ROS in Space
Thoughts on Developing and Deploying ROS for Space Robotics

Terry Fong
Intelligent Robotics Group
NASA Ames Research Center
terry.fong@nasa.gov

irg.arc.nasa.gov
Space Robotics

PROTOTYPES

FLIGHT SYSTEMS

ROS in Space – Thoughts on Development & Deployment
Prototypes

Custom use cases
- **Planetary rovers**: natural terrain, instruments, multiple modes
- **Free-flyers**: full 6-DOF in micro-gravity
- **Dexterous manipulators**: in-space and planetary surface operations

Diverse deployments
- Laboratory – indoor & outdoor
- Field tests – planetary analog sites (craters, deserts, etc.)
- International Space Station

Robot software requirements
- Support applied research & development
- Enable complex proof-of-concept / demonstrations
- Facilitate deployment (including constraints: comm, ops, etc.)
Design to minimize risk

- Risk (technical, schedule, mission) is due to complexity
- Complexity = how hard something is to understand or verify
- Good software architecture is the most important defense against incidental complexity

Fanatical emphasis on code quality (# defects/KLOC)

1.0 Software industry average
0.5 Open-source projects (2011 Coverity survey)
0.6 Linux 2.6 (7 MLOC)
0.1 NASA flight software (mission critical code)

Use of fault protection

- Fault containment (limit cascade effects)
- Randomized testing (avoid bias to specific errors)
**Documentation**
- Design & implementation details
- End-to-end traceability from requirements to code

**Software V&V**
- **Verification**: demonstrate that software meets requirements
- **Validation**: demonstrate that software satisfies its intended use in its intended environment

**Lines of Code**
- Roomba: 1 KLOC
- Stanley: 100 KLOC
- Mars Exploration Rovers: 555 KLOC
- Curiosity rover: 2,000 KLOC
Control Modes for Space Robots

**Ground control**
- **Mission control** operates robot (on flight vehicle or other planet)
- Off-load routine & tedious work from astronauts: maintenance, repetitive tasks (e.g., inventory)
- Perform robotic exploration (e.g., field geology)

**Crew centric**
- **Astronauts** remotely operate robots from inside flight vehicle
- Extra-vehicular activities (outside vehicle, surface, etc.)
- Perform structural inspection, mobile sensor, surface field work
Robot Software Needs (1)

Framework
- Application structure
- Concurrency & synchronization
- Service management

Middleware
- Efficient data distribution (optimize transport & message size)
- Diverse data types, communication patterns, rates, QoS, latency
- Pluggable transport layer (if possible…)

Building blocks
- Robot skills and primitivies
- Services or behaviors or modules
- Structured data (e.g., maps)
Robot Software Needs (2)

**Architecture**
- Control loops
- Data flow patterns
- Layers

**Support tools**
- Simulation (for development & debugging)
- Data logging & replay
- Configuration management (versioning, dependencies, etc)
Use of ROS (by NASA)

**Suitability**
- ✔ Prototype work (including Space Station testing)
- ✘ Flight systems

**Sustainability**
- Will ROS exist *throughout* a project’s lifecycle?
- Code stability & standards (API vs. ABI)

**Code quality**
- Design & implementation
- Correctness, robustness, reliability, defects

**Licensing**
- How can we guarantee that code really is unencumbered?
- Re-release / bundling

**Tech support**
- Bug fixes, documentation, design reference, etc.