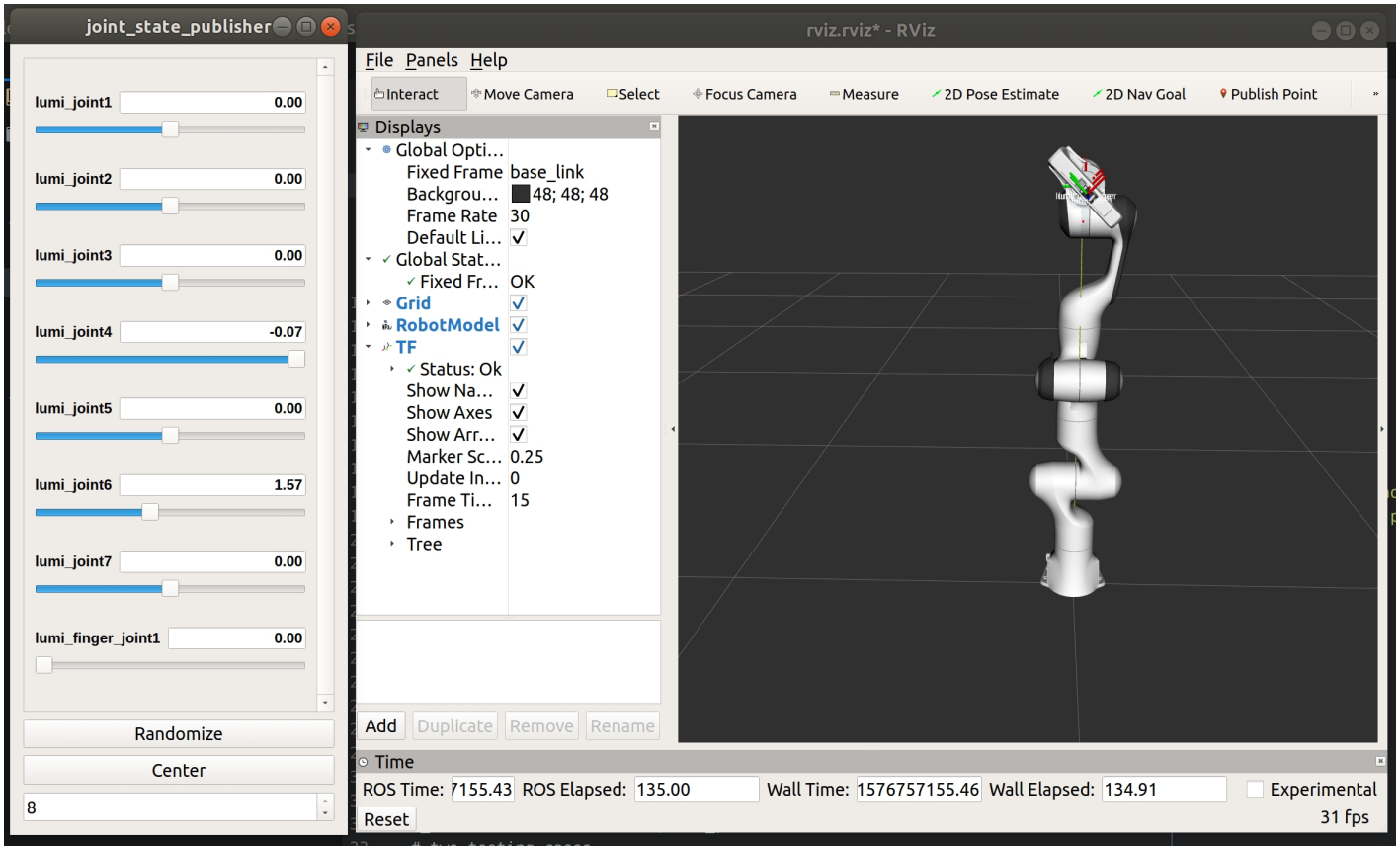
Introduction to ROS Control

Tran Nguyen Le

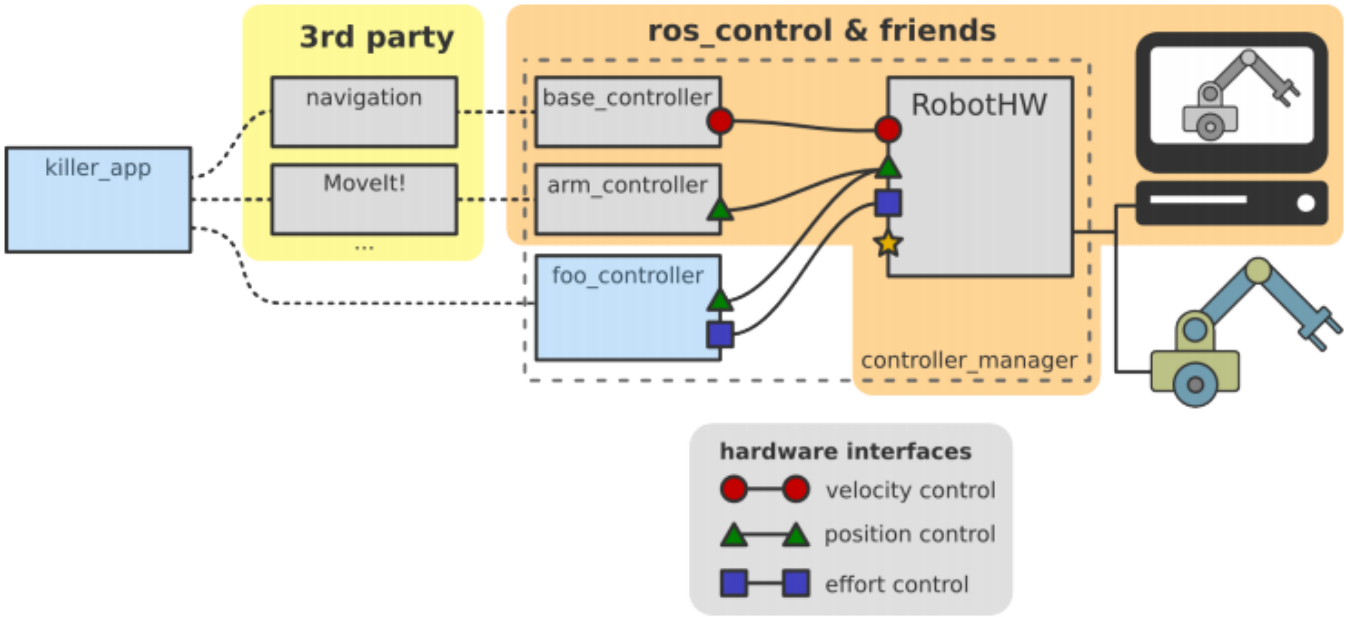
What for?

- Gravity!
- How to keep robot stable?
- Control all joints of the robot with some certain values.



ROS control

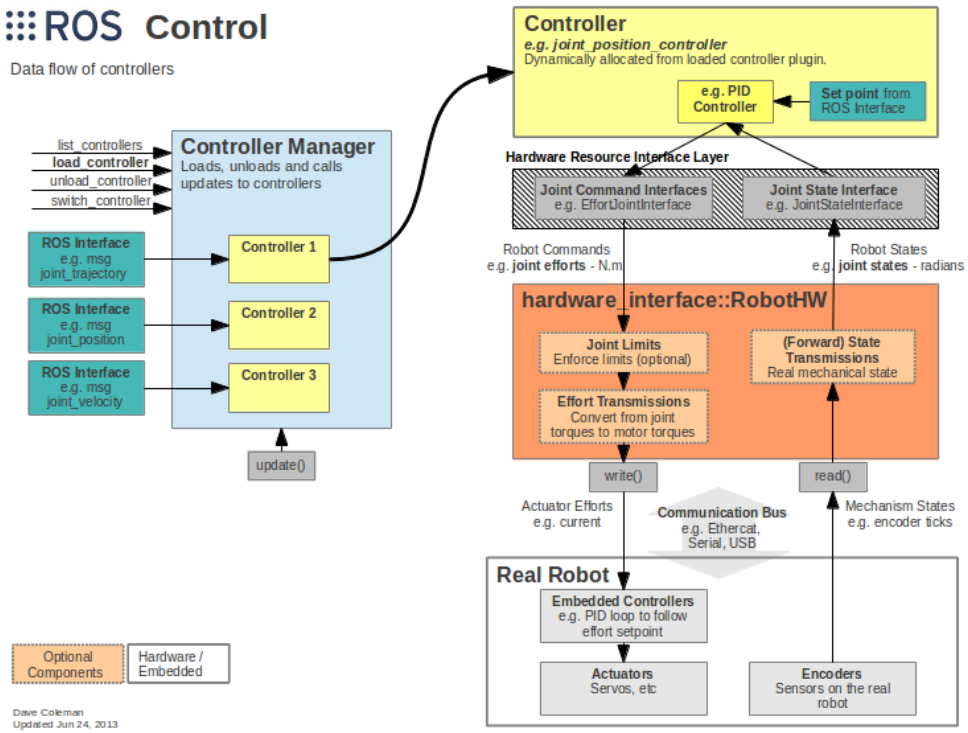
- ROS Control[1] is a full ecosystem inside ROS that allows you to control joints/actuators of robots either in simulation or real world.



[1] http://wiki.ros.org/ros_control

ROS Control Component

- **The Controller Manager:** Stack of controllers. It handles the load/unload and execution of controllers.
- **The Robot Controller:** The actual controllers that will logically control all joints.
- **The Robot Hardware Inferface (RobotHW):** allows the connection to the real hardware. This is created specifically for each robot.



Type of ROS Controllers

The ros_control packages provide a set of controller plugins to interact in different ways with the joints of the robot

- **position_controllers**
  - **joint_position_controller**: This plugin accepts position values as input.
- **velocity_controllers**
  - **joint_velocity_controller**: This plugin accepts velocity values as input.
- **effort_controllers**
  - **joint_position_controller**: This plugin accepts position values as input.
  - **joint_velocity_controller**: This plugin accepts velocity values as input.
  - **joint_effort_controller**: This plugin accepts effort (torque) values as input.

Hardware Interface

After the controllers are well coded, their output is sent to the hardware interface. The hardware interface is a software representation of the robot and its abstract hardware. **In short, the hardware interface acts as a bridge between the controller and the robot or the simulator.**
Some available hardware interfaces:

- Joint Command Interfaces
  - Effort Joint Interface
  - Velocity Joint Interface
  - Position Joint Interface
- Joint State Interfaces

Writing your own controller

```
#include <controller_interface/controller.h>
#include <hardware_interface/joint_command_interface.h>
#include <pluginlib/class_list_macros.h>

namespace controller_ns{

class PositionController : public controller_interface::Controller<hardware_interface::EffortJoi
ntInterface>
{
public:
  bool init(hardware_interface::EffortJointInterface* hw, ros::NodeHandle &n)
  {
    // get joint name from the parameter server
    std::string my_joint;
    if (!n.getParam("joint", my_joint)){
      ROS_ERROR("Could not find joint name");
      return false;
    }

    // get the joint object to use in the realtime loop
    joint_ = hw->getHandle(my_joint);  // throws on failure
    return true;
  }

  void update(const ros::Time& time, const ros::Duration& period)
  {
    double error = setpoint_ - joint_.getPosition();
    joint_.setCommand(error*gain_);
  }

  void starting(const ros::Time& time) { }
  void stopping(const ros::Time& time) { }

private:
  hardware_interface::JointHandle joint_;
  static const double gain_ = 1.25;
  static const double setpoint_ = 3.00;
};
PLUGINLIB_DECLARE_CLASS(package_name, PositionController, controller_ns::PositionController, con
troller_interface::ControllerBase);
}//namespace
```

Demo Time