear velocity
Waypoints
Control mode
Actions on gripper
+x
-x
-y
+y
+z
-z





Courtesy of Michele Penzo and in collaboration ICE lab

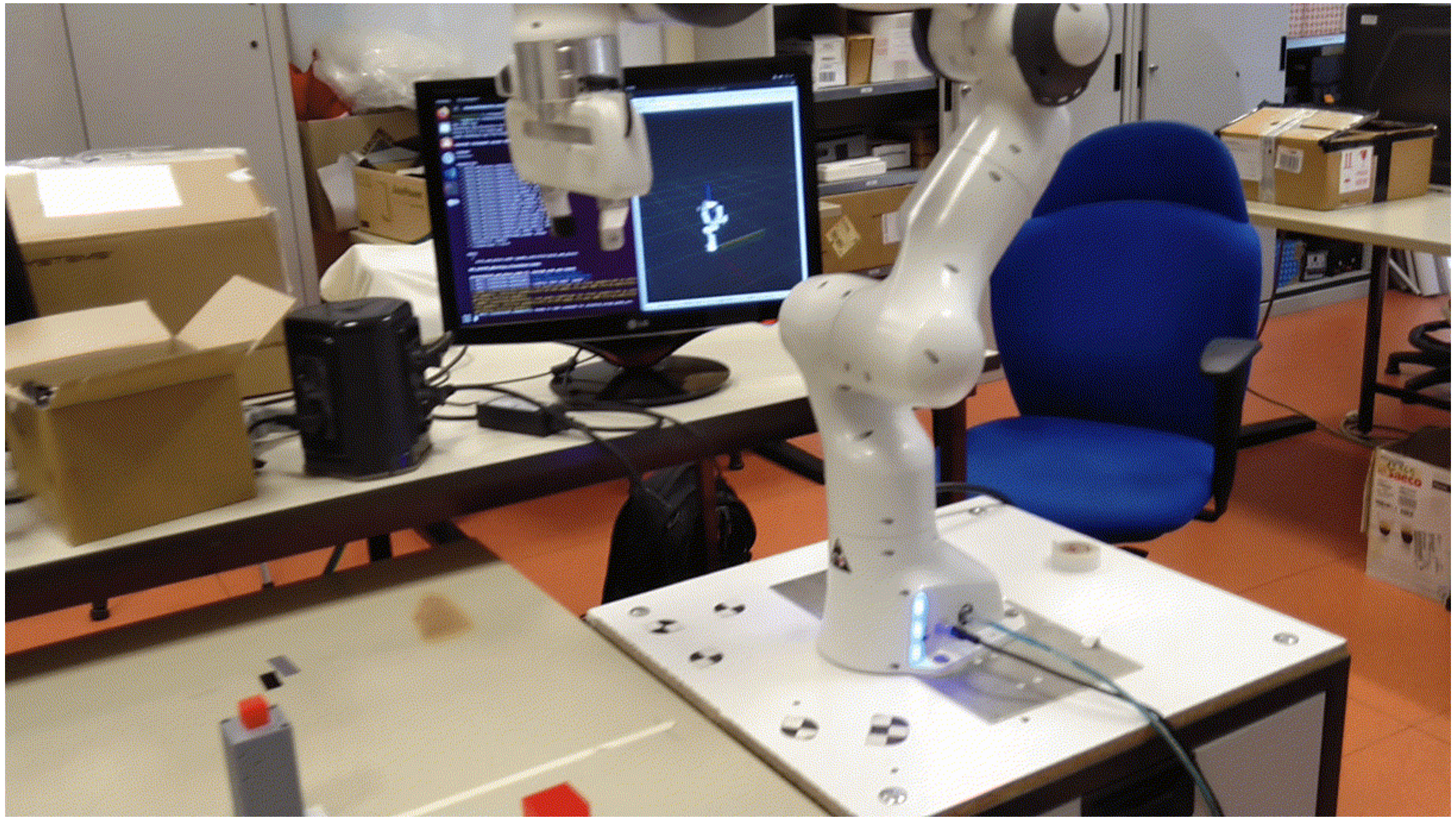
Previous project example: Using KUKA IIWA

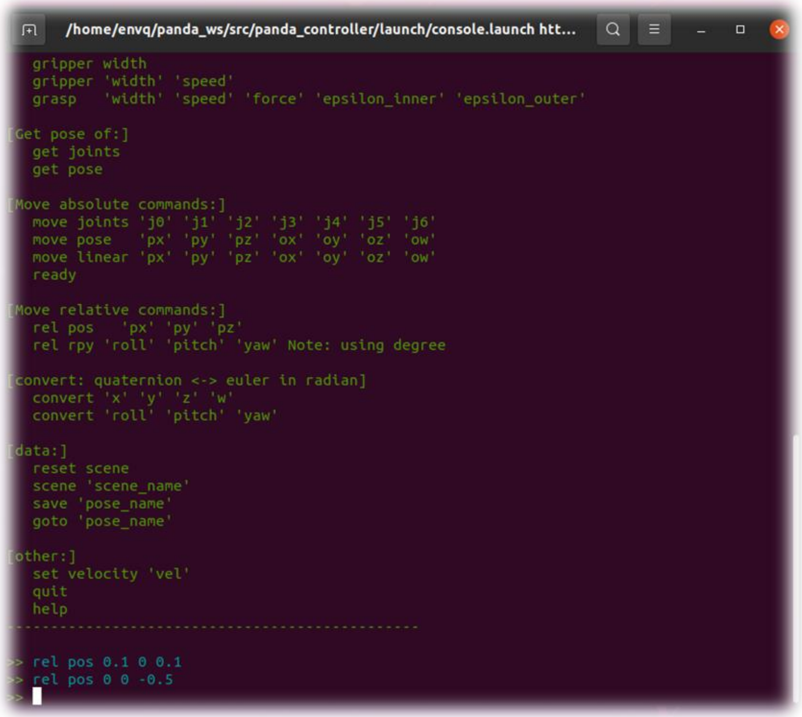


Courtesy of Michele Penzo and in collaboration ICE lab

ICE Lab
INDUSTRIAL COMPUTER ENGINEERING

Previous project example: Programming pick and place task with C++ and ROS



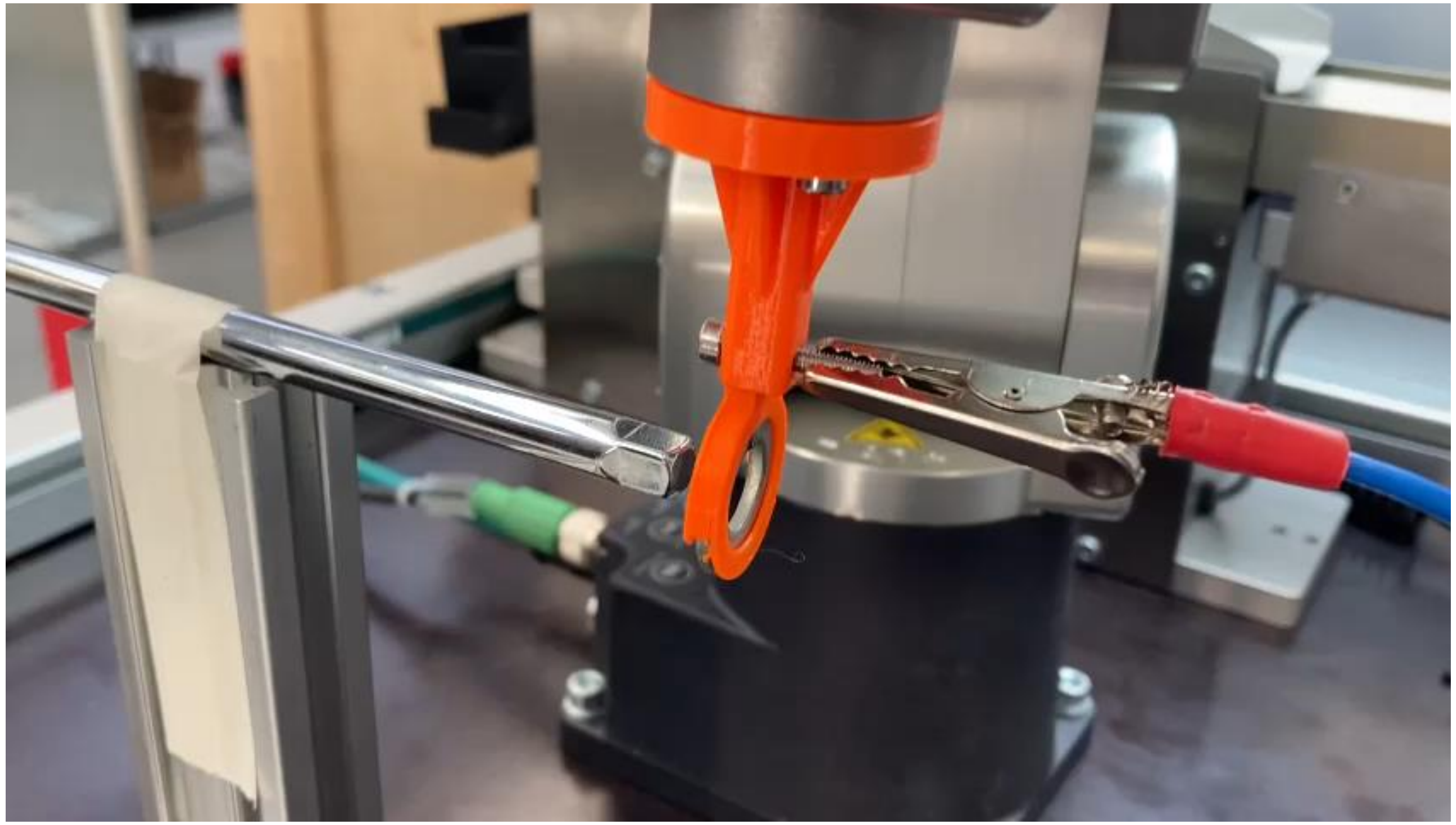


/home/envq/panda_ws/src/panda_controller/launch/console.launch htt...

```
  gripper width
  gripper 'width' 'speed'
  grasp   'width' 'speed' 'force' 'epsilon_inner' 'epsilon_outer'

[Get pose of:]
  get joints
  get pose

[Move absolute commands:]
  move joints 'j0' 'j1' 'j2' 'j3' 'j4' 'j5' 'j6'
  move pose   'px' 'py' 'pz' 'ox' 'oy' 'oz' 'ow'
  move linear 'px' 'py' 'pz' 'ox' 'oy' 'oz' 'ow'
  ready

[Move relative commands:]
  rel pos   'px' 'py' 'pz'
  rel rpy 'roll' 'pitch' 'yaw' Note: using degree

[convert: quaternion <-> euler in radian]
  convert 'x' 'y' 'z' 'w'
  convert 'roll' 'pitch' 'yaw'

[data:]
  reset scene
  scene 'scene_name'
  save 'pose_name'
  goto 'pose_name'

[other:]
  set velocity 'vel'
  quit
  help
-----------------------------------------------

>> rel pos 0.1 0 0.1
>> rel pos 0 0 -0.5
>>
```

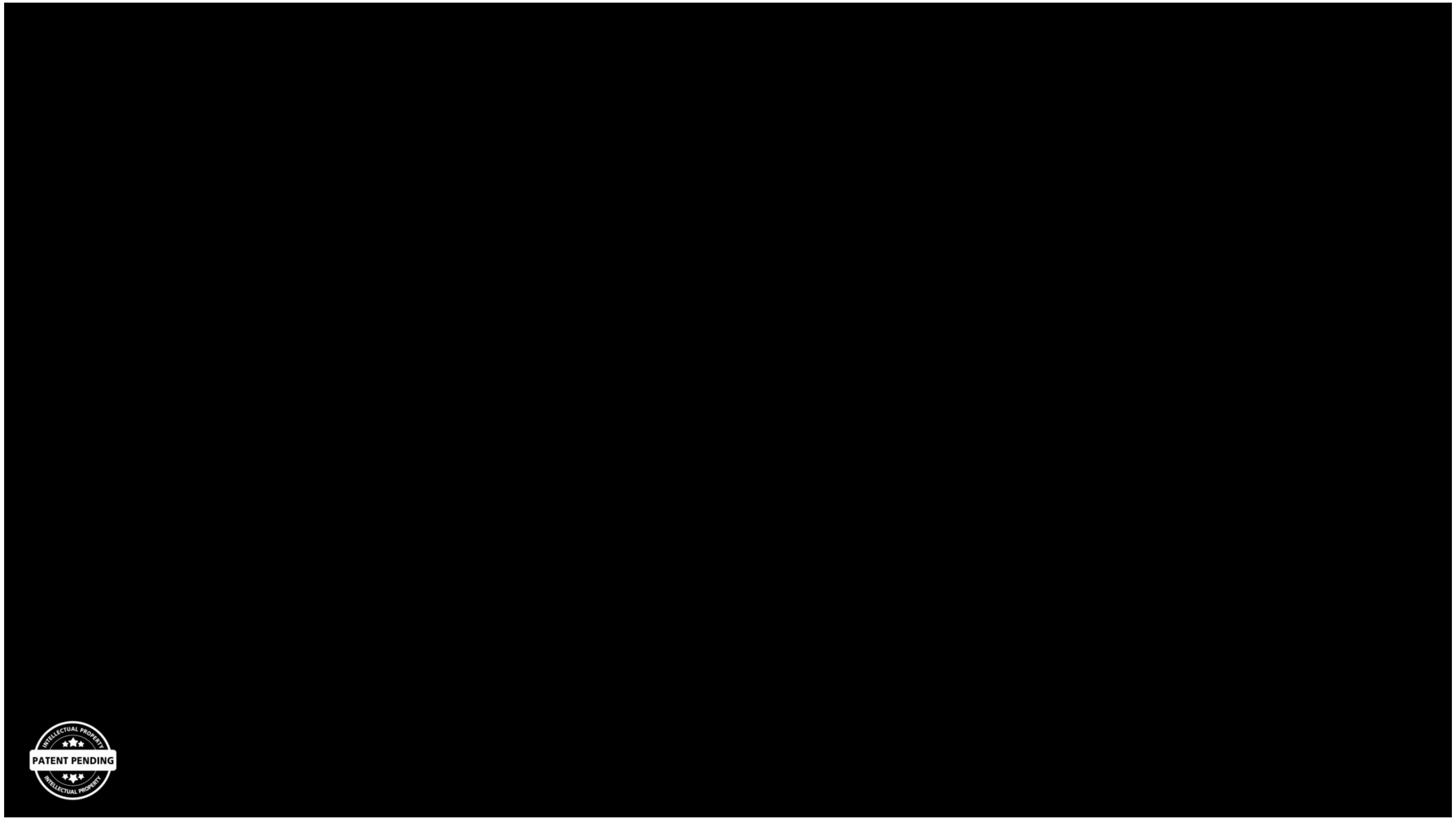
Courtesy of Enrico Sgarbanti and in collaboration Altair lab

Previous project example: esoteric robotic applications (of industrial relevance)



Courtesy of Enrico Sgarbanti and in collaboration Nimbo srl

Internship, thesis and other opportunities are available with local companies



NIMBO
Our Knoledge meets Your Experience.

BLM GROUP

ALTAIR robotics lab

Final Project Rules

- You can carry out the final project of the course in small groups (2 or 3 people). Depending on the number of group members, the difficulty of the project can be changed
- Each student must complete a final project for only one of the two modules of the course. For the other module you must prove that you have done the assigned homework.
- Course grade will be based on the evaluation of the final project + up 2/30 bonus points obtained from the evaluation of homework of the other module.
- The verification of the project and the homework will be done through two oral presentations (possibly carried out remotely) in which questions can be asked to verify the design / implementation choices. Each presentation will be carried out with the teacher of the specific module.
- Module B homework verification is individual for each student.

Module B required Homework

Homework B1: Installing and using a ROS workstation

Homework B2: Use MoveIt to perform motion planning of the e.Do in virtual enviroment

See previous lessons for detailed homework description

Module B - Final Project proposal 1

<u>Programming at least two manufacturing tasks using the real COMAU e.Do :</u>

- Learn the standard programming environment available for e.Do manipulator (open-source based on ROS, see previous lesson)
- Learn how to integrate the manipulator in a workcell consisting of other external sensors/actuators
- <u>Project could start only in the second half of July 2022</u>

COMAU

ICE Lab INDUSTRIAL COMPUTER ENGINEERING

e.Do

Module B - Final Project proposal 2

FULLY VIRTUAL REMOTE PROJECT

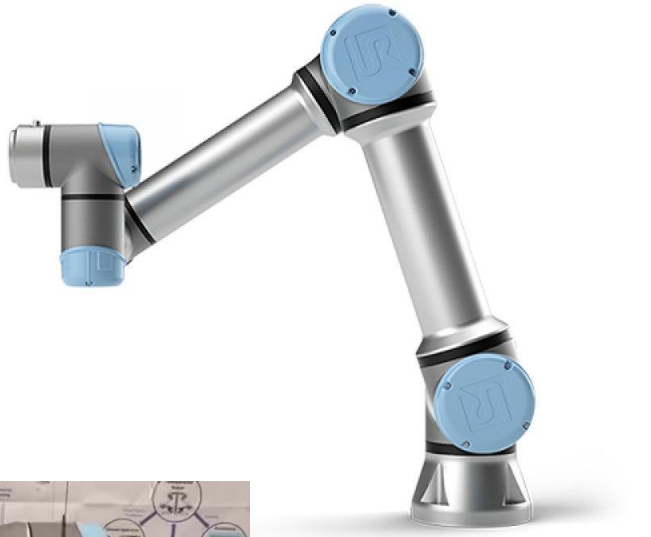
Programming a flexible industrial cell based on ROS:

- Geometric modeling of a e.Do mounted on a table (cell available in Odino lab), able to perform at least 3 different manufacturing tasks:
  - Object Pick and place,
  - Glue sealing (following a cartesian path with trajectory constraints in orientation and speed)
  - Quality inspection
- Motion planning in MoveIt
- Detailed documentation is required
- Optional: Task programming in Moveit
- Optional: User interface based on Rqt

Module B - Final Project proposal 3

<u>Programming at least two manufacturing tasks using UR5e (Not based on ROS):</u>

- Learn the standard programming environment available for Universal robot cobots
- Use offline tools to design the tasks and verify your code before testing on the real robot
- Learn how to operate the real robot (with support of ICE Laboratory personnel)
- Detailed documentation is required



ICE Lab
INDUSTRIAL COMPUTER ENGINEERING

Module B - Final Project proposal 4

<u>Learn how to program the ABB Yumi collaborative robot:</u>

- Learn how to use ABB Robotic Studio
- Using offline programming functionalities program at least two tasks where both manipulator are needed.
- Test the previous point on the real robot available in ICE lab (with support of ICE Laboratory personnel)

- Optional: learn how to use collaborative capabilities of the Yumi robot
- Optional: program at least one collaborative task

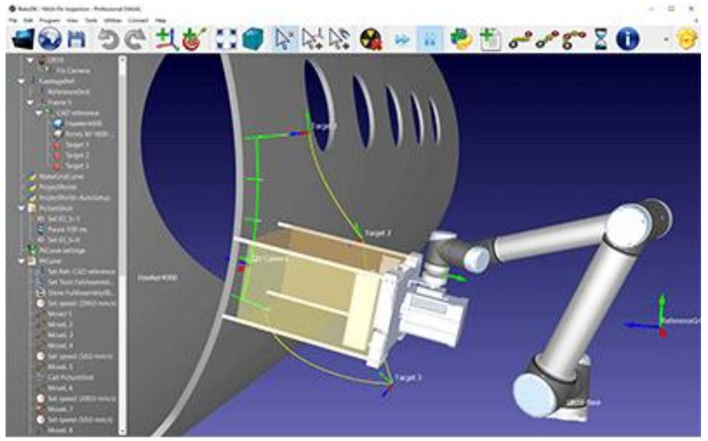
- Possible extensions to vision-based tasks

ICE Lab
INDUSTRIAL COMPUTER ENGINEERING

Module B - Final Project proposal 5

<u>Learn how to use RoboDK:</u>

- Learn how to use RoboDK offline programming software
- Document the main characteristics of the SW, in particular w.r.t. course contents (e.g., geometric modeling, calibration, motion planning, programming)
- Program at least two different task with at least two different manipulators
- Optional: Using post processor generate native program to replicate the task on a real manipulator available in ICE lab (TBD, could be e.Do)
- Optional: Using Robot driver replicate a simple task on a real manipulator available in ICE lab



Module B – Students Final Project proposals

If you are interested in more complex project (and/or want to cover other training activities in your career) you are welcome to propose your idea, in particular if:

- You are interested in doing a joint project between Module A and B
- You are interested in doing a joint project with other teaching activities (e.g., Robotic Vision and Control, Mobile Robotics, pHRI, …)
- You are interested in working with other manipulators, or focusing on some specific aspect (e.g. robot virtual simulation, learning from human demonstration,)

- **You are interested in high level robot programming ☺**

Questions?

Diego Dall’Alba diego.dallalba@univr.it
Department of Computer Science – University of Verona
Altair Robotics Lab