Software Framework for Research in Semi-Autonomous Teleoperation

Co-PIs: Peter Kazanzides, Russell Taylor, Greg Fischer, Blake Hannaford

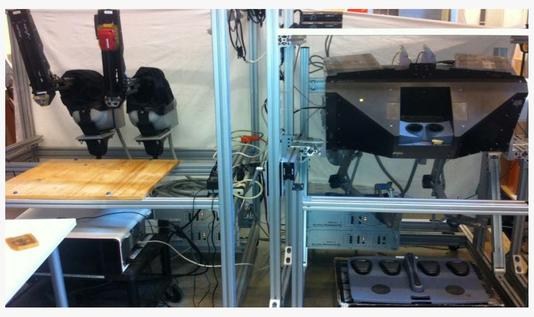
NRI 1637789
Johns Hopkins University

NRI 1637759
Worcester Polytechnic Institute

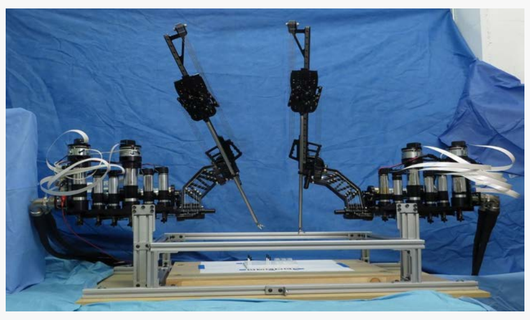
NRI 1637444
University of Washington

Project Overview

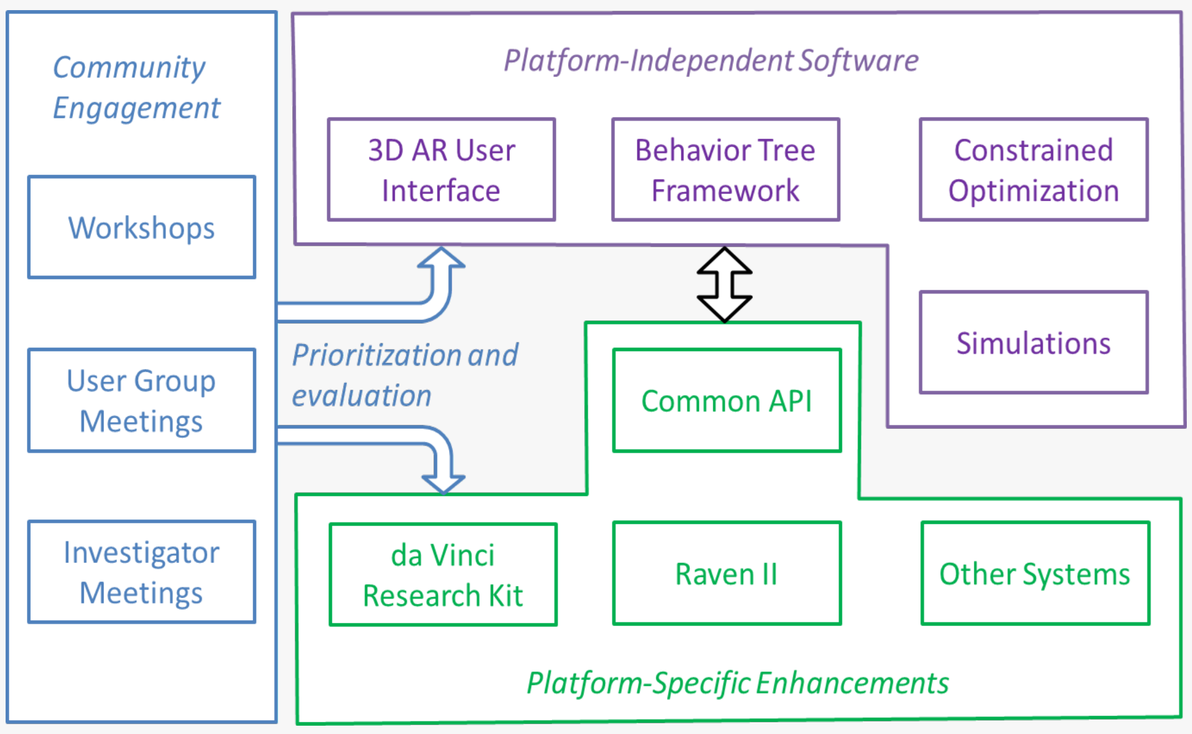
- Community Engagement:
  Organized a CRTK Tutorial at IROS 2018, Madrid.
  Organized a Workshop at ICRA 2018, Brisbane.
- Platform-specific Enhancements:
  Initial implementation of a ROS CRTK API to Raven II and dVRK.
- New Platform-independent Software:
  Prototyped vision pipelines and worked on robot simulations in Gazebo/rviz.
  Initial implementation of CRTK Python client API



da Vinci Research Kit (dVRK)



Raven II



JHU Highlights

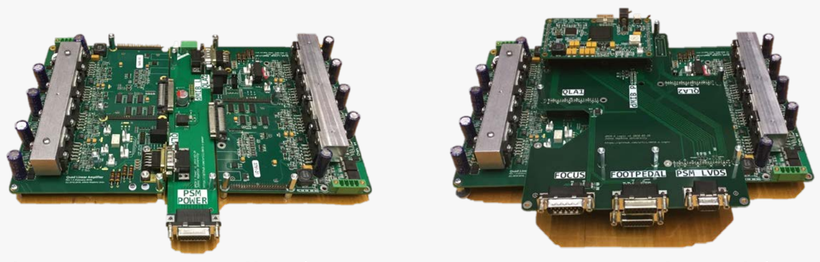
dVRK Software Release 1.6.0:

- New Features
  - Experimental ROS CRTK interface
  - Added ROS tf2 broadcaster
  - Camera manipulator (ECM) tele-operation
  - Software and hardware support for da Vinci:
    - Endoscope focus controller
    - Operator present head sensor
- Improvements and bug fixes
  - Improved velocity estimation on FPGA
  - Added audio feedback for console events
  - Better performance for ROS publishers/subscribers
  - Factorized and added new Qt Widgets

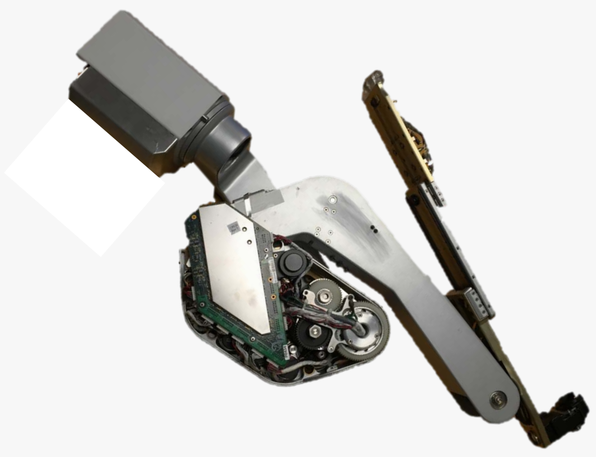
Support for full da Vinci (including Setup Joints):

- Fixed last issues with controller design (JHU)
- Finalizing orders for first batch, ~15 groups (WPI)

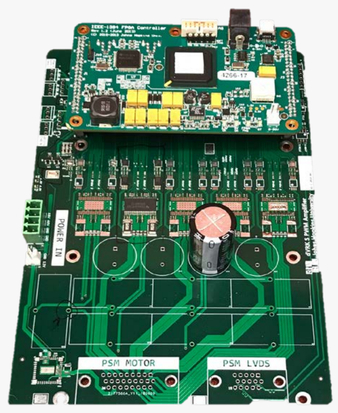
dVRK-S/Si



da Vinci S controller (2xQLA + FPGA + custom boards)



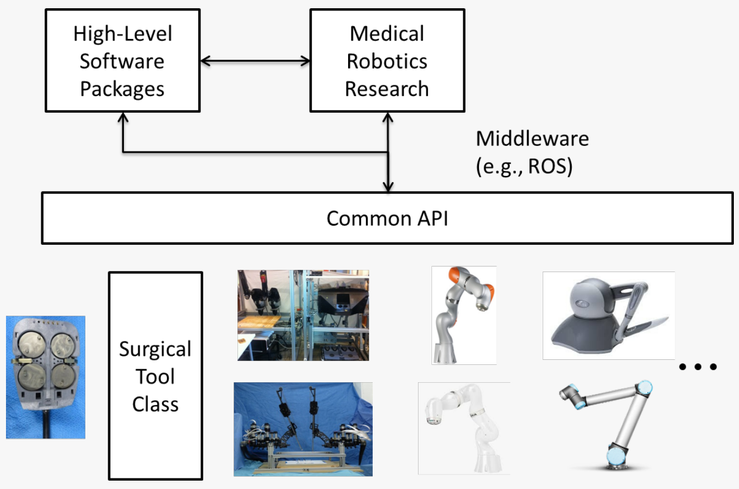
da Vinci S PSM



da Vinci S controller (PWM + FPGA)

- Initial prototype interface to da Vinci S/Si PSM
- Working prototype using 2 x QLA + FPGA, but gets hot
- Designed new PWM based controller, needs testing

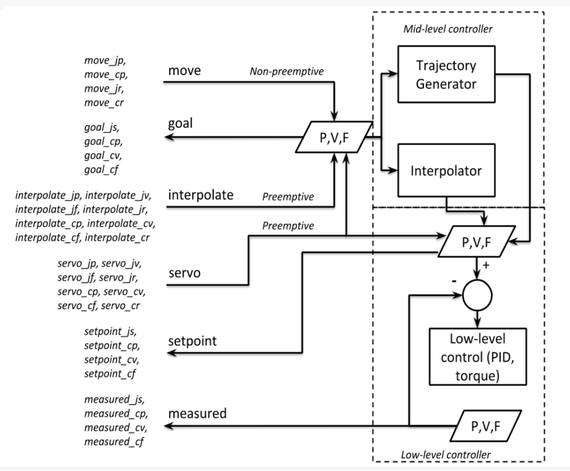
Common API (JHU, WPI, UW)



Guiding Principles:

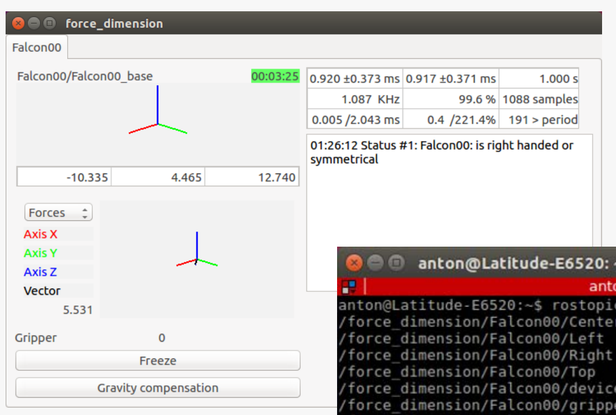
- As simple as possible
- Guided by uses cases
- Logical and consistent naming conventions
- Short enough to type into interpreter
- Publish/subscribe (ROS topics) and client/server (ROS services)

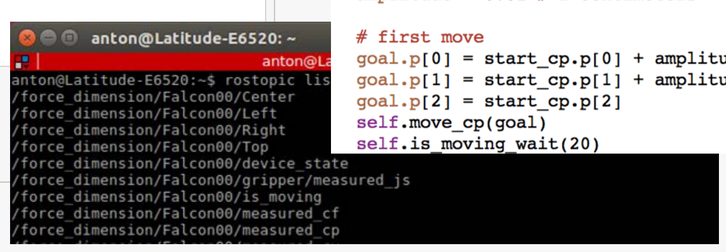
	Syntax
Control level	`servo` : direct real-time stream (pre-emptive) `interpolate` : interpolated stream (pre-emptive) `move` : plan trajectory to goal (pre-emptive), monitor with `is_moving`
Feedback	`measured` : sensor feedback `measuredN` : redundant sensor feedback (N=2, 3...) `setpoint` : current setpoint to low-level controller `goal` : most recent `interpolate` or `move` goal
Space	`j` : joint `c` : cartesian
Type	`p` : position `r` : relative `v` : velocity or twist `f` : generalized force (effort and wrench) `s` : state for joint feedback (includes position, velocity and effort)



Results:

- Initial CRTK implementation for *dVRK*, *Novint Falcon* and *Sensable Omni*
- Python CRTK Client API prototype






```
# create a new goal starting with current position
start_cp = PyKDL.Frame()
start_cp.p = self.measured_cp().p
start_cp.M = self.measured_cp().M
goal = PyKDL.Frame()
goal.p = self.measured_cp().p
goal.M = self.measured_cp().M
amplitude = 0.01 # 2 centimeters

# first move
goal.p[0] = start_cp.p[0] + amplitude
goal.p[1] = start_cp.p[1] + amplitude
goal.p[2] = start_cp.p[2]
self.move_cp(goal)
self.is_moving_wait(20)
```

https://github.com/collaborative-robotics