Real-Time Motion Control in ROS: Uniting Machinekit HAL with Tormach's ZA6 Robot

John Morris

Software Engineering

Brandon Duarte

Software Engineering

TORMACH®

TORMACH®

Headquartered in Madison WI

100% employee-owned

Our Customers

Hobbyist

Small businesses

Researchers

ZA6 Robot

6kg industrial arm

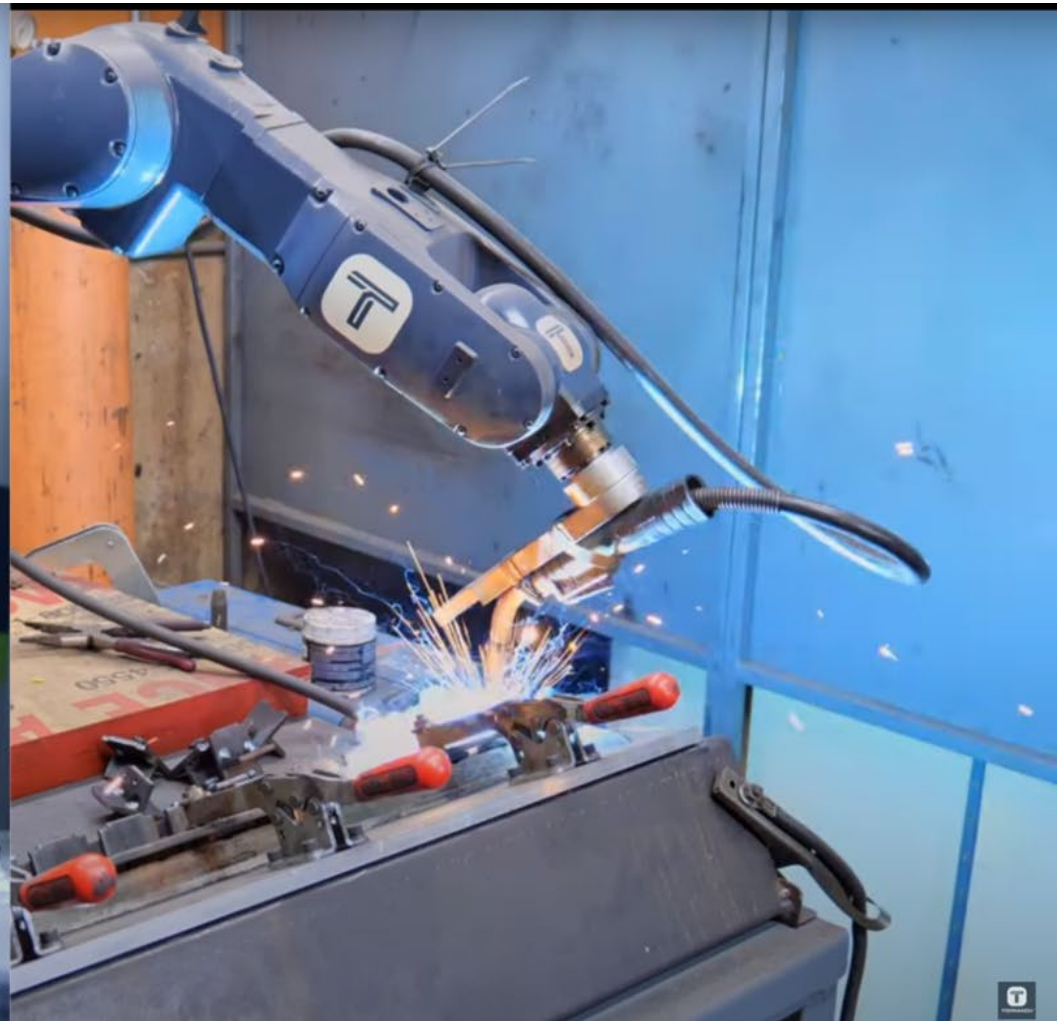
Natural fit with machine tool products

Powered by ROS and fully open source

"Each new robotic platform we integrate with, we have to evaluate RT expectations and requirements. We would much rather have that solved out of the box and use that engineering time building more sophisticated robots!"

Nathan Brooks, CTO

Machinekit HAL cuts machine control development time



Machinekit HAL: Real Time Threading Environment

Real-time thread programming

Linux RT thread "flavor" support

- RT_PREEMPT
- and/or POSIX (non-RT)
- or Xenomai

Real Time Scheduling

- scheduling algorithm e.g. POSIX `SCHED_FIFO`
- elevated scheduling priority

More real-time programming

RT <-> non-RT communication

- lock-free ring buffers
- shm
- mutexes

Arch-dependent programming

- Atomic 64-bit access
- hi-res timers

Other

- `mlockall()`: VM page faults

Fix real-time jitter sources

OS and hardware tuning

- Intel CPU C-States (power saving)
- CPU frequency scaling (Intel; AMD)

Dedicated RT CPU

- `isolcpus`
- cgroup `cpuset`
- CPU affinity

Machinekit HAL: Modular components for common tasks

Hardware drivers

- EtherCAT
- “hostmot” FPGA f/w
- Beaglebone PRUSS
- Modbus
- GPIO

Motor control

- PID
- PWM generator
- Step/dir generator
- Quadrature

Logic

- and/or/not/xor
- flipflop
- mux/demux
- oneshot
- lut
- timedelay

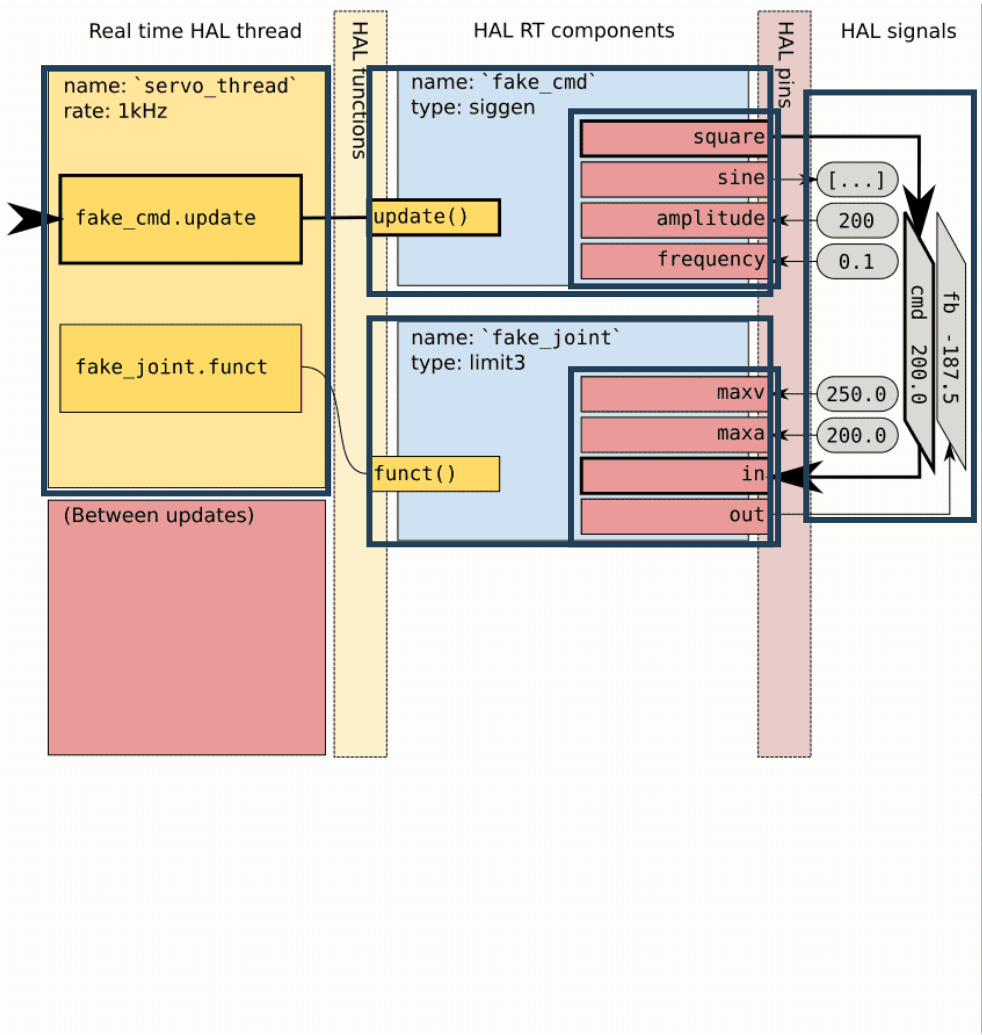
Instrumentation

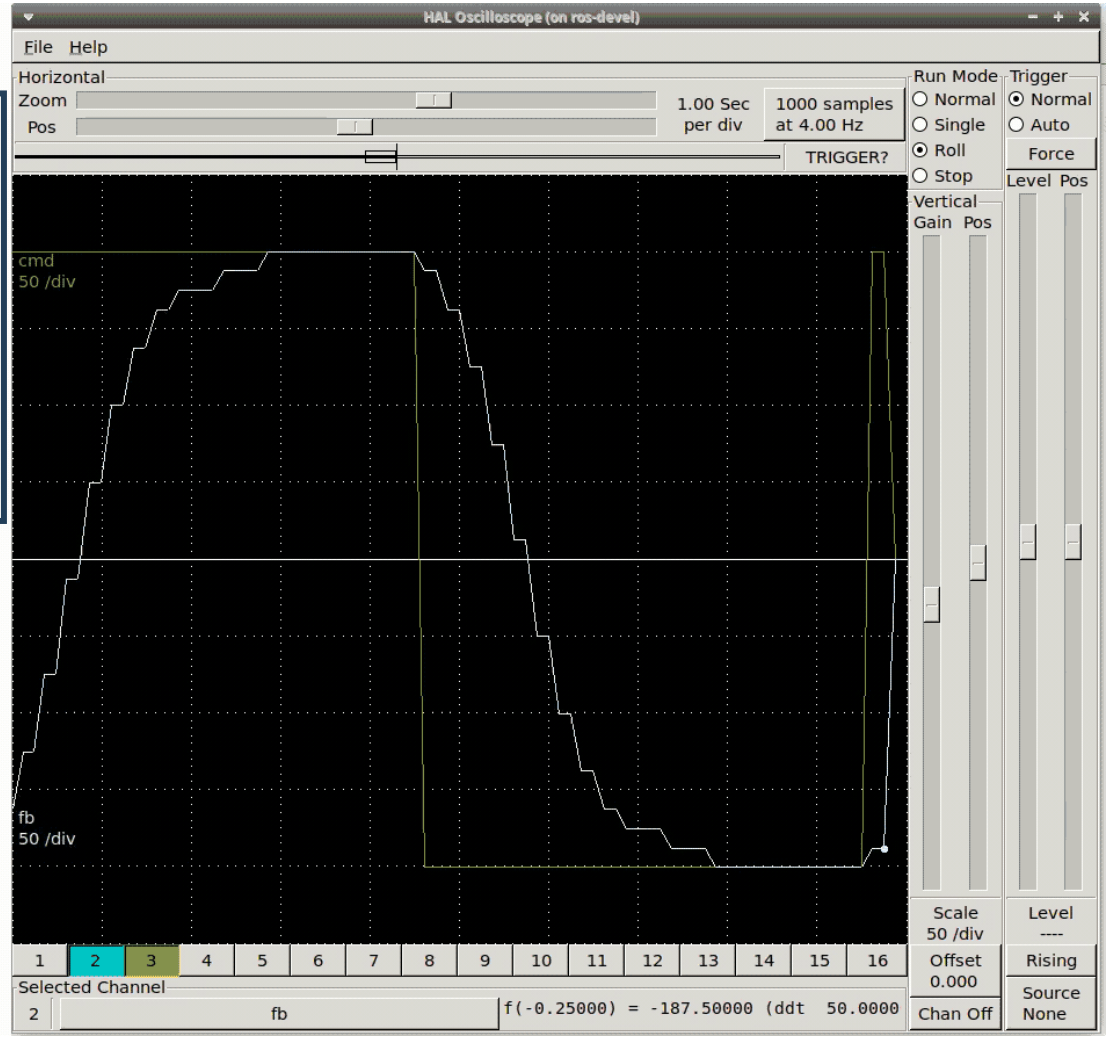
- halscope
- sampler
- streamer
- latencybins
- histobins

Arithmetic & math

- ddt/integrator
- biquad/lowpass/kalman filters
- limit1/limit2/limit3
- offset/scale
- sum/mult
- siggen

Machinekit HAL: Basic concepts





TORMACH®

Machinekit HAL: Basic configuration

1. load “siggen” comp

2. load “limit3” comp

3. net “cmd” & “fb” signals

4. set up real time thread

5. start the scope!

```
# Fake "command":  signal generator component square wave
loadrt siggen names=fake_cmd
setp fake_cmd.amplitude 200
setp fake_cmd.frequency 0.1

# Fake "joint":  limit3 componont
#     output is accel + velocity limited input
newinst limit3 fake_joint
setp fake_joint.maxv 250.0
setp fake_joint.maxa 200.0

# Connect "cmd" and "fb" signals
net cmd fake_cmd.square => fake_joint.in
net fb <= fake_joint.out

# Define 4Hz RT thread, "pokey_thread", add functions and start
loadrt threads name1=pokey_thread period1=250000000
addf fake_cmd.update pokey_thread
addf fake_joint.funct pokey_thread
start

# Start halscope (wait for it to exit, then exit HAL)
loadusr -w halscope
```

Save to `demo.hal` and run:
`halrun -f demo.hal`

Machinekit HAL: Easy C and Python APIs

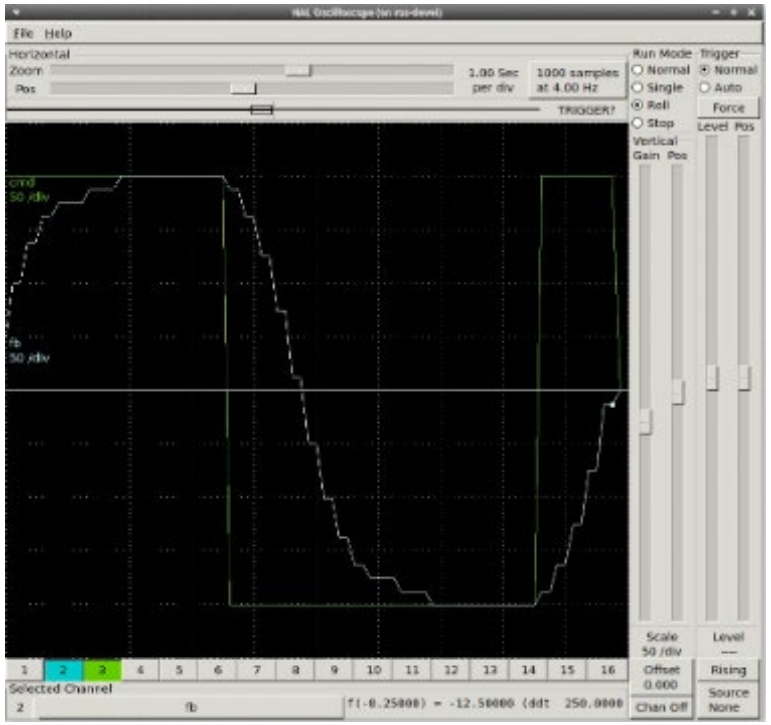
```
component and2 "Two-input AND gate";
pin in bit in0;
pin in bit in1;
pin out bit out
    "TRUE if both in0 & in1 are TRUE, else FALSE";
function _ nofp;
license "GPL";
;;

FUNCTION(_)
{
    out = in0 && in1;
    return 0;
}
```

```
#!/usr/bin/python
import hal, time

# Set up HAL component and mark it "ready"
h = hal.component("and2_py")
h.newpin("in0", hal.HAL_FLOAT, hal.HAL_IN)
h.newpin("in1", hal.HAL_FLOAT, hal.HAL_IN)
h.newpin("out", hal.HAL_FLOAT, hal.HAL_OUT)
h.ready()

try:
    while 1:
        # Main update loop
        time.sleep(1)
        h["out"] = h["in0"] and h["in1"]

except KeyboardInterrupt:
    raise SystemExit
```

Machinekit HAL: Tools for inspecting running systems

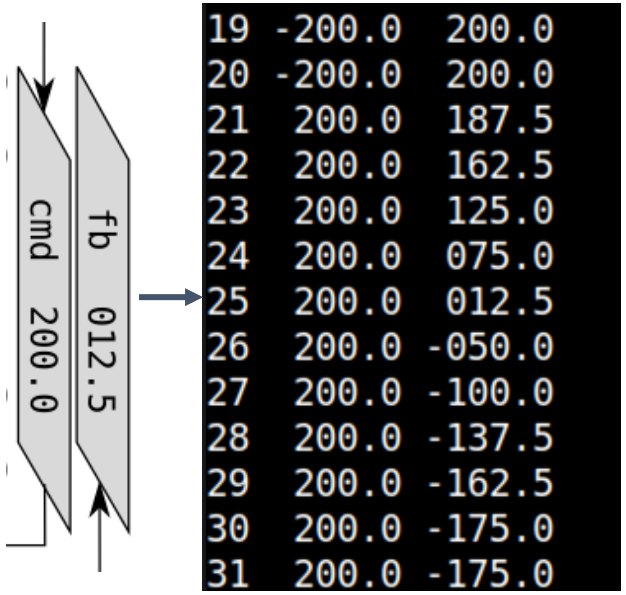
halscope

visualize live data



halsampler

data logging



halcmd

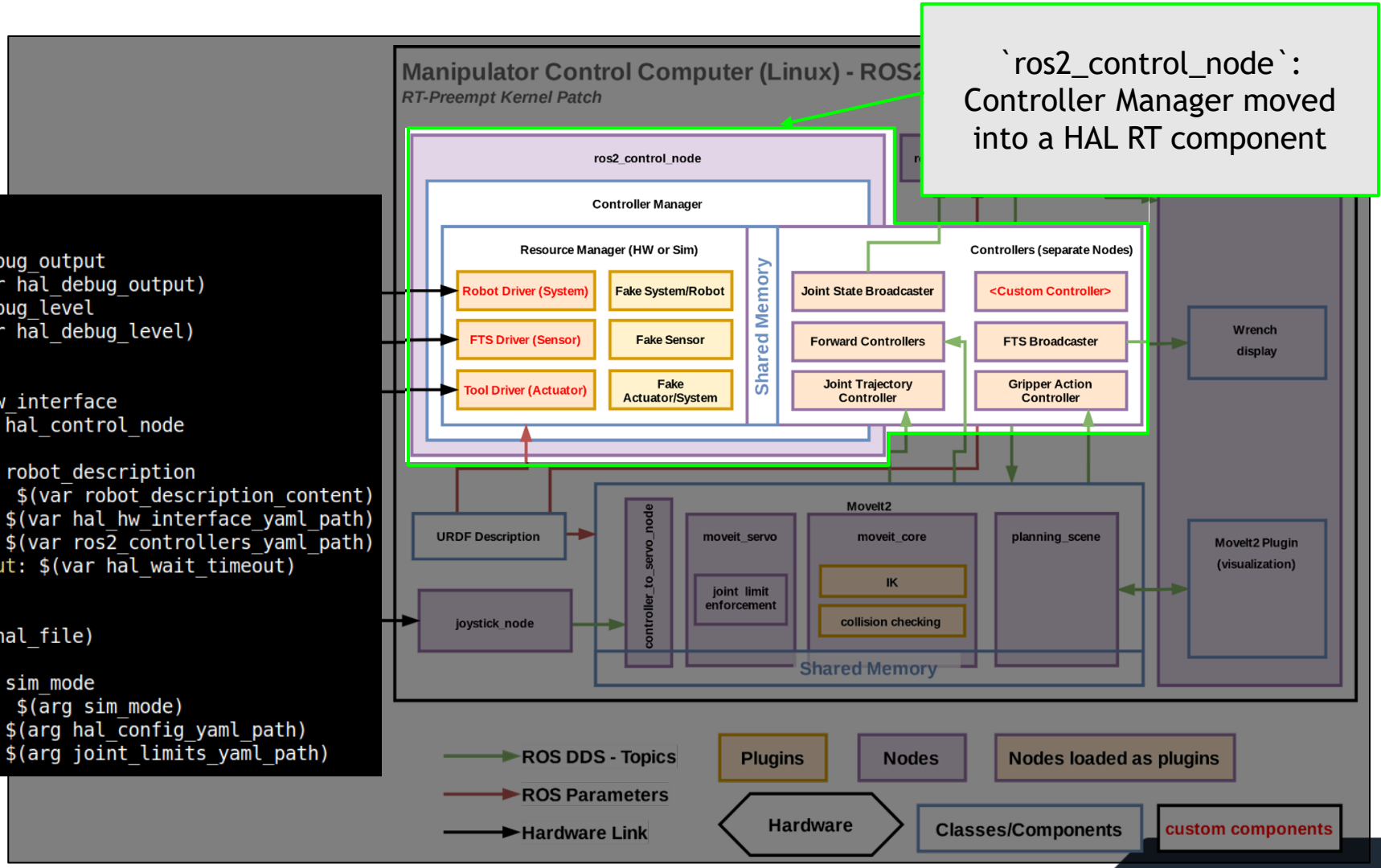
update live system

```
$ halcmd show pin fake_cmd.frequency
Component Pins:
  Comp   Inst Type  Dir         Value  Name
    78        float IN            0.1  fake_cmd.frequency

$ halcmd show pin fake_cmd.amplitude
Component Pins:
  Comp   Inst Type  Dir         Value  Name
    78        float IN            200  fake_cmd.amplitude

$ halcmd show sig cmd
Signals:
Type          Value  flags Name  linked to:
float           200  --    cmd
                                 <== fake_cmd.square
                                 ==> fake_joint.in

$ halcmd show pin fake_joint.maxv
Component Pins:
  Comp   Inst Type  Dir         Value  Name
    93     95 float IN            250  fake_joint.maxv

$ halcmd show pin fake_joint.maxa
Component Pins:
  Comp   Inst Type  Dir         Value  Name
    93     95 float IN            200  fake_joint.maxa

$ halcmd show sig fb
Signals:
Type          Value  flags Name  linked to:
float          -200  --    fb
                                 <== fake_joint.out
```

`hal_ros_control` cuts robot control development time

ros2_control + HAL: Controller Manager in HAL RT thread

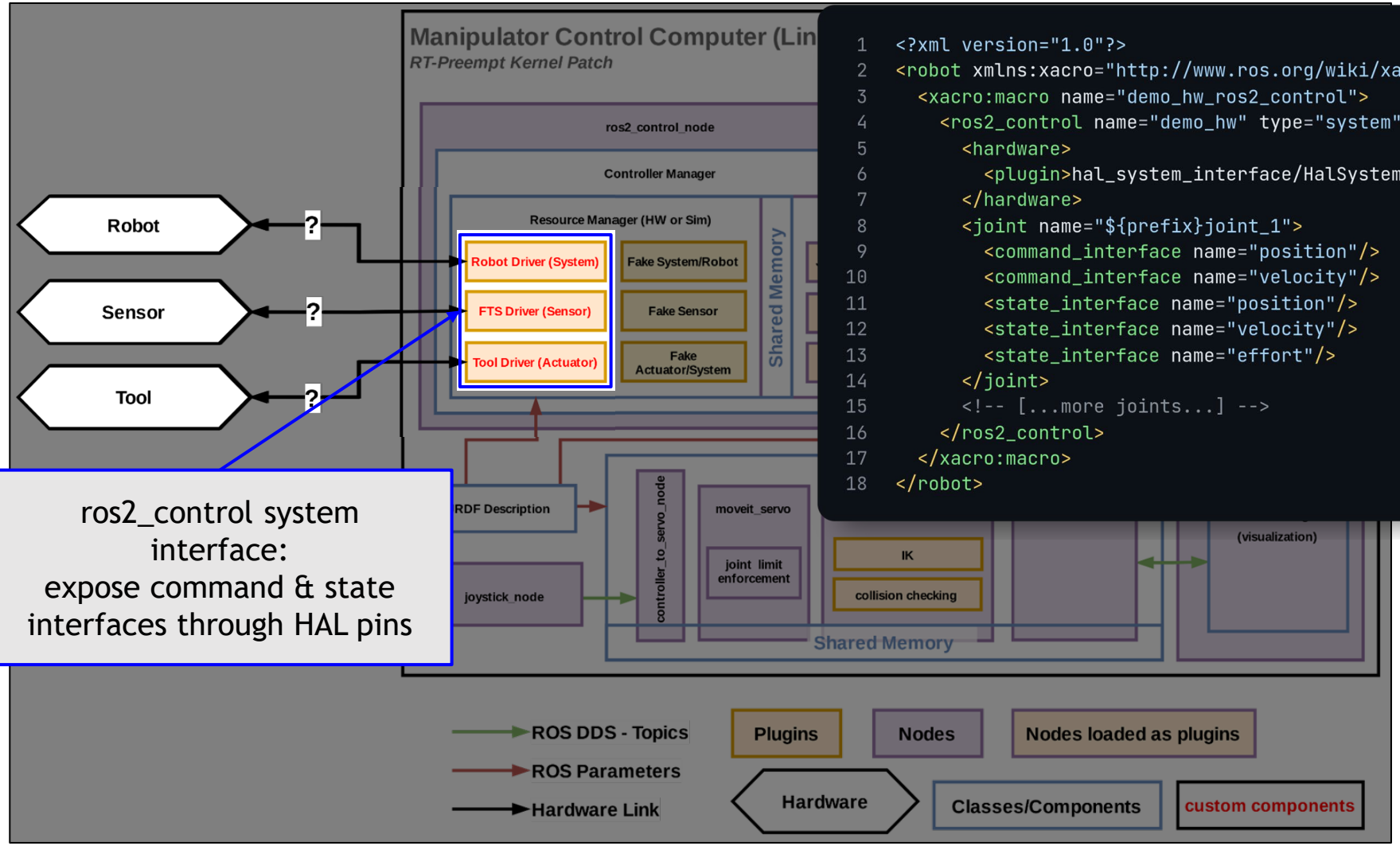
```
- hal_config:
    param:
      - name: hal_debug_output
        value:  $(var hal_debug_output)
      - name: hal_debug_level
        value:  $(var hal_debug_level)
    actions:
      - hal_rt_node:
          pkg: hal_hw_interface
          component: hal_control_node
          param:
            - name:  robot_description
              value:  $(var robot_description_content)
            - from:  $(var hal_hw_interface_yaml_path)
            - from:  $(var ros2_controllers_yaml_path)
          wait_timeout: $(var hal_wait_timeout)
      - hal_files:
          hal_files:
            - $(var hal_file)
          param:
            - name:  sim_mode
              value:  $(arg sim_mode)
            - from:  $(arg hal_config_yaml_path)
            - from:  $(arg joint_limits_yaml_path)
```



[Diagram from ros2_control docs]

TORMACH®

ros2_control + HAL: Hardware interface

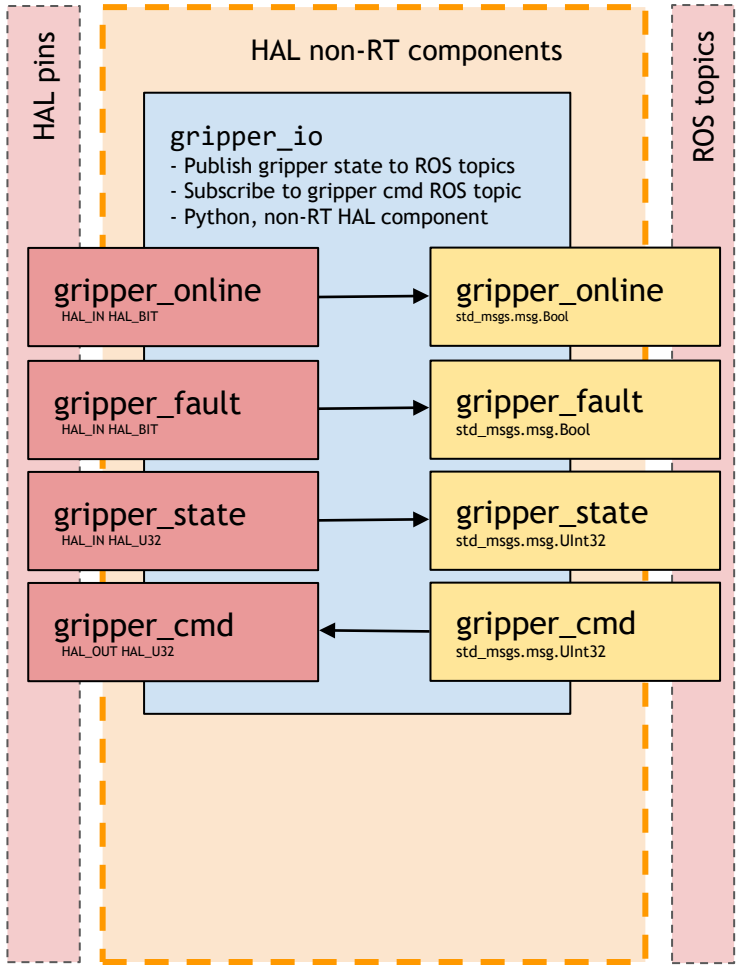



```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="demo_hw_ros2_control">
    <ros2_control name="demo_hw" type="system">
      <hardware>
        <plugin>hal_system_interface/HalSystemInterface</plugin>
      </hardware>
      <joint name="${prefix}joint_1">
        <command_interface name="position"/>
        <command_interface name="velocity"/>
        <state_interface name="position"/>
        <state_interface name="velocity"/>
        <state_interface name="effort"/>
      </joint>
      <!-- [...more joints...] -->
    </ros2_control>
  </xacro:macro>
</robot>
```

ros2_control system interface:
expose command & state interfaces through HAL pins

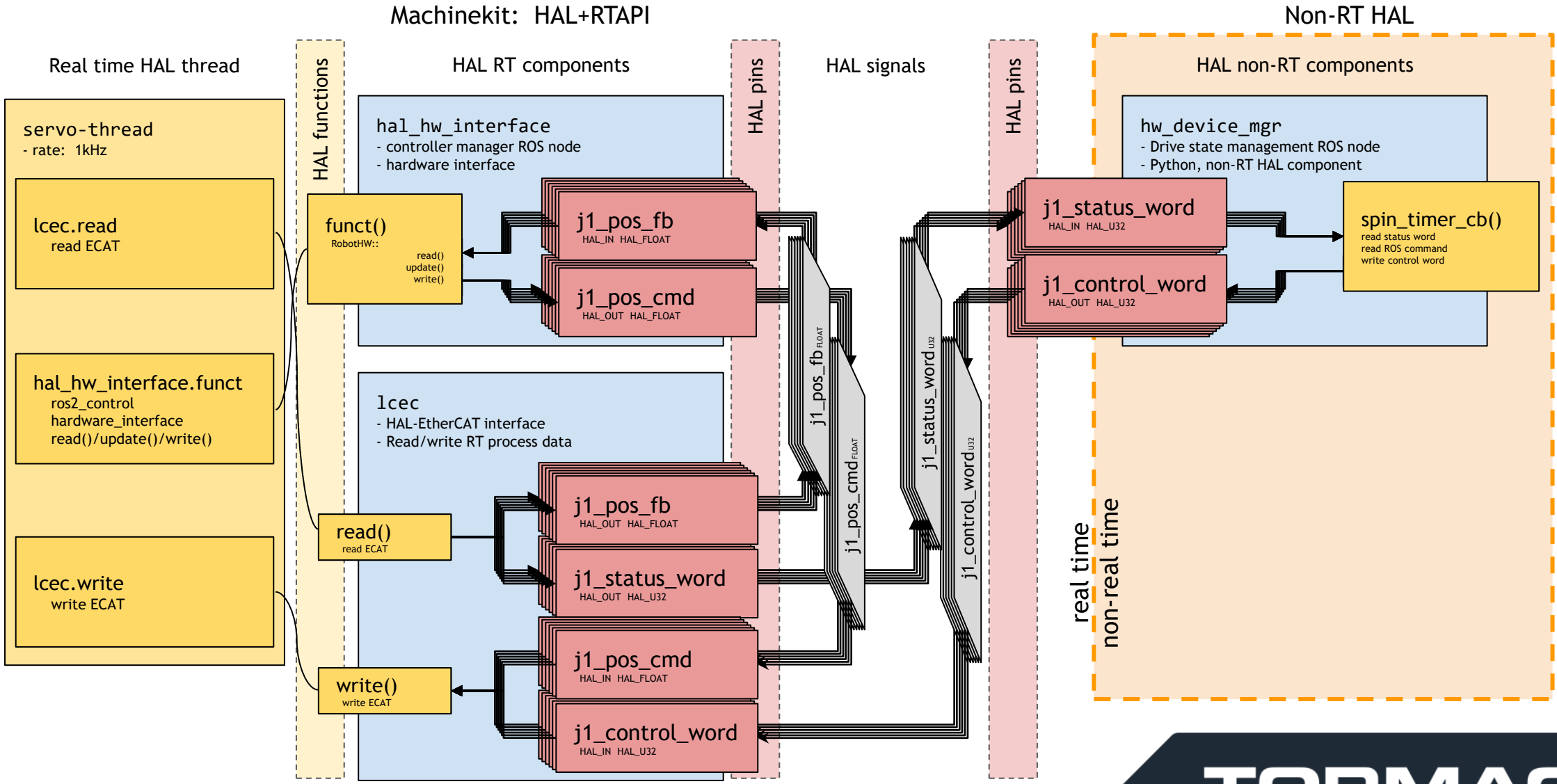
[Diagram from ros2_control docs]

`hal_ros_control` and ROS: Other interfaces



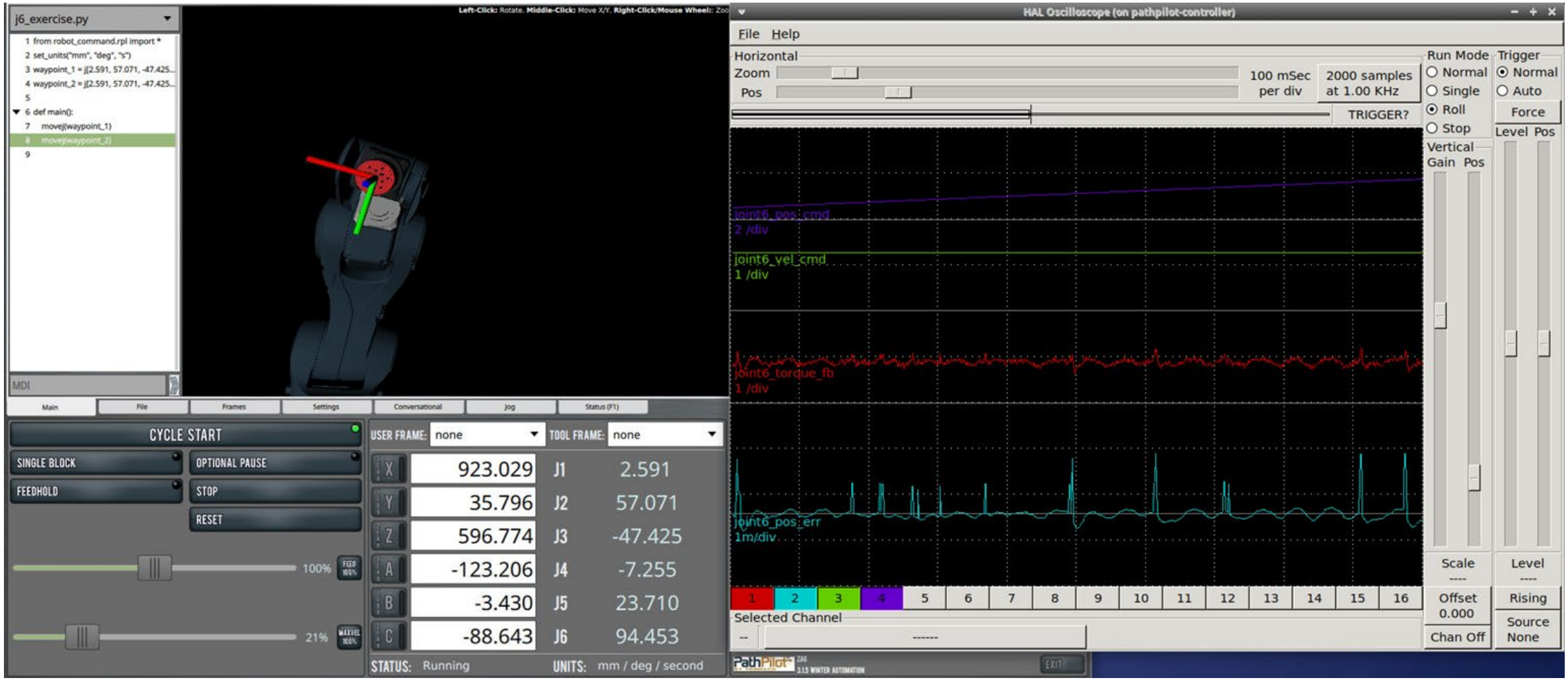

```python
from hal_hw_interface.ros_hal_component import RosHalComponent
from hal_hw_interface.ros_hal_pin import RosHalPinSubscriber, RosHalPinPublisher

class GripperIO(RosHalComponent):
    compname = "gripper_io"

    def setup_component(self):
        self.pins = [
            RosHalPinPublisher("gripper_online", hal_type="BIT"),
            RosHalPinPublisher("gripper_fault", hal_type="BIT"),
            RosHalPinPublisher("gripper_state", hal_type="U32"),
            RosHalPinPublisher("gripper_cmd", hal_type="U32"),
        ]

    def update(self):
        for p in self.pins:
            p.update()
```

Basic `hal_ros_control` robot hardware configuration



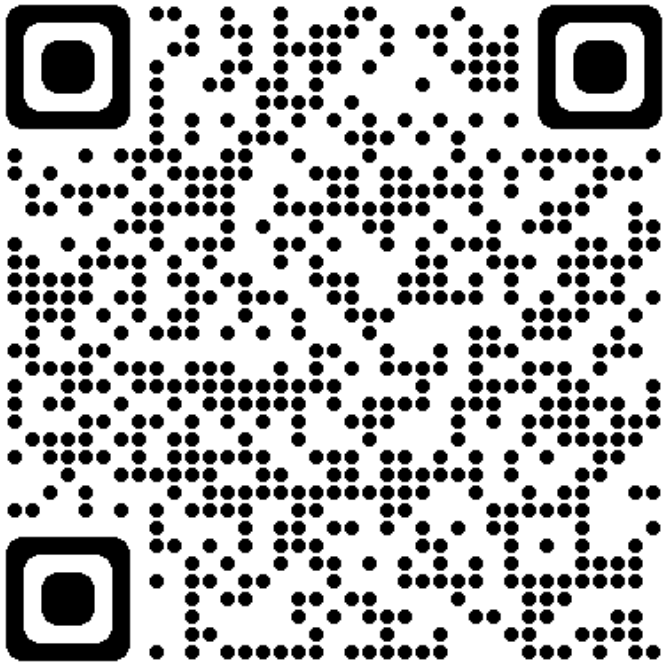
TORMACH®

ZA6 Robot: ROS-based, fully open source control stack



TORMACH®

HAL-ROS Control



https://github.com/tormach/hal_ros_control