Astrobee: ROS-based Flight Software for a Free-flying Robot in Microgravity

Andrew Symington
Stinger Ghaffarian Technologies | NASA Ames

On behalf of the Astrobee Flight Software Team

Lorenzo Flückiger (lead)
Kathryn Browne
Brian Coltin
Jesse Fusco
Theodore Morse
Andrew Symington

nasa.gov/astrobee github.com/nasa/astrobee

What is Astrobee?

- Astrobee is a one square foot free-flying, holonomic robot
- Designed to operate inside the International Space Station
- The objective is to use the robot
  - to conduct surveying tasks,
  - as a remotely operated pan-tilt camera,
  - and to carry out scientific experiments.
- Designed to ultimately operate without crew assistance
- Software is open source and upgradeable on-orbit

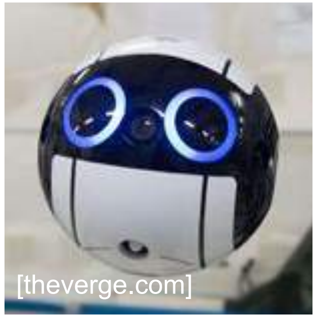
Astrobee: A new platform for free-flying robotics on the international space station. Trey Smith, Jonathan Barlow, Maria Bualat, Terrence Fong, Christopher Provencher, Hugo Sanchez, Ernest Smith et al. In *AIAA SPACE 2015 Conference and Exposition* (p. 4643).

Other Free-Flying Robots on the ISS

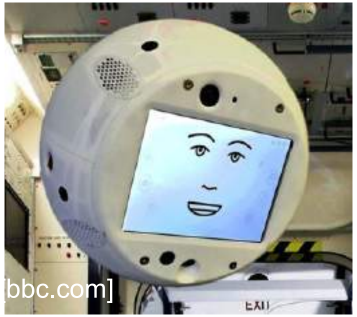
- Astrobee replaces the 4kg SPHERES robots
  - Most-used payload on the ISS
  - Limited processing resources, CO_2 and non-rechargeable batteries
- Other Free-Flyers on the space station
  - IntBall - 1kg free-flying camera
  - CIMON - 5kg AI-based assistant for astronauts

[nasa.gov]

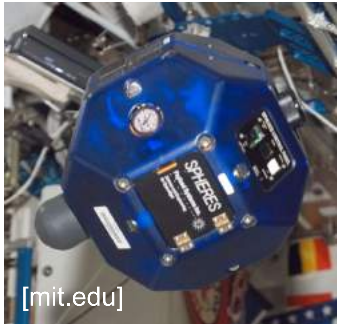
IntBall


[theverge.com]

CIMON

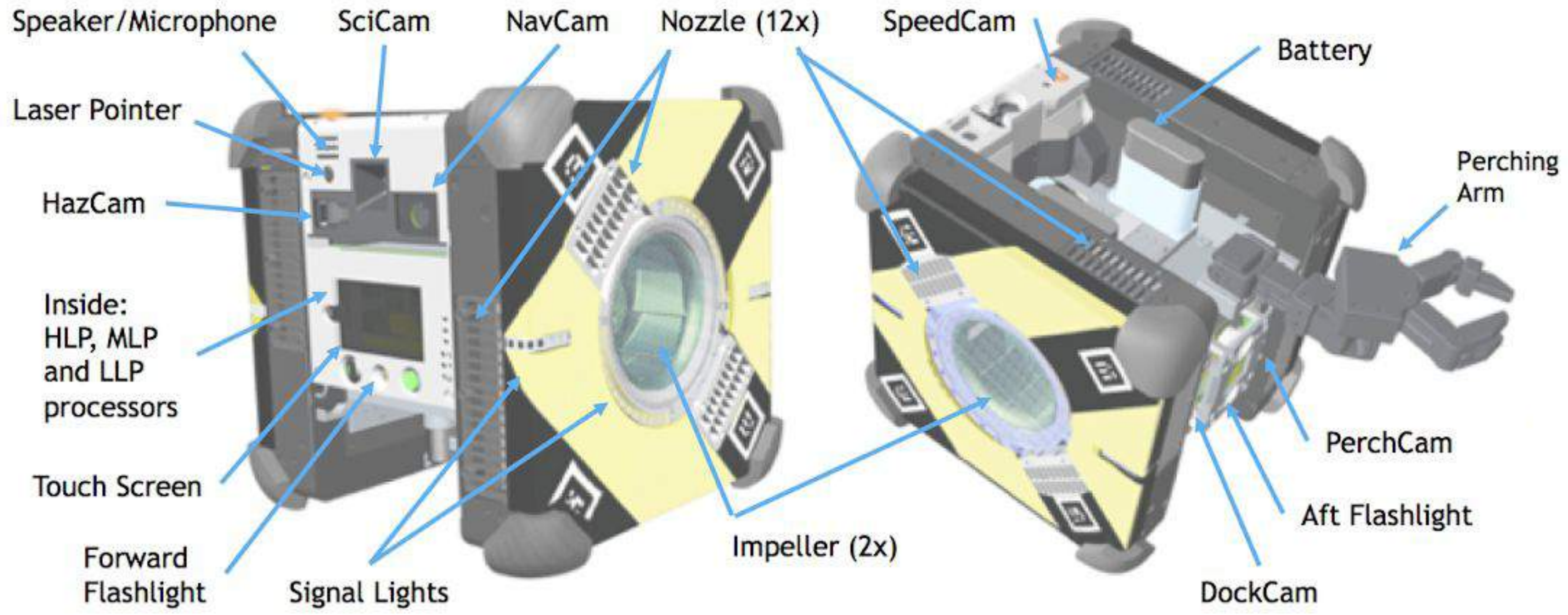

[bbc.com]

SPHERES

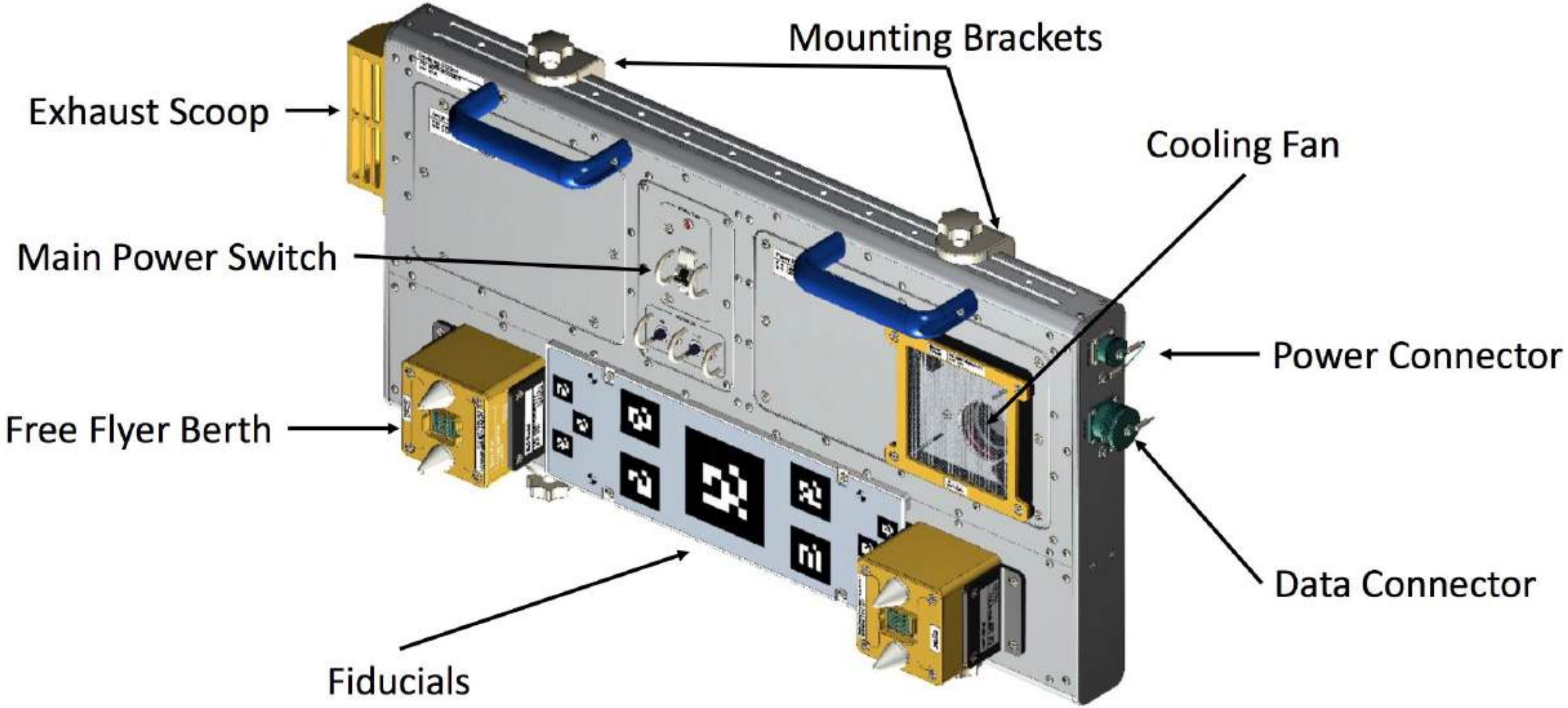


[mit.edu]

Hardware: Free-Flyer

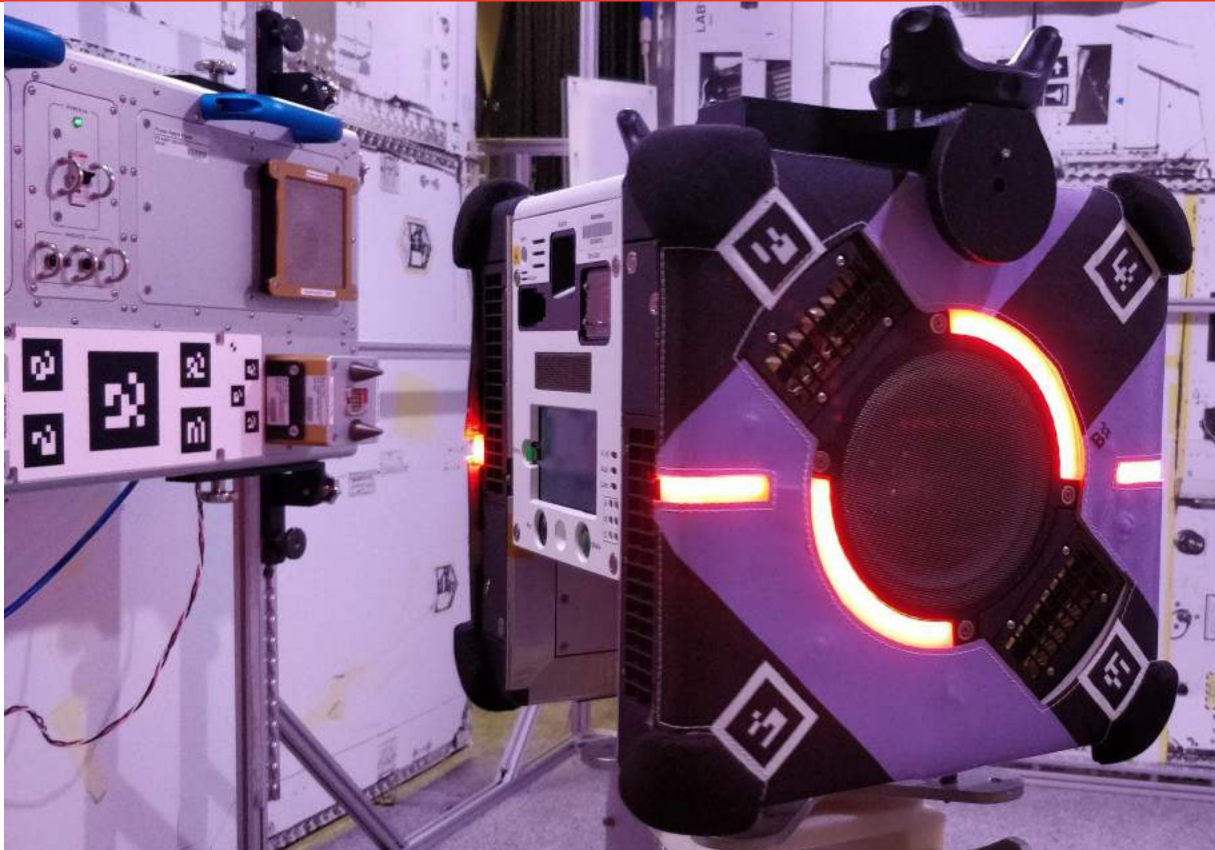


Hardware: Dock



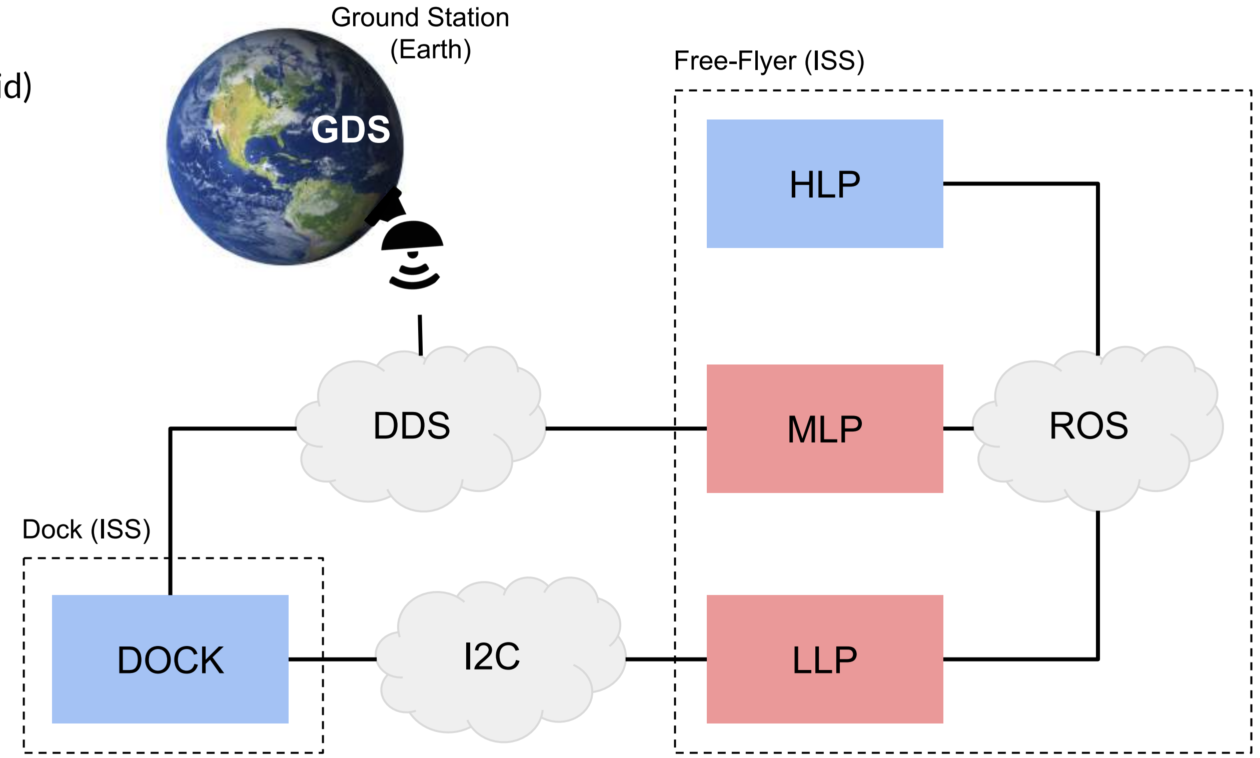
Project State

- Hardware design complete
- Hardware assembly in process
- Currently undergoing tests
  - Functionality
  - EMI
  - Acoustic
  - Vibration
  - Thermal
  - ...
- Software upgradeable on-orbit
- Dock launch likely NG-10
- Three robots to follow in 2019...



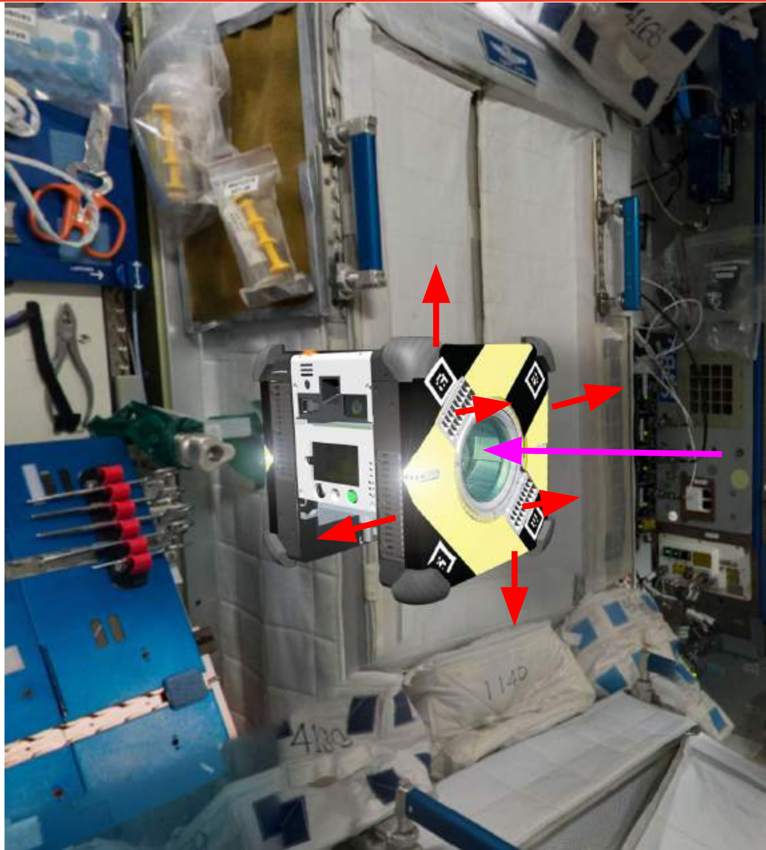
Processors and Connectivity

- **High-level Processor** (Android)
    - Guest Science
- **Mid-level Processor** (Linux)
    - Executive
    - Mobility
    - Localization
    - Communications
- **Low-level Processor** (Linux)
    - State estimation
    - Control
    - Actuation
    - Signaling

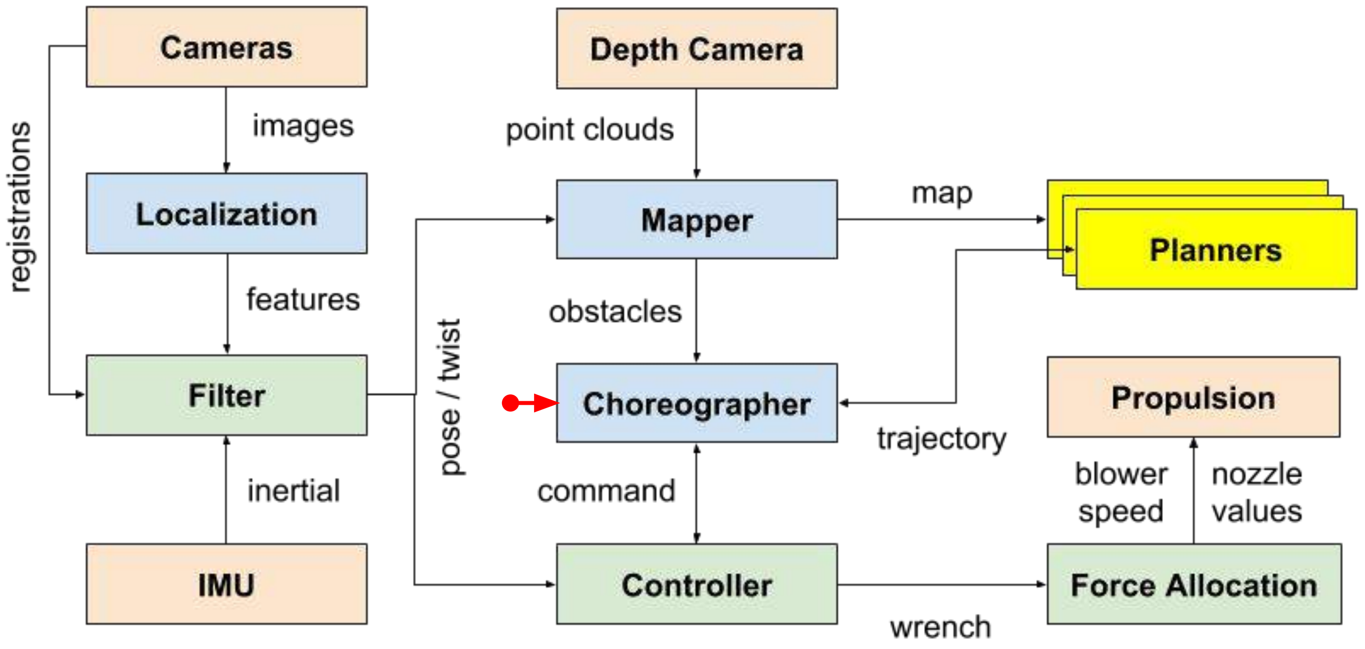


Propulsion System

- Two propulsion modules per robot -- port and starboard sides
- Each propulsion module has
  - One centrifugal impeller
  - One plenum
  - Six nozzles
- Impellers counter-rotate to balance torque
- Force and torque coupled in plenum pressure
- We hold impeller speed constant
- **Control vector** - Six nozzle apertures
- **Advantages** - No propellant, rechargeable, no exposed rotors
- **Disadvantages** - Limited power, microgravity-specific

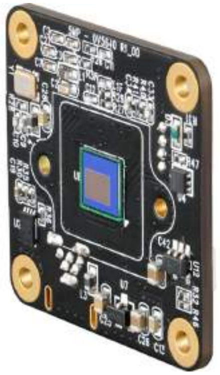


Motion Control Pipeline



Localization

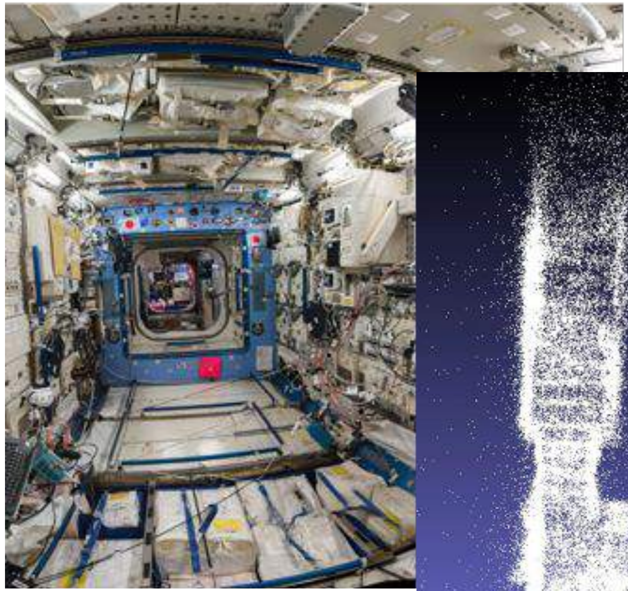
- Three sensing modalities
  - **Inertial Measurement Unit -** Epson M-G362PDC1
  - **RGB camera -** Imaging Source 42BUC03-ML
    - HFOV: Front 110.8°, Aft: 85°
    - 1280 x 960 pixels, Global shutter
  - **Depth camera -** PMD PicoFlexx
    - HFOV: 62°, 0.1 - 4.0 m range
    - 224 x 172 pixels
- Three localization pipelines
  - **Sparse Mapping** - General navigation
  - **Marker tracking -** Used for docking
  - **Handrail tracking -** Used for perching
- Augmented-state EKF fuses inertial with localization pipeline features

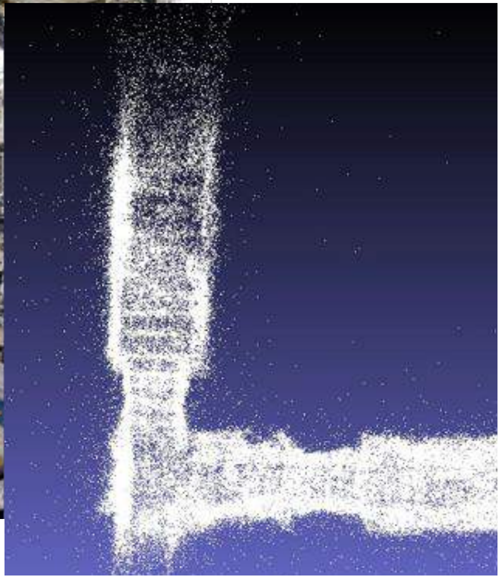


Localization: General Navigation

- Offline
  - Capture a set of video or images
  - Bundle adjustment
  - Output is sparse feature map
- Online
  - Detect BRISK features <uv>
  - Find corresponding feature <xyz> in sparse map
  - We also use features from visual odometry
  - Features used as EKF corrections

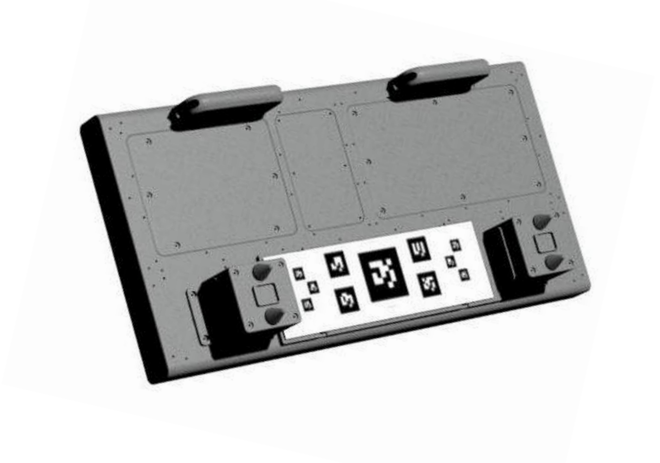
Localization from Visual Landmarks on a Free-Flying Robot. Brian Coltin, Jesse Fusco, Zack Moratto, Oleg Alexandrov, and Robert Nakamura. *International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, 2016, pp. 4377-4382.

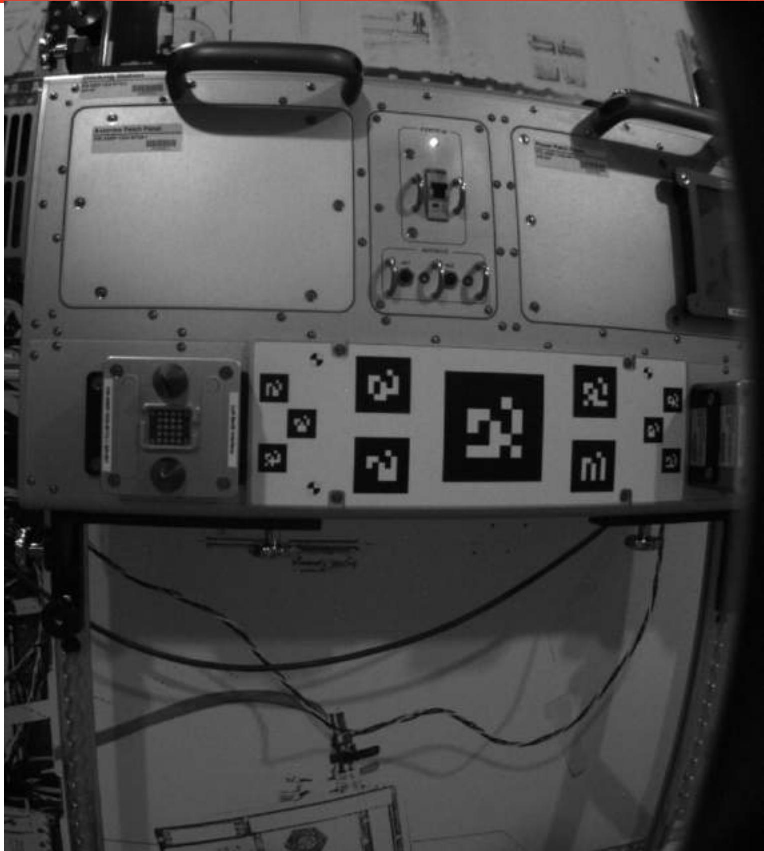




Localization: Docking

- Docking demands greater localization accuracy
- Augmented Reality (AR) targets provide richer features
- Arrangement of 11 targets supports a wide range of views
- Dock pose is bootstrapped from general navigation
- During docking localization is performed relative to dock

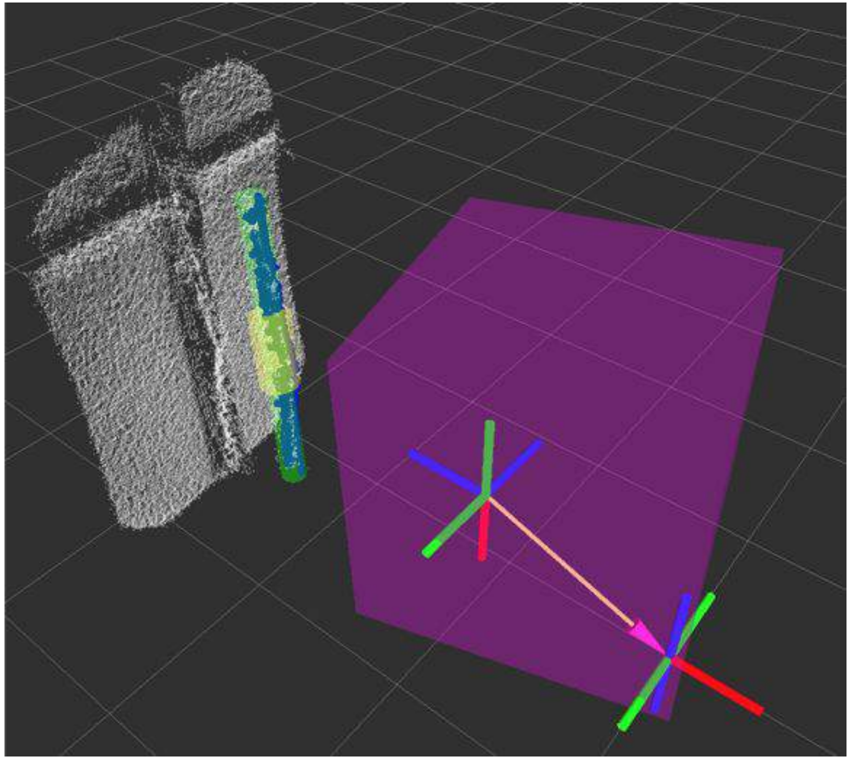




Localization: Perching

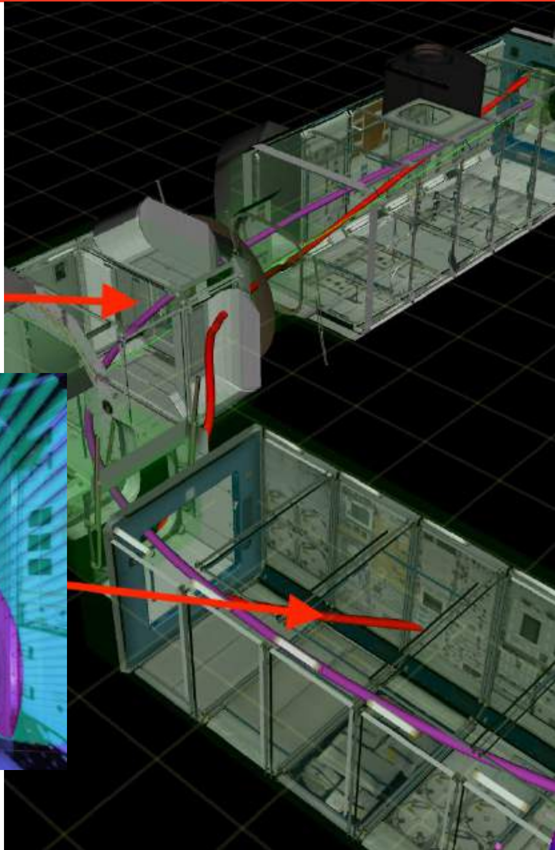
- Detect and localize with respect to handrails
- Processing depth camera images
  - Plane (ISS wall) estimation with RANSAC
  - Line (Handrail) estimation with RANSAC
  - Handrail end point detection
  - Pose estimation
  - Landmark sampling
- Landmarks used as filter corrections

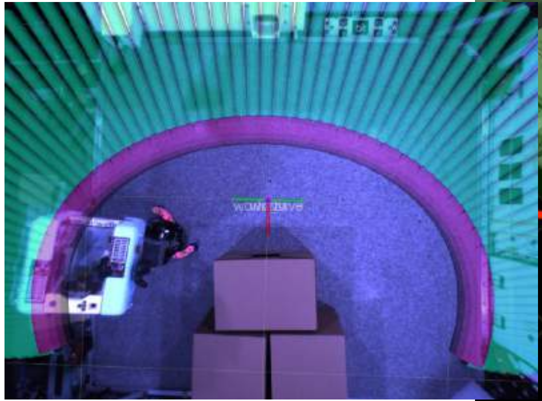
Handrail detection and pose estimation for a free-flying robot. Dong-Hyun Lee, Brian Coltin, Theodore Morse, In-Won Park, Lorenzo Flückiger, and Trey Smith. *International Journal of Advanced Robotic Systems*. Published January 17, 2018



Motion Planning

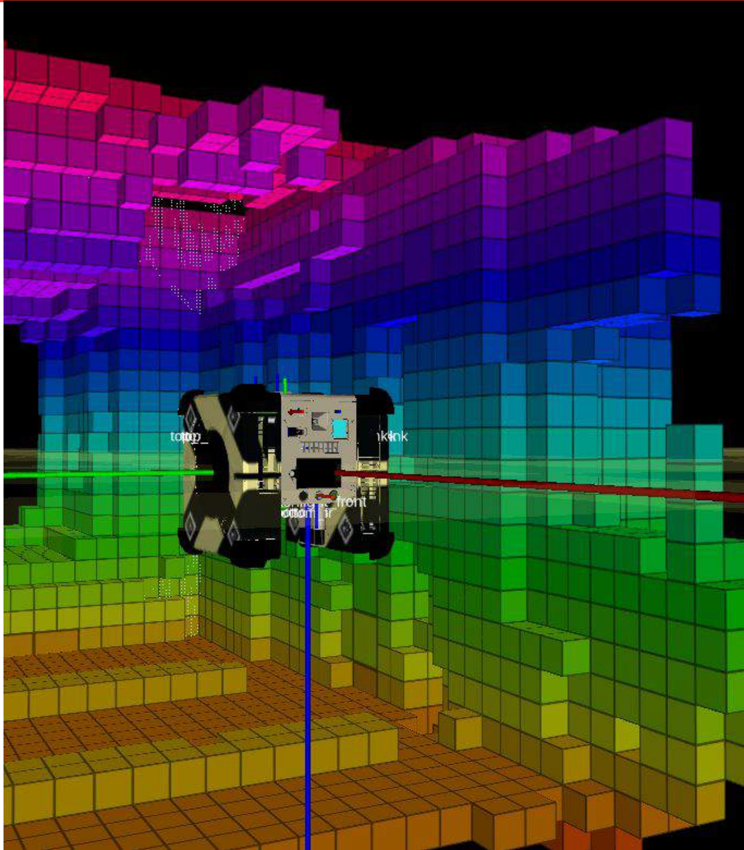
- Choreographer provides an extensible planning interface
- Baseline ‘trapezoidal’ planner
  - Plans straight lines between poses
  - Trapezoids on velocity and angular velocity
  - Agnostic to zones or obstacles
  - Useful for docking and perching
- Quadratic Program ‘qp’’ planner
  - Written by Mike Watterson
  - Generates curved trajectories
  - Minimizes jerk
  - Respects zones and obstacles
- Other planners currently being written...





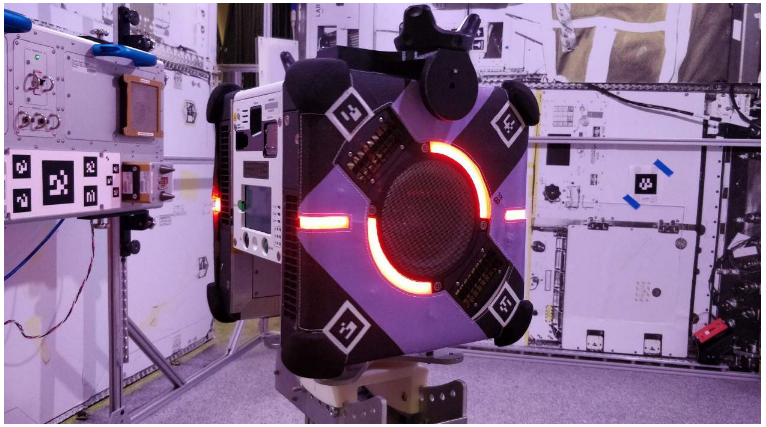
Safe Operation - Limits, Zones & Obstacles

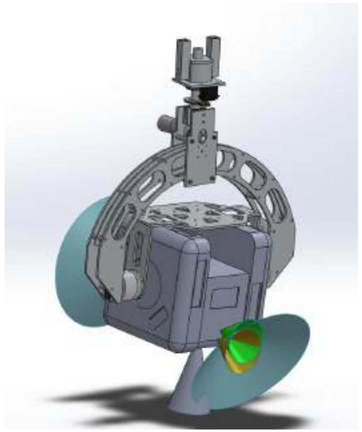
- Planners enforce limits on
  - **Acceleration** - hardware limited by plenum pressure
  - **Twist** - hardware limited by speed camera
- Planners also abide by keep-in and keep-out zones
- Must also stop on detecting an obstacle in path
- Needs to be robust to outliers
- We used *OctoMap* on depth camera measurements
  - We don't do free-space updates
  - Bresenham (line drawing) on current trajectory
  - Collision checking between line and map
- Runs in real-time on ARM platform
- Developed by an visiting student, Marcelino Almeida

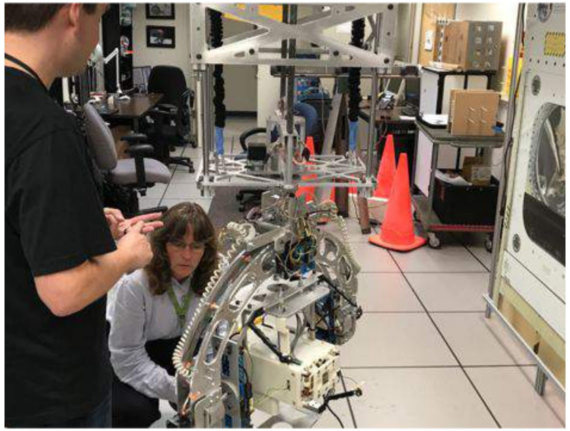


Hardware Test Environments

- How do we hardware test our hardware?
  - **FlatSat** - avionics core, no body or propulsion
  - **Granite Laboratory** - 3DoF only, propulsion used
  - **Micro-Gravity Test Facility** - 6DoF, propulsion emulated by gantry/gimbal

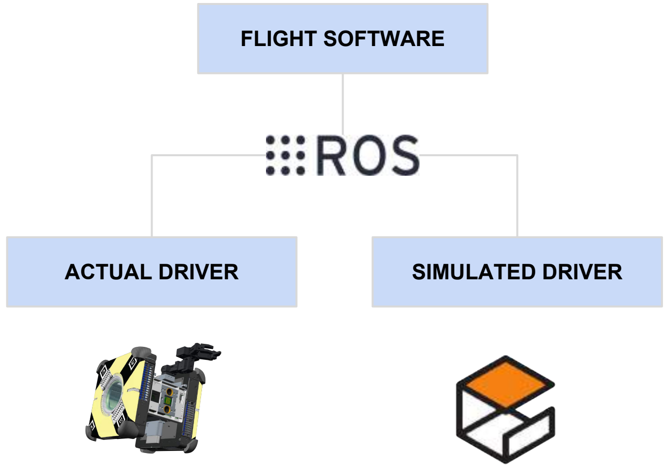






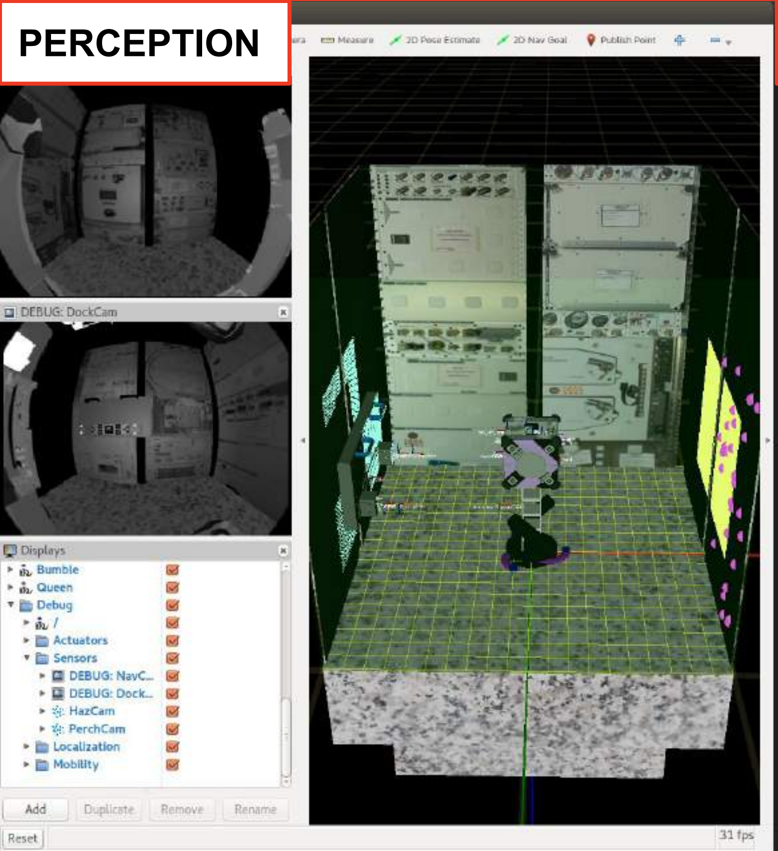
Simulation

- Software includes a **realistic** and **representative** simulation of two worlds
    - International Space Station (iss)
    - Granite lab (granite)
- Simulation is written as sensor / model plugins for
- We use ROS messages / services as a hardware abstraction layer
- Most code that runs in simulation will run on a real robot
- Important: We don't simulate computer vision
    - Sparse mapping
    - Marker tracking
    - Visual odometry
- We can speed up the simulation to run faster than realtime
    - Limit: generating point clouds / images, calling sequential localization / GNC

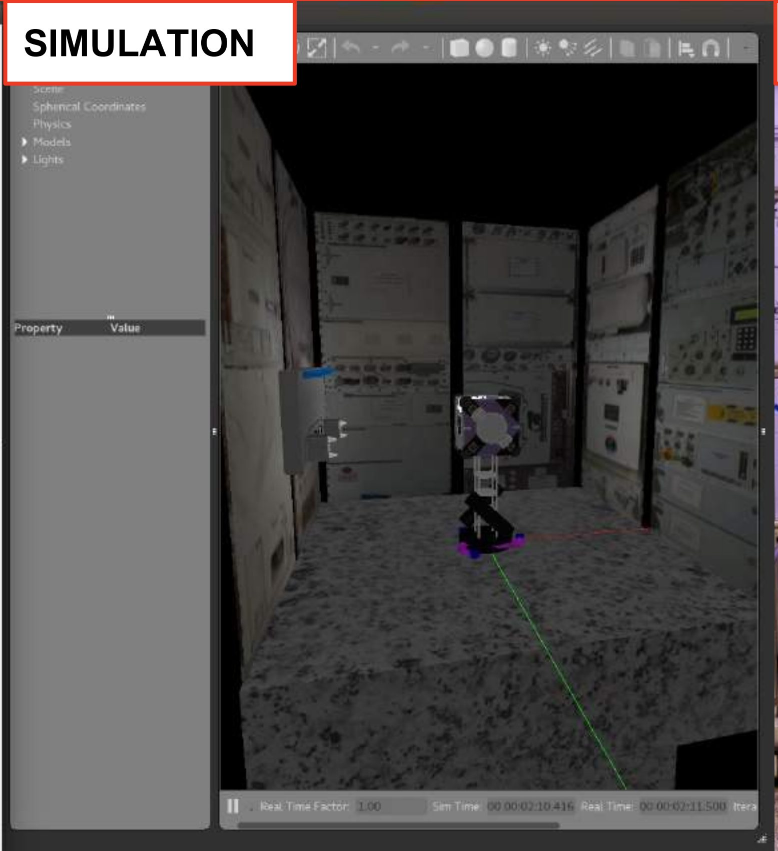


Simulation: Granite Lab

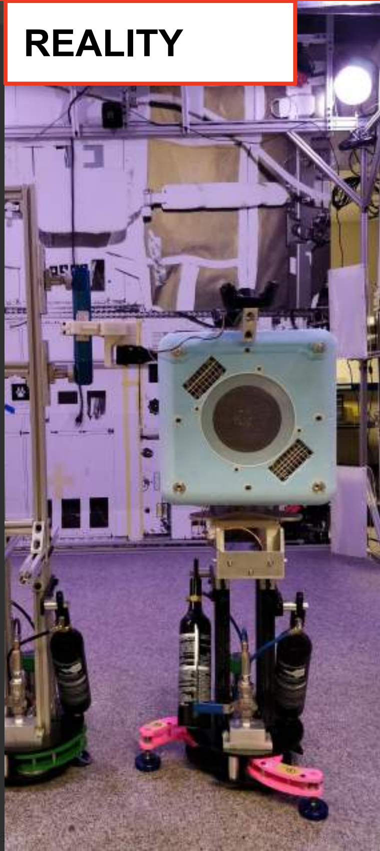
PERCEPTION



SIMULATION

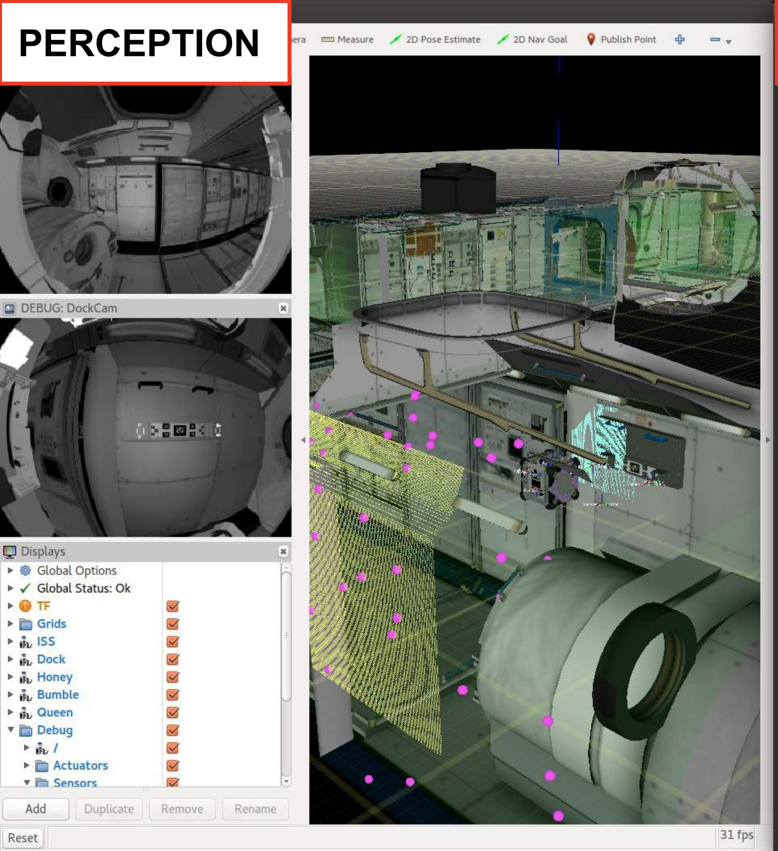


REALITY



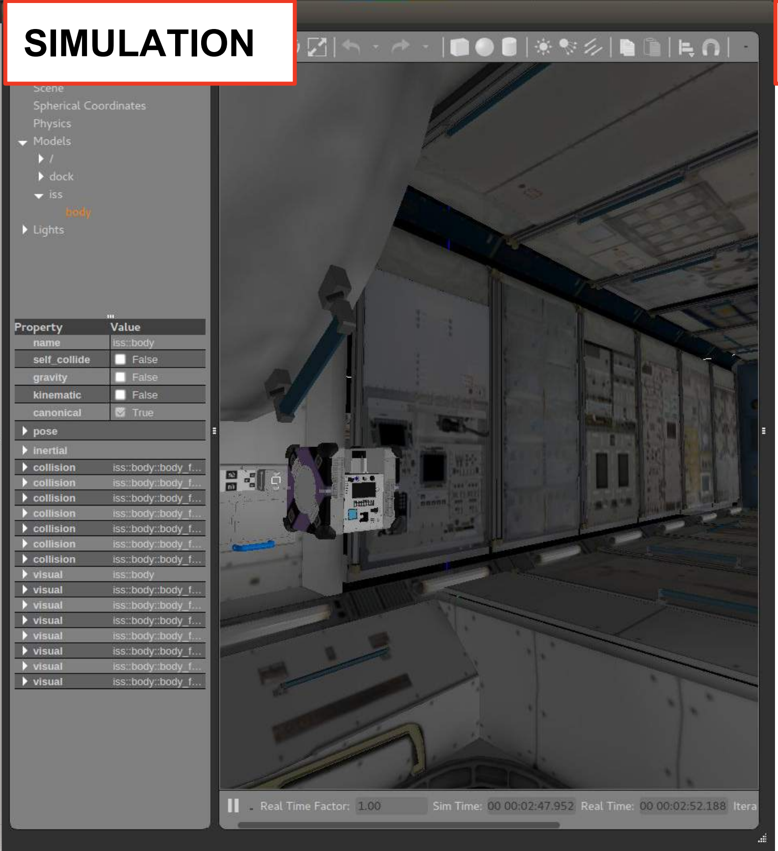
Simulation: International Space Station

PERCEPTION



- Global Options
- ✓ Global Status: Ok
- TF ✓
- Grids ✓
- ISS ✓
- Dock ✓
- Honey ✓
- Bumble ✓
- Queen ✓
- Debug ✓
  - / ✓
  - Actuators ✓
  - Sensors ✓

SIMULATION



Property	Value
name	iss::body
self_collide	False
gravity	False
kinematic	False
canonical	True
pose	
inertial	
collision	iss::body::body_f...
collision	iss::body::body_f...
collision	iss::body::body_f...
collision	iss::body::body_f...
collision	iss::body::body_f...
collision	iss::body::body_f...
collision	iss::body::body_f...
visual	iss::body
visual	iss::body::body_f...
visual	iss::body::body_f...
visual	iss::body::body_f...
visual	iss::body::body_f...
visual	iss::body::body_f...
visual	iss::body::body_f...
visual	iss::body::body_f...

Real Time Factor: 1.00 Sim Time: 00 00:02:47.952 Real Time: 00 00:02:52.188 Itera

REALITY

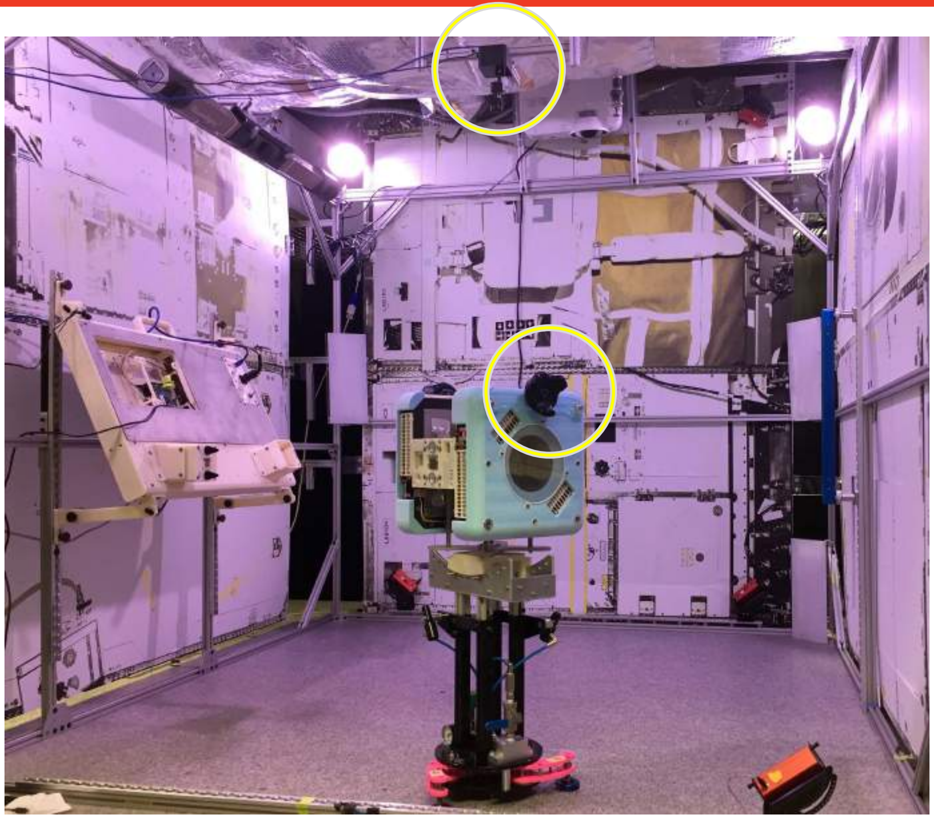
?

Check back in 2019

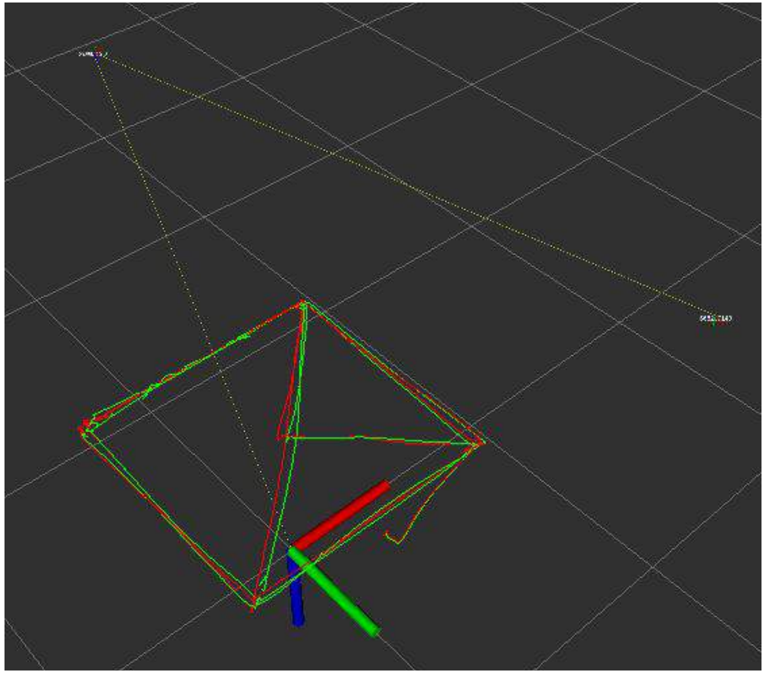
Ground Truthing: HTC Vive

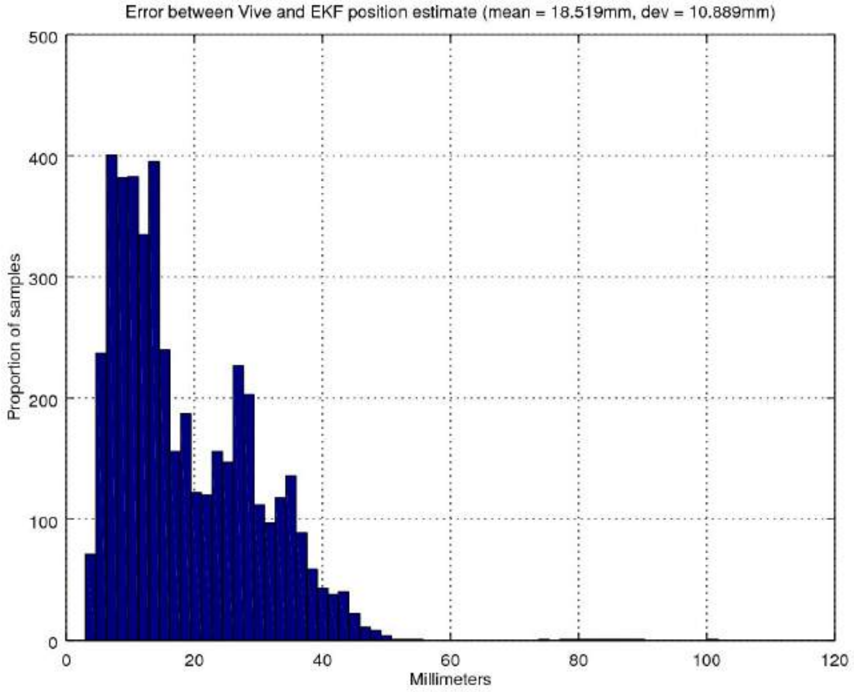
- How to measure localization performance?
- HTC Vive Tracking 1.0 system
- Basic operating principle
  - Lighthouses and trackers
  - IR sync pulses and laser sweeps
  - Inertial measurement unit
  - Sensor fusion
- Miguel Borges' IROS 2018 presentation

HTC Vive: Analysis and Accuracy Improvement. Miguel Borges, Andrew Symington, Brian Coltin, Trey Smith, Rodrigo Ventura. International Conference on *Intelligent Robots and Systems (IROS), Madrid, 2018.*



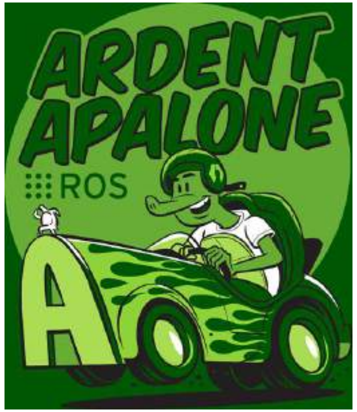
Ground Truthing: Initial Results




Error between Vive and EKF position estimate (mean = 18.519mm, dev = 10.889mm)
Proportion of samples
Millimeters
0
100
200
300
400
500
0
20
40
60
80
100
120

ROS and Astrobee

- ROS Indigo and Kinetic have been invaluable tools for Astrobee
- We've been using DDS for a while at NASA
- So, we're excited about ROS 2.0
    - In the short term, upgrade unlikely due to limited resources
    - In the long term, perhaps a community-driven upgrade?
- Other projects within the IRG
    - Tensegrity, K-REX



ROS: Timeouts on Actions

- **Problem -** in a single threaded NodeHandle context, to safeguard against service locking we needed to use actions to manage long-running tasks. But what if the action server dies midway through a goal?
  - Each nodelet could in theory manage all timeouts, but the code gets verbose
- **Solution -** a wrapper for the SimpleActionClient that manages timeouts in a clean way
- We augmented the class with the following functions
  - SetConnectedTimeout(...)
  - SetActiveTimeout(...)
  - SetResponseTimeout(...)
  - SetDeadlineTimeout(...)
- Added extra result codes to the result callback
  - void ResultCallback(FreeFlyerActionState::Enum, ResultConstPtr const&)

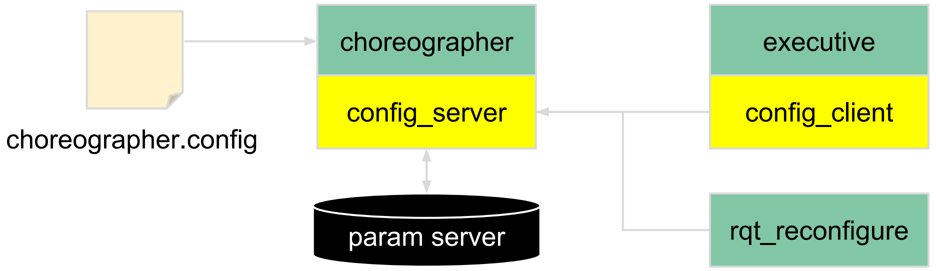
```
class FreeFlyerActionState {
 public:
  enum Enum {
    SUCCESS             =  1,
    PREEMPTED           =  0,
    ABORTED             = -1,
    TIMEOUT_ON_CONNECT  = -2,
    TIMEOUT_ON_ACTIVE   = -3,
    TIMEOUT_ON_RESPONSE = -4,
    TIMEOUT_ON_DEADLINE = -5,
  };
};
```

ROS: Configuration Files

- **Problem**: YAML works well, but it does have a few drawbacks
  - No support for complex data types - matrices, structures, etc.
  - No scripting support (cannot link variables)
  - No context support beyond selecting which YAML file is loaded
- **Solution**: We use LUA files, and access them through a C++ API.
- Optional ConfigServer and ConfigClient
  - Uses dynamic_reconfigure messages, so works with rqt_reconfigure
  - Enables dynamic reconfiguration of a node at runtime
    - ReconfigureCallback(...)
  - Requires well-specified LUA config file
  - Configuration persists on rosparam server
  - Resilient against a node restarts

```
require "context"

parameters = {
  {
    id = "enable_obstacles",
    reconfigurable = true,
    type = "boolean",
    default = false,
    unit = "boolean",
    description = "Enable obstacles?"
  }
}
```



ROS: Namespaces, Faults and Lifecycles

- **Problem -** We required a clean pipeline for communicating the robot namespace, asserting system faults, checking if nodes have died and a mechanism for starting, stopping or restarting nodes as needed.
- **Solution -** We developed a wrapper class around Nodelet, called **FreeFlyerNodelet**. All nodes in our software inherit from FreeFlyerNodelet, and are therefore also Nodelets. Our wrapper class:
    - Transparently sends a heartbeat on topic /heartbeat while the node is alive.
    - Provides an **AssertFault**(...) function that the child class can call to notify the system of a fault.
    - Provides the ability for a node to query the namespace of the robot on which it is running.
    - Provides the ability for a node to send diagnostic_msgs
- Fault information is gathered by executive, which can load / unload the nodelet from the manager as needed
- Our hardware drivers are also FreeFlyerNodelets
    - How does this work with Gazebo dynamic library loading?
    - Workaround: Gazebo plugin calls **FreeFlyerNodelet::Setup**(...), which bypasses **Nodelet::onInit**(...)

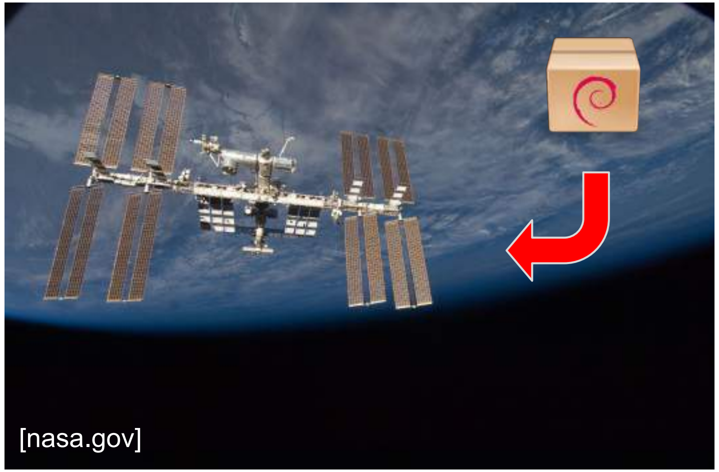
ROS: Cross-compilation for ARM

- **Problem** - Back in 2015 we had great trouble linking ROS packages against an ARM rootfs. This was mainly because the cmake scripts auto-generated by catkin hard-coded the full library paths
- **Solution** - Forking catkin and adding code to rewrite the library path in a cross-compile context
- Although it works well, there is definitely room to improve our build system
  - Migration to catkin_tools, catkin_simple, etc.

```
if (${component}_LIBRARIES)
  set(temp_LIBRARIES)
    foreach(library ${${component}_LIBRARIES})
      string(REGEX REPLACE "^/usr/lib" "${ARM_CHROOT_DIR}/usr/lib" library ${library})
      string(REPLACE "i386-linux-gnu" "arm-linux-gnueabihf" library ${library})
      string(REGEX REPLACE "^/opt/ros/kinetic" "${ARM_CHROOT_DIR}/opt/ros/kinetic" library ${library})
      list(APPEND temp_LIBRARIES ${library})
    endforeach()
  set(${component}_LIBRARIES ${temp_LIBRARIES})
endif()
```

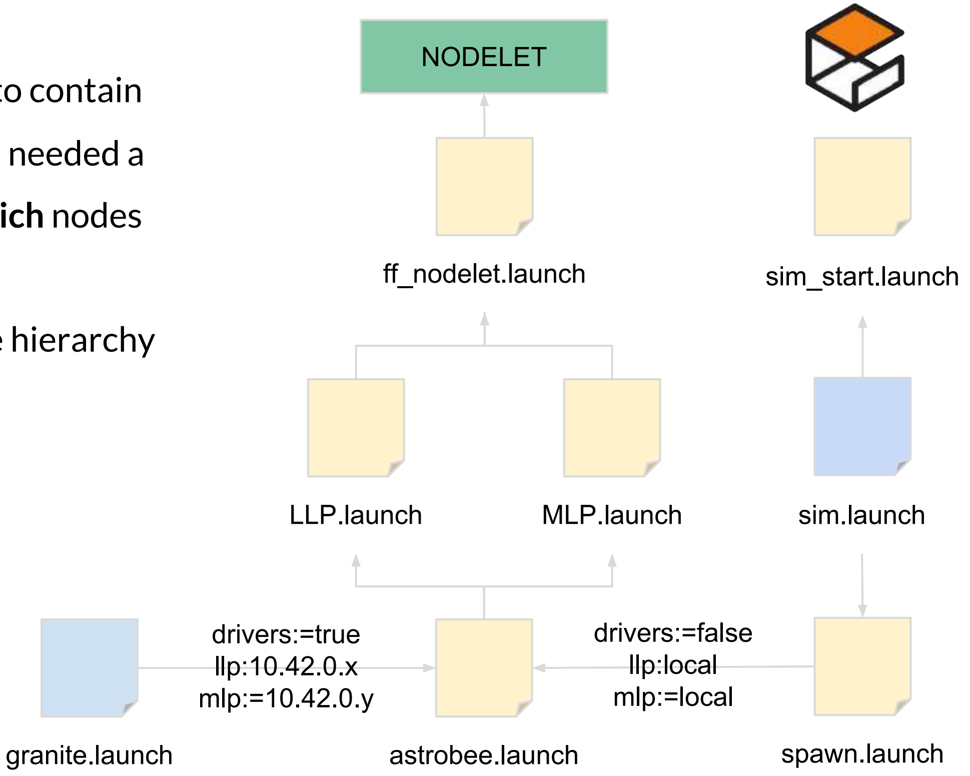
ROS: Dependencies and Debian Packages

- **Problem** - Parts of our code are released as Debian packages, either for x64 or ARM. In some cases these packages link against libraries provided by ROS. If the library locations change, it breaks our build.
    - The great OpenCV 3 linker errors of 2017: https://github.com/ros-perception/vision_opencv/issues/193
- **Solution** - Continually test and release fixes as needed.
    - This won't be a problem on-orbit until we upgrade.
    - We have release scripts that build and release Debians rapidly.
- Aptly is perhaps a good choice for us?



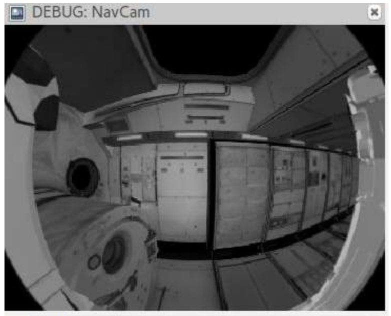
ROS: Launch System

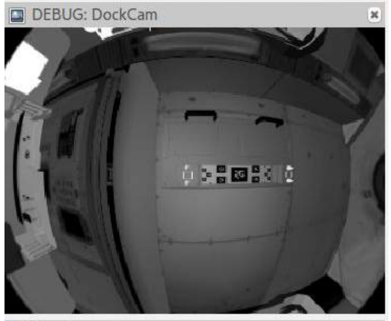
- **Problem -** over time, our launch folder started to contain a great number of task-specific launch files. We needed a launch system that enabled us to customize **which** nodes were launched and **where**
- **Solution -** machine tags and modular launch file hierarchy
- Following arguments supported:
  - **llp/mlp/sim**: hostname for <machine> tag
  - **nodes**: limit to launching a specific set of nodes
  - **ns**:namespace of robot
- Example: Simulation with MLP-in-the-loop
  - roslaunch sim.launch mlp:=10.42.0.32



ROS: TF2 Transform Consistency

- **Problem** - How to ensure consistent extrinsics between perception and simulation?
    - Our software defines extrinsics in LUA and broadcasts as world -> sensor TF2 transforms
    - Gazebo simulator requires transform declared in the model SDF / URDF
    - Probably very hard work to have xacro inject values from LUA into SDF
- **Solution** - Each simulated sensor plugin listens for a TF2 transform describing its extrinsics, and adjusts its pose in response to this information
- **Future work**: add noise perturbation to extrinsics
- Biggest implementation challenges
    - Sensor::SetPose(...) in Gazebo 7.x doesn't seem to also adjust the Ogre camera pose
    - Different conventions in camera frames between our software and Gazebo
- **Remaining bug** - Gazebo GUI sometimes doesn't update when SetPose(...) is called.





Guest Science

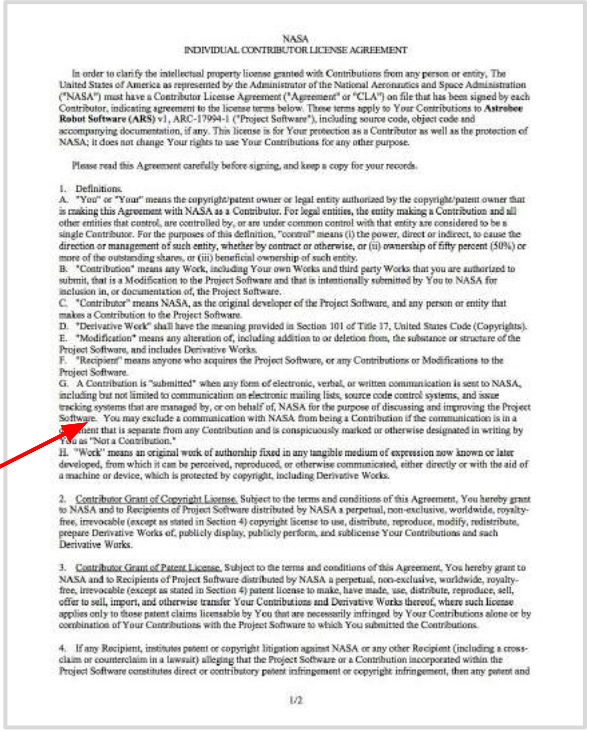
- **Guest Science** the mechanism for partnering with Astrobee to run experiments
  - Hardware - develop a payload with USB connectivity to Astrobee
  - Software - runs on the HLP
- Software development process
  - Android application loaded on HLP
  - Quad-core smartphone-clase ARM processor
  - Communication with ground over DDS
- First Guest Scientists - **REALM**
  - RFID-based Inventory management on ISS
  - REALM-2 adding reader to Astrobee
- More information
  - https://www.nasa.gov/content/guest-science-resources

Zero Robotics

- Programming competition for school kids in the US and Europe
- Every year there is a new game developed by MIT
- Teams write code to control a SPHERES robot in microgravity
- Teams first compete against each other in simulation
- Championships are held on the ISS using SPHERES robots
- Development underway to transition to Astrobee
  - MIT students currently prototyping game
  - New use cases for flight software and simulation
- For more information
  - http://zerorobotics.mit.edu
  - https://robotics.nasa.gov/events/zerorobotics.php

Open Source Release - Currently v0.4.x

- We're excited to offer a representative simulator for microgravity robotics research that is built upon open source community tools
- You can find all of our code here:
  - https://github.com/nasa/astrobee
- We actively encourage you to checkout the code
- Pull requests
  - We're always looking to improve our software
  - Contributions are warmly welcomed!
  - Although, you'll need to sign some paperwork before we can accept it
- Mechanical designs, schematics and firmware are not open source



NASA
INDIVIDUAL CONTRIBUTOR LICENSE AGREEMENT

In order to clarify the intellectual property license granted with Contributions from any person or entity, The United States of America as represented by the Administrator of the National Aeronautics and Space Administration ("NASA") must have a Contributor License Agreement ("Agreement" or "CLA") on file that has been signed by each Contributor, indicating agreement to the license terms below. These terms apply to Your Contributions to **Astrobee Robot Software (ARS)** v1, ARC-17994-1 ("Project Software"), including source code, object code and accompanying documentation, if any. This license is for Your protection as a Contributor as well as the protection of NASA; it does not change Your rights to use Your Contributions for any other purpose.

Please read this Agreement carefully before signing, and keep a copy for your records.

1. Definitions.
A. "You" or "Your" means the copyright/patent owner or legal entity authorized by the copyright/patent owner that is making this Agreement with NASA as a Contributor. For legal entities, the entity making a Contribution and all other entities that control, are controlled by, or are under common control with that entity are considered to be a single Contributor. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.
B. "Contribution" means any Work, including Your own Works and third party Works that you are authorized to submit, that is a Modification to the Project Software and that is intentionally submitted by You to NASA for inclusion in, or documentation of, the Project Software.
C. "Contributor" means NASA, as the original developer of the Project Software, and any person or entity that makes a Contribution to the Project Software.
D. "Derivative Work" shall have the meaning provided in Section 101 of Title 17, United States Code (Copyrights).
E. "Modification" means any alteration of, including addition to or deletion from, the substance or structure of the Project Software, and includes Derivative Works.
F. "Recipient" means anyone who acquires the Project Software, or any Contributions or Modifications to the Project Software.
G. A Contribution is "submitted" when any form of electronic, verbal, or written communication is sent to NASA, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, NASA for the purpose of discussing and improving the Project Software. You may exclude a communication with NASA from being a Contribution if the communication is in a document that is separate from any Contribution and is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."
H. "Work" means an original work of authorship fixed in any tangible medium of expression now known or later developed, from which it can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device, which is protected by copyright, including Derivative Works.

2. Contributor Grant of Copyright License. Subject to the terms and conditions of this Agreement, You hereby grant to NASA and to Recipients of Project Software distributed by NASA a perpetual, non-exclusive, worldwide, royalty-free, irrevocable (except as stated in Section 4) copyright license to use, distribute, reproduce, modify, redistribute, prepare Derivative Works of, publicly display, publicly perform, and sublicense Your Contributions and such Derivative Works.

3. Contributor Grant of Patent License. Subject to the terms and conditions of this Agreement, You hereby grant to NASA and to Recipients of Project Software distributed by NASA a perpetual, non-exclusive, worldwide, royalty-free, irrevocable (except as stated in Section 4) patent license to make, have made, use, distribute, reproduce, sell, offer to sell, import, and otherwise transfer Your Contributions and Derivative Works thereof, where such license applies only to those patent claims licensable by You that are necessarily infringed by Your Contributions alone or by combination of Your Contributions with the Project Software to which You submitted the Contributions.

4. If any Recipient, institutes patent or copyright litigation against NASA or any other Recipient (including a cross-claim or counterclaim in a lawsuit) alleging that the Project Software or a Contribution incorporated within the Project Software constitutes direct or contributory patent infringement or copyright infringement, then any patent and

1/2

Thank you & Video