Astrobee: Free-Flying Robots for the International Space Station

Oleg Alexandrov[2], Jonathan Barlow[2], Jose Benavides[1], Maria Bualat[1], Roberto Carlino[2], Brian Coltin[2], Jose Cortez[2], Earl Daley[1], Jeffrey Feller[1], Lorenzo Flückiger[2], Terrence Fong[1], Jesse Fusco[1], Ruben Garcia Ruiz[2], Kathryn Hamilton[1], Simeon Kanis[2], Aric Katterhagen[2], Yunkyung Kim[2], John F. Love[1], Michael McIntyre[1], Blair McLachlan[1], Andres Mora Vargas[3], Zack Moratto[1], Marina Moreira[2], Theodore Morse[4], Henry Orosco[1], In-Won Park[2], Christopher Provencher[1], Hugo Sanchez[1], Khaled Sharif[2], Ernest Smith[2], Trey Smith[1], Ryan Soussan[2], Andrew Symington[2], Rafael Omar Talavera[1], Vinh To[2], DW Wheeler[1], Jongwoon Yoo[4]

Abstract

The Astrobees are free-flying robots that operate inside the International Space Station (ISS) and were launched to the ISS in 2019. Since then they have successfully performed hundreds of activities in space supporting almost two dozen separate research projects. The robots were designed to overcome multiple challenges unique to the ISS environment, including safety, upgradeability and maintainability, limited mass and computation, and unique localization challenges from lack of gravity and a constantly changing environment. This article provides an overview of Astrobee, from hardware and software design to deployment results and activities.

I. Introduction

In 2019, three Astrobee robots (Fig. 1) replaced the Synchronize, Position, Hold, Engage, Reorient, Experimental Satellites (SPHERES) that have supported Guest Science research on the ISS since 2006. The SPHERES were among the most-used ISS payload facilities [66]. Like SPHERES, the Astrobees' primary purpose is to support guest scientists conducting research on the ISS, but they go beyond SPHERES in many ways: (1) Astrobee is designed to minimize burden on astronauts by executing most experiments fully autonomously. Because Astrobee uses battery-powered propulsion and autonomous docking to recharge, astronaut support is not required to resupply consumables. Astrobee was designed from the beginning to operate safely without astronaut supervision, incorporating safety controls such as collision mitigation and use of non-flammable materials. (2) Astrobee uses a vision-based navigation system that does not require installing any new infrastructure in the ISS interior, enabling it to operate throughout the U.S. Orbital Segment of the ISS. (3) Astrobee offers many new capabilities to guest scientists, including multiple payload bays for mounting Guest Science hardware, six built-in cameras of various types, substantially increased computational capabilities, and more.

In addition to supporting guest scientists, Astrobee was designed to meet two further objectives. First, future missions of NASA's Moon-to-Mars exploration architecture will be uncrewed for extended periods. This means Intra-Vehicular Robots (IVR) will be needed to provide vital caretaking services that are currently performed by astronauts on the ISS. Astrobee serves both as a feasibility demonstration and a reference design for future mission-critical IVR systems.

Second, Astrobee is designed to enable automating some ISS services, saving astronauts time so they can better support ISS science utilization. An Astrobee can serve as a flying camera that can be repositioned without astronaut assistance. Other possible use cases include performing audio surveys and video surveys, both recurring tasks that require astronauts to laboriously take instrument readings throughout the ISS.

This article first presents Astrobee's concept of operations (§II) and then the Astrobee hardware (§III) and software (§IV). Special attention is paid to what has been the biggest continued challenge for Astrobee, localization (§IV-H). Next, we survey results from Astrobee's numerous on-orbit activities (§VI). Finally, we discuss lessons learned (§VII) and related work (§VIII).

II. Concept of Operations

A major design driver for Astrobee operations was the need to minimize use of astronaut time, which is considered the most precious resource of the ISS program. Minimizing astronaut burden both greatly enhances the productivity of Guest Science, and strengthens the case that Astrobee is a valid demonstration for future mission-critical IVR systems that will need to operate without any astronaut support.

Corresponding author: andres.moravargas@nasa.gov
[1] NASA Ames Research Center, Building N269, Moffett Field, CA 94035.
[2] KBR Wyle Services/NASA Ames Research Center, Mountain View, CA, 94035
[3] Astrion/NASA Ames Research Center, Mountain View, CA, 94035
[4] Stinger Ghaffarian Technologies (SGT)/NASA Ames Research Center, Mountain View, CA, 94035

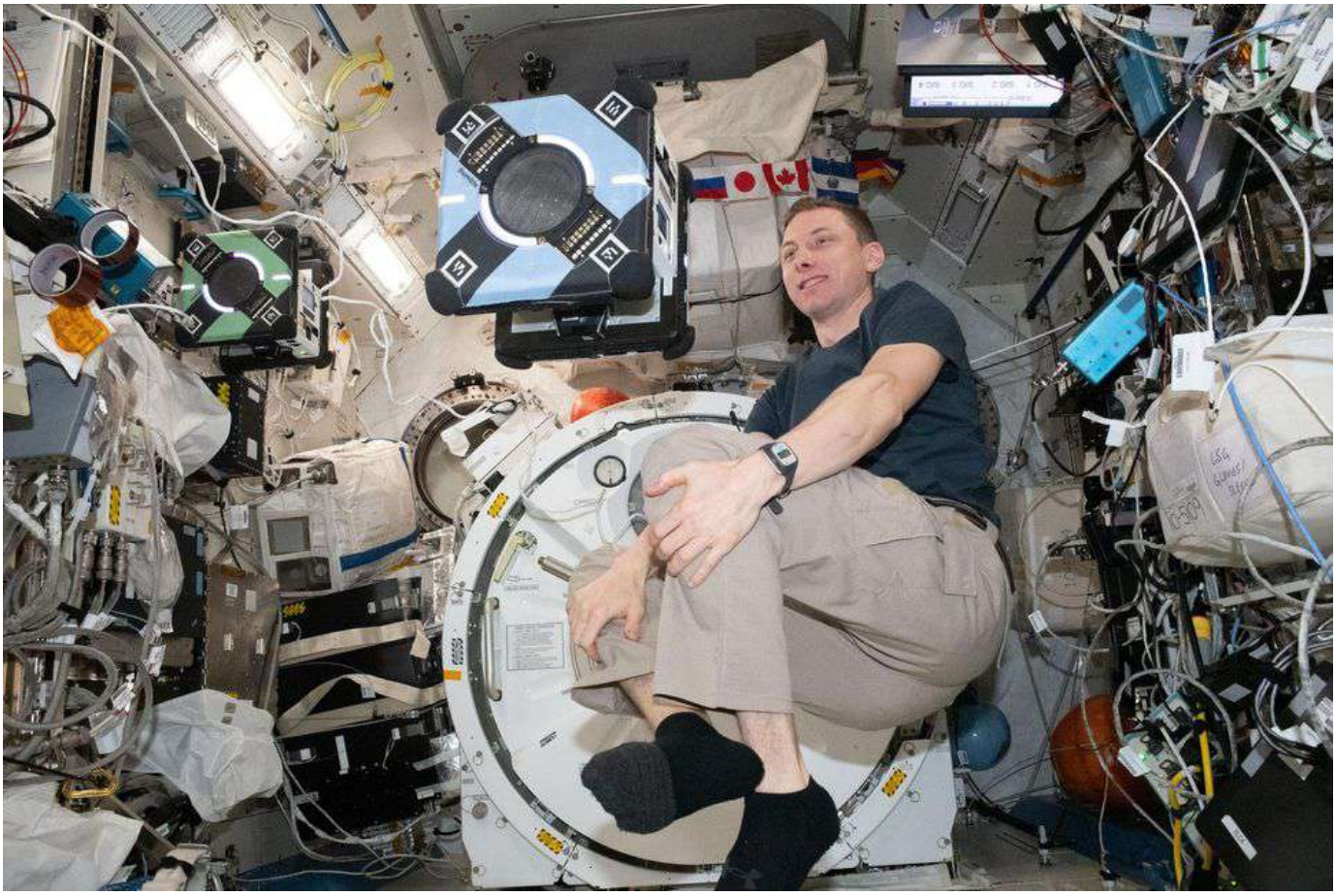


Fig. 1: Astrobee robots fly inside the ISS along with NASA Flight Engineer Woody Hoburg. Credit: NASA/JSC.

For Astrobee's predecessor SPHERES, each activity required astronaut time to retrieve the SPHERES unit from storage, replace its battery pack and CO_2 canister, set up a crew laptop to communicate with SPHERES and run the experiment, oversee the entire SPHERES activity, and finally do teardown. A typical session might require four hours of astronaut time. This burden very much limited the number of on-orbit experiment sessions available to guest scientists. With this context in mind, Astrobee was designed to use battery-powered fans for propulsion, autonomously dock to recharge, and operate safely without astronaut supervision.

Three main use cases drove requirements during Astrobee development:

- **Guest Science.** Astrobee's primary objective is to serve the needs of guest scientists. Of course, it is difficult to predict what will interest future researchers, but the general theme is improving flexibility and configurability. Example derived design elements: (1) Holonomic thrust capability, allowing Astrobee propulsion to simulate any spacecraft propulsion system. (2) Ability to carry payloads. (3) High thrust performance, in case experiments require significant force to be applied to a payload, like the SPHERES-Slosh experiment [93].
- **Free-flying camera.** Astrobee can act as a mobile camera to stream video of astronaut activities, controlled by ground operators without astronaut assistance. Example derived design elements: (1) High-quality camera (SciCam) and compressed video streaming capability. (2) Ability to perch on handrails for less disruptive observation. (3) Multiple-hour flight time to enable continuous observation of lengthy activities.
- **Sensor surveys.** Astrobee can collect sensor data sets throughout the ISS, automating what is otherwise a time-consuming astronaut task, and improving value by providing precise position data for sensor samples. Example derived design elements: (1) Ability to navigate across multiple ISS modules without requiring infrastructure changes such as marker tags. (2) Autonomous execution of plans containing motion and sensing commands. (3) Operator tools for editing plans and managing plan execution.

Table I summarizes many of Astrobee's operational phases and modes, including key capabilities. Data flow is discussed in §IV-A.

Phase	Mode	Description
Ground	Testing	Astrobee is tested in simulation, in 3-DOF under its own power on the Ames Granite Table (§VI-B).
Launch	Storage	Astrobee complies with ISS requirements and packed in foam for launch and return.
On-orbit	Flying	Each Astrobee can fly for $\sim$ 2.5h - 3h on a battery charge depending on the activity's requirements (§III-C). Astrobee can fly throughout the U.S. segment of the ISS (§IV-G), communicating via WiFi (§IV-A).
	Guest Science	Guest scientists add software or hardware payloads to Astrobee. Payloads that use the quick-release levers can be swapped by astronauts quickly with no tools (§III-G).
	Docking	Each Astrobee autonomously berths on a docking station with power and Ethernet (§III-K).
	Perching	Astrobee autonomously perches on handrails and disables propulsion, minimizing noise and power use (§III-H). The arm pans and tilts to point Astrobee's cameras.
	Hibernating and Storage	Two Astrobees hibernate and charge on the dock (§III-C). Ground operators can wake the robots via the dock. A third robot is stowed without batteries.
	Handling	Astronauts can easily grasp and move Astrobee including when perched or docked. Fold-out velcro straps make temporary stowage easier.
	Maintenance	Astrobees are designed for easy on-orbit replacement or upgrades (§III-B).

TABLE I: Astrobee operating phases and modes

A. Astronaut Interaction

Astrobee requires no astronaut support for most activities, with a few exceptions:

- Astronaut support is important during initial on-orbit commissioning of each Astrobee.
- Astronauts are needed to set the proper lighting conditions at the JEM and move equipment to stowage locations prior to Astrobee operations.
- Astronauts are needed to swap Guest Science payloads.
- Astronauts may be needed for occasional maintenance, such as vacuuming air intakes to reduce dust accumulation, or replacing or upgrading parts.
- If an Astrobee gets stuck or lost and the ground operator can't recover, an astronaut may be needed to return it to the dock for debugging, or to power cycle it.
- Finally, some Guest Science experiments require astronaut support for tasks the robot is not capable of, such as experiment-specific setup and cleanup.

Astrobee is designed to minimize impact on astronaut health, safety, and productivity. Astrobee propulsion was optimized for minimal acoustic noise. Many aspects of Astrobee design relate to astronaut safety, ranging from soft bumpers and fabric covers on the propulsion modules (in case of collision with astronauts) to screens at propulsion intakes (to avoid entangling astronaut hair).

Flying cameras that can travel throughout the ISS present a novel challenge for astronaut privacy and respecting payload proprietary restrictions. Video and audio from the ISS are strictly controlled to prevent privacy breaches. All live video is monitored by the ISS Video Control Center which cuts the video if needed. All recorded images and video undergo an additional review process before internal or external release.

B. Operator Interaction

Operators on the ground monitor Astrobee's execution at all times while it is in flight. Initially, operations were managed by the Astrobee Facility team from the Multi-Mission Operations Center at Ames Research Center (ARC). However, operations shifted to be fully remote during the pandemic and have largely remained in that mode ever since.

The Astrobees typically operate autonomously, executing custom plans or controlled by Guest Science software. However, for simple tasks, or to recover from failures, ground operators may teleoperate the robots. Real-time operator interaction is challenging due to communication latency between actions— as such, teleoperation commands are typically "go to this pose" rather than commanding with a joystick. An additional challenge comes from regular Loss of Signal (LOS) periods when the communications link to the ISS is down. LOS periods typically vary from several seconds to tens of minutes— depending on the length, operators may proceed as normal, or command Astrobee to float in place, or return to the dock.

C. ISS Environment and Other Design Considerations

The ISS environment uses an Environmental Control and Life Support System (ECLSS, [79]) to maintain the internal atmospheric conditions such as temperature, air composition and pressure. The temperature ranges between 18 to 27°C, the

air is composed of approximately 79% nitrogen and 21% oxygen, and the nominal total pressure is maintained at 101.3kPa. These conditions are similar to Earth's at sea level.

When designing a payload for use inside the ISS, the biggest priority is the crew's safety. With no published safety ISS payload operation ICD, the burden of guaranteeing the safety of the crew lies on the payload developer. Each aspect of interactions between items inside the ISS, crew, and the Astrobee is analyzed independently through a series of Safety Review Panels. Two examples of ISS safety guidance through these panels are the astronaut push off and the collision with ISS structure.

Astronaut push off force is considered the force a person weighting approximately 57kg would exert on a 10x10cm area by standing or pushing on it. The worst case scenario for the Astrobee interacting with other items inside the ISS is colliding with the windows inside the JEM. The maximum velocity the Astrobee robots should achieve is 2.1 m/s. The velocity was determined by calculating the fastest achievable speed the robot could attained in a straight line from the far side of the Columbus module through NOD2 to the JEM windows. The velocity is calculated by physics principles that observe the Astrobee mass and the Astrobee maximum acceleration. Further details about these and other design considerations are given in Section III and III-J.

III. Astrobee Hardware

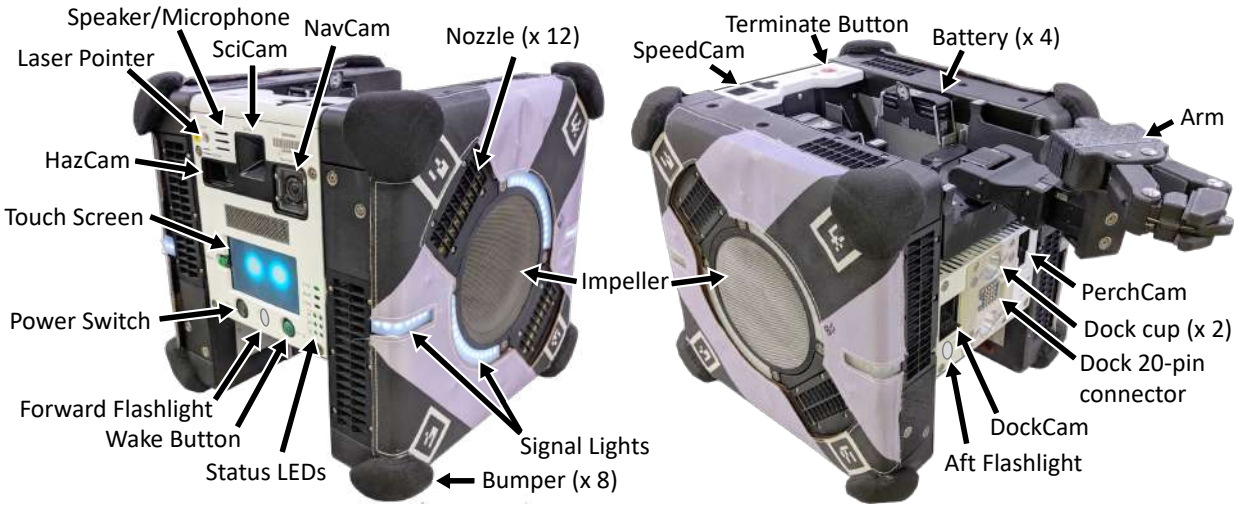


Fig. 2: Astrobee free flyer. Left: Forward and left sides. Right: Top and aft sides, with arm deployed.

Astrobee's external hardware features are illustrated in Fig. 2. We next discuss each of the key subsystems.

A. Propulsion

Astrobee's intra-vehicular (IV)-only propulsion subsystem uses two battery-powered fans and twelve adjustable flow-rate nozzles. Unlike SPHERES, which relied on consumable gas, Astrobee's propulsion relies on rechargable electricity alone.

1) Propulsion Trade Study: This propulsion approach was selected after an extensive trade study which evaluated several different high-level propulsion approaches. The main requirements were: (1) Enable autonomous resupply without astronaut support; (2) Provide holonomic motion control, i.e., the ability to produce instantaneous thrust in any direction and torque around any axis; (3) Produce 0.6 N max thrust on at least one motion axis; (4) Keep acoustic noise below 65 dBA at max thrust. Approaches were evaluated according to criteria including compatibility with ISS safety rules, low operational overhead, low complexity and development risk, low volume, and ability to support long sorties.

Some approaches were easily eliminated. Cold-gas thrusters, like the SPHERES CO_2-based system, develop thrust simply by storing propellant at high pressure. They were rejected due to concerns about enabling safe autonomous resupply (which would involve autonomous docking with a high-pressure fluid transfer system). Warm-gas thrusters are distinguished from cold-gas in that they use a chemical reaction to heat the propellant and can achieve higher exhaust velocity. They were rejected as being too hazardous to operate inside the ISS. Flapping approaches were considered intriguing, but too low in technological readiness.

Three more leading options were evaluated more seriously:

1) *Onboard compressor.* Essentially replicate the SPHERES cold-gas propulsion system, but replace the liquid CO_2 tanks with compressed air tanks filled by an onboard compressor.
2) *Distributed fans.* Use a system of six axial fans with reversible fan rotation, like a multi-rotor drone adapted for zero-gee.

3) *Centralized fans.* Use two impellers to drive air flow to many independently controllable nozzles.

In the first iteration of the propulsion trade study, we selected the distributed fans option. The onboard compressor option was rejected because we estimated it would have a very high power-to-thrust ratio. Furthermore, storing enough compressed air to last for an entire sortie would be impractical, due to its low energy density.

The centralized fans option was rejected due to perceived high complexity and development risk. The point design we evaluated (Fig. 3), derived from the Personal Satellite Assistant project [35], included relatively complicated ducts to the nozzles, an as-yet unspecified nozzle design, and required complementary reaction wheels to provide full 6-DOF control.

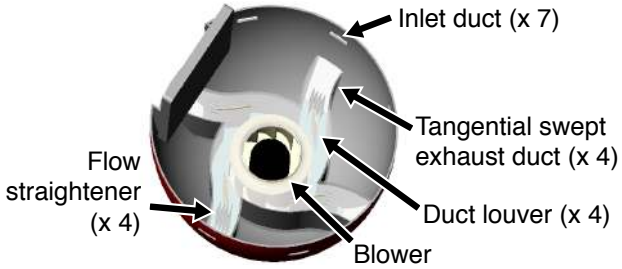


Fig. 3: Initial centralized fans concept, before redesign to present form

The distributed fans option appeared to offer the best combination of simplicity and high performance (Fig. 4).

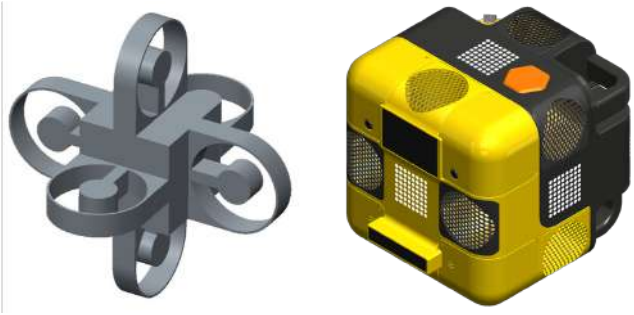


Fig. 4: Distributed fans approach. Left: Basic geometry. Right: Packaging concept with enclosed ducts.

However, as we proceeded into development, we ran into a number of unexpected challenges. The off-the-shelf axial fans we tested required on the order of a few seconds to go from zero to max thrust, due to rotor inertia. That raised concerns about the ability of the robot's motion control system to reject any sudden air disturbances, so we decided to upgrade the axial fans to use Variable Pitch Propellers (VPP). VPPs increase responsiveness by allowing the fan thrust to be set to full forward, full reverse, or zero thrust by adjusting the blade pitch, without changing the RPM rate of the rotor.

The VPP-based design progressed well at first, as tested with large-scale 3-DOF Astrobee prototypes using off-the-shelf hobby-grade VPP parts (Fig. 5). However, as we refined the design, we determined that the specified arrangement of six fans and the need for free air flow left uncomfortably little volume to accommodate other components such as core computing. Furthermore, we had growing concerns about our ability to develop custom VPP parts scaled down from the prototype size (23 cm diameter) to flight-relevant size ($\sim$9 cm), while retaining sufficient reliability for years of service life.

Fig. 5: Astrobee prototype 2 with VPP fans, testing on the Ames Granite Table

As such, we reopened the trade study with an added emphasis on reliability. The onboard compressor approach was again rejected, this time more conclusively, after quantitative performance evaluation. The centralized fans approach was ultimately

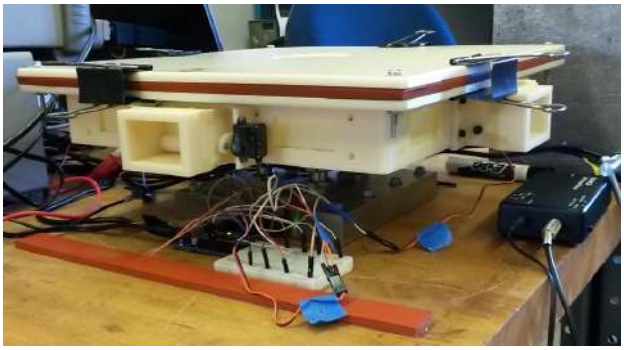


Fig. 6: Measuring centralized fan thrust performance with a two-nozzle prototype mounted on a force sensor

selected because with more effort invested, including a proof-of-concept test (Fig. 6), we landed on a superior point design that replaced complicated ducting with a plenum and added more nozzles to eliminate the need for reaction wheels. This approach ultimately went to flight.

2) Propulsion Design: Each Astrobee has two propulsion modules, occupying the entire left and right sides of the robot. Each propulsion module is built around a plenum. A single centrifugal impeller draws air in through an intake; the air lightly pressurizes the plenum and then flows out through six nozzles (Fig. 7). Each nozzle blows air in a fixed direction with an adjustable flow rate (Fig. 8). With six independently controllable nozzles on each propulsion module, the robot has twelve nozzles in all, sufficient to provide holonomic control (Fig. 9).

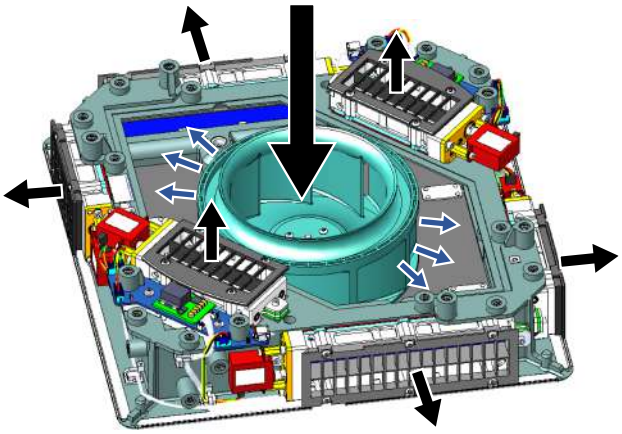


Fig. 7: Air flow through an Astrobee propulsion module (cover removed to expose plenum cavity)

The two propulsion modules are interchangeable and can be exchanged or replaced while the Astrobee is orbit. Their impellers rotate at the same speed in opposite directions, canceling gyroscopic moment and drag torque. The impeller speed is adjustable to trade peak performance versus reduced power and acoustic noise. By building each propulsion module around a single impeller, the impeller diameter can be maximized, achieving the same overall thrust with lower RPM rate, power, and noise. Impellers were selected over axial fans because they are better at maintaining flow rate performance in the presence of a pressure differential, like the one induced by drawing air through the screen at the propulsion module intake.

Axis	Nozzles Used		
	Force	**Torque 1**	**Torque 2**
+X	RX-, LX-	RZ-, LZ+	AY-, AY+
-X	RX+, LX+	RZ+, LZ-	FY-, FY+
+Y	FY-, AY-	RX+, RX-	LZ+, LZ-
-Y	FY+, AY+	LX+, LX-	RZ+, RZ-
+Z	RZ-, LZ-	RX+, LX-	AY+, FY-
-Z	RZ+, LZ+	RX-, LX+	AY-, FY+

TABLE II: Matched pairs of nozzles that produce pure force or pure torque

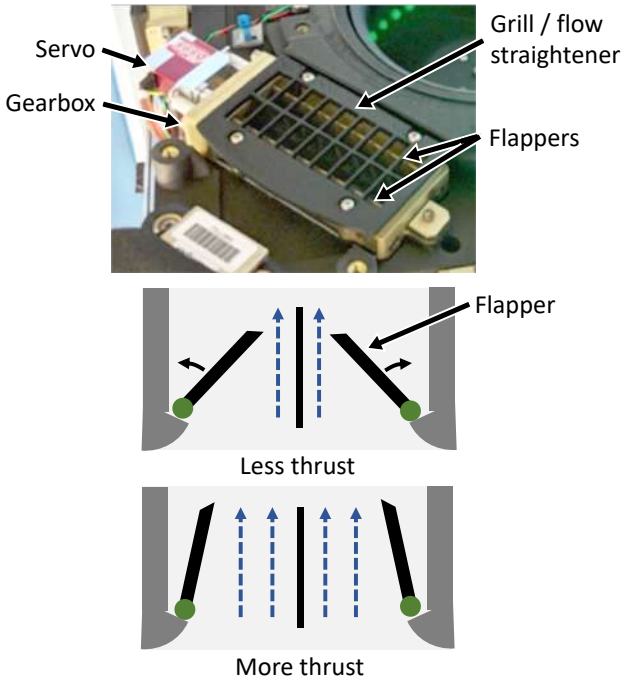


Fig. 8: Astrobee nozzle. Top: Nozzle detail photo, with propulsion module cover removed. Bottom: Cross section showing how flappers control thrust.

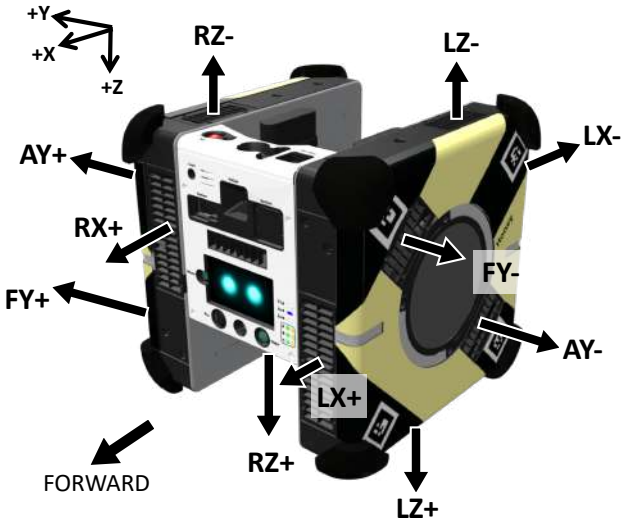


Fig. 9: Astrobee nozzle geometry. Let $\mathbf{F}$ denote nozzle thrust direction and $\mathbf{r}$ nozzle displacement from the robot center. Paired nozzles have parallel thrust direction $\mathbf{F}$ and lever arm $\mathbf{r} \perp \mathbf{F}$. Pure force nozzle pairs have the same sign for $\mathbf{F}$ but opposite sign for $\mathbf{r} \perp \mathbf{F}$, so the forces add but the torques $\mathbf{r} \times \mathbf{F}$ cancel. Pure torque nozzle pairs have opposite sign for both $\mathbf{F}$ and for $\mathbf{r} \perp \mathbf{F}$, so the forces cancel but the torques $\mathbf{r} \times \mathbf{F}$ add.

Each Astrobee has nozzle thrust vectors offset so they do not pass through the robot's center of mass. Instead, they are arranged in pairs such that for any translation axis (+/- X, Y, Z) there is a matched pair of nozzles that together provide a pure force along that axis (Table II). Similarly, for any rotation axis, there is a matched pair of nozzles that provide a pure torque about that axis. In fact, there are two redundant nozzle pairs available for each pure torque.

It is well-known that holonomic control of a rigid body moving in n-DOF requires $n + 1$ unidirectional thrusters [31]. With $12 > 7$ nozzles in a symmetrical arrangement, Astrobee is able to generate force and torque with high efficiency in more directions. The best efficiency occurs when the desired force direction happens to align with the direction of one of the balanced nozzle pairs. Applying force in other directions requires combining more nozzles, with part of the force from each nozzle wasted as cosine loss and in canceling undesired components of the force from other nozzles. In the limit when alignment with the nozzles is very poor, achieving the desired force direction could require almost perfect cancellation of large

input thrust vectors to form a small resultant vector, making the force allocation problem poorly conditioned and very sensitive to errors in the propulsion system calibration. Astrobee's nozzle arrangement mitigates these concerns.

Each Astrobee has holonomic control, but does not have symmetrical performance. It typically flies along a straight trajectory facing forward, with most of its thrust directed forward (starting) or aft (stopping). By design, thrust performance is greatest on the X axis (forward/aft) where the nozzles have larger open area. On the Y and Z axes, the nozzle open area is smaller, reducing nozzle size and mass and enabling finer control at low flow rates. Performance is weakest on the Y axis (left/right) because constructing a pure Y force happens to require using two nozzles on the same propulsion module, thus drawing power from only one impeller rather than both. Y axis performance could have been improved by routing a pressure-equalizing conduit between the two propulsion modules, but that approach was rejected to save space and maintain modularity.

Likewise, Astrobees have asymmetrical attitude control performance. The analysis is similar, but an additional factor is that the offset distance between the center of the nozzle and the desired torque axis varies. Maximum torque turns out to be about twice as great for yaw as for pitch. Nevertheless, the attitude control authority is considered more than adequate on all axes, and was not a major design driver.

An important design driver for Astrobee was propulsion efficiency. In typical spacecraft design, the achievable Δv for a given propellant mass fraction scales linearly with the thruster exhaust velocity v_e. However, as an aircraft, Astrobee has unlimited propellant, and instead focuses on power efficiency of thrust (W/N), which scales *inversely* with v_e. Therefore, Astrobee is designed to operate efficiently with low plenum pressure ($\sim$0.1 psi) and exhaust velocity ($\sim$11 m/s). For comparison, the SPHERES satellites use extra-vehicular (EV)-analog cold gas thrusters that operate at $\sim$25 psi with exhaust velocity $\sim$250 m/s.

Operating at low pressure also simplifies the design, eliminating the need for a compressor, and reducing the need for tight seals. The trade-off is that achieving sufficient thrust requires large open area for each nozzle (14-24 cm^2, compared to $\sim$0.005 cm^2 for SPHERES), and together the nozzles occupy $\sim$4% of the total robot surface area.

Although considerable design effort was invested in minimizing acoustic noise of the impeller, an unexpected twist was that the nozzle servos also contributed significant noise. Individual stand-alone servos subjectively seemed quiet, but we found that attaching nozzles to the propulsion module significantly amplified their noise, perhaps because the thin aluminum base plate of the plenum acted as a sounding board. This issue was mitigated by integrating isolator pads made of sound-absorbing Sorbothane gel into the nozzle/plenum mechanical attachments.

For reliability and safety, it was important to enclose the moving parts of the propulsion module. The air intake has a fine screen ($\sim$1 mm) to avoid drawing in astronaut hair or drawing in floating particles and ejecting them near astronaut eyes. Each exhaust nozzle is protected by a coarser grill to avoid pinched fingers or other jams.

3) Propulsion Modeling: Our design optimization process used well-known affinity laws to predict how the performance of a specified fan or nozzle type would change at different sizes and different operating points (RPM, volumetric flow rate, and pressure differential). We calibrated these models empirically for a variety of fan and nozzle types using lab testing with off-the-shelf or 3D printed parts (Fig. 6). This approach allowed us to quickly investigate the different design options in the trade study. It also provided critical information for tuning the flight design, guiding us to select more efficient shapes for the intake and exhaust nozzles and to increase the diameter of the fans to reduce acoustic noise.

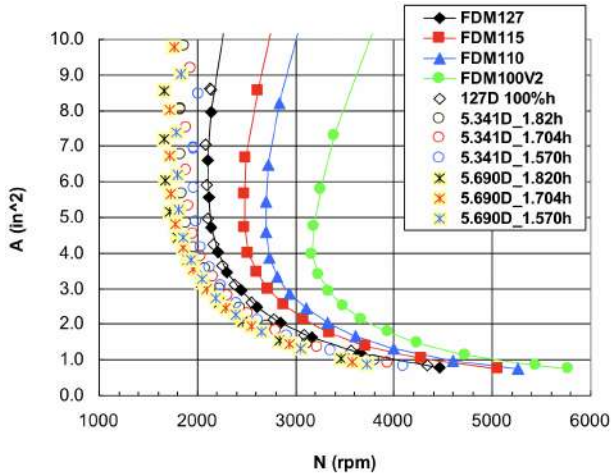


Fig. 10: Constant-thrust curves ($T = 0.3$ N) for multiple impeller sizes and shapes. FDM curves refer to the curves of 3D printed nozzles with variations using different flappers, grills, and overall dimension variations. Area was changed by changing the flapper angle either with a screw clamp or 3D printed with that area. Numbered items were standard calibrated nozzles (circular nozzles with smooth openings) with an area that was changed by adding or subtracting additional calibrated nozzles.

An example result is shown in Fig. 10. In this plot, there are multiple curves representing different fan sizes and shapes. The curve for each fan shows all the combinations of fan RPM rate N and nozzle open area A that achieve the required maximum thrust (0.3 N for one nozzle). Of course, we want to minimize N, in order to reduce acoustic noise and power consumption. The key insight is that, for the higher-performing fans, there is a significant benefit from increasing A up to $\sim$5 in^2 (32 cm^2), but little benefit beyond that point. This particular analysis drove the eventual flight design. The selected fan type is similar to the leftmost curve in the plot, and we chose to target $A = 5$ in^2 for the largest nozzles[1], which is an order of magnitude larger than the nozzles in our initial concept drawings.

Propulsion modeling also plays a role in motion control. Here, our model can be summarized as:

$$\begin{bmatrix}\mathbf{F}\\ \boldsymbol{\tau}\end{bmatrix} = M\mathbf{u} \tag{1}$$

$$u_i = 2A_i C_{d,i}^2 \Delta P \tag{2}$$

$$A_i = k_1 - k_2 \cos(\theta_i) \tag{3}$$

$$\Delta P = g\left(\sum_i \frac{u_i}{C_{d,i}}\right). \tag{4}$$

The variables are:

- $\mathbf{F}$ and $\boldsymbol{\tau}$ are total force and torque, expressed in the Astrobee's body frame.
- Each $u_i > 0$ is the magnitude of thrust produced by nozzle i.
- A_i is the open area of nozzle i.
- ΔP is the pressure differential between the plenum interior and the ambient air.
- θ_i is the flapper angle for nozzle i.

The calibrated values are:

- M is the 6×12 system matrix that relates individual nozzle thrust to total thrust. If each nozzle i has position $\mathbf{r}_i$ and direction $\mathbf{p}_i$, each row i of A is a concatenation of $\mathbf{r}_i$ on the left and $\mathbf{r}_i \times \mathbf{p}_i$ on the right. The $\mathbf{r}$ and $\mathbf{p}$ values were extracted from the Astrobee CAD model.
- $C_{d,i}$ is the discharge coefficient of nozzle i, a measure of efficiency. Values ($\sim$0.9) were measured empirically.
- The constants k_1 and k_2 relating flapper angle to nozzle open area are computed from the nozzle dimensions in the CAD model.
- The function g relates the pressure differential in the plenum to a normalized measure of total thrust. (At higher total thrust, total flow rate is higher and plenum pressure is lower.) Evaluating g requires empirically calibrating the impeller fan curve (pressure versus flow rate) and solving the steady-state exit equations for the plenum.

In Astrobee motion control, the force allocator (§IV-G) is given target force and torque and must calculate the corresponding flapper angles. They are computed as follows:

- Solve (1) for $\mathbf{u}$ under the constraint that each $u_i > 0$. This can be seen as finding a feasible solution to a linear program (LP). In practice, rather than using a general-purpose LP solver, our implementation solves (1) using a matrix pseudo-inverse, then corrects negative values by adding scaled null vectors (pre-computed combinations of four nozzles that produce zero net force and torque).
- Compute ΔP using (4). For real-time onboard use, the function g is implemented using a pre-computed lookup table.
- Solve (2) for each nozzle open area A_i.
- Solve (3) for each flapper angle θ_i.

B. Structure

Astrobee's structural design balances requirements on mass, volume, modularity, and geometry constraints on components such as cameras and bumpers.

The structure is designed to maintain integrity under a variety of loads, including: the ISS common launch vehicle vibration environment for soft-stow payloads, astronaut pushoff (up to 556 N applied over a 10 $\times$ 10 cm area), and collision with ISS structure at up to 50 cm/s, which is the maximum uncontrolled runaway velocity, and significantly higher than the normal operating maximum speed of 2.1 m/s.

Modularity was a key design driver. On-orbit replaceable units (ORUs) include the top forward sensor module, propulsion modules, nozzles, DockCam, PerchCam, touch screen, cooling fan assembly, the main processor boards, and payloads. For astronaut convenience, ORUs are designed to use a minimum number of fasteners, with all fasteners captive to avoid the risk of loose fasteners in zero-gee.

Each Astrobee has a core frame machined from aluminum 6061. In an unconventional approach for a spaceflight project, Astrobee relies heavily on 3D printing to enable complicated geometry that is otherwise too challenging or costly to manufacture.

[1]The final design achieved only $A = 3.7$ in^2 (24 cm^2) after accounting for occlusions due to the flapper mechanism.

The most commonly used material is non-flammable Ultem 9085. To save mass, the propulsion module plenum and impeller are 3D printed using high strength-to-weight carbon composite Windform XT 2.0 material. Overall, 3D printed materials account for 32% of Astrobee mass (not counting batteries).

C. Power

Each Astrobee carries an electrical power system (EPS) board controlled by a PIC microcontroller. Power to many components is individually switched, including each of the main processors, propulsion modules, payloads, and smaller devices such as the flashlights and laser. The EPS sends telemetry on internal temperatures, voltages, currents, etc. to the Low Level Processor (LLP). (The LLP is described in §III-D.)

Astrobees are powered by Inspired Energy ND2054HD34 rechargeable Li-ion batteries. The battery is in a 4S1P configuration and operates at a nominal 14.4 VDC, storing up to 49 Wh of energy. Each Astrobee carries between two and four batteries (trading off battery life versus minimum mass for improved acceleration), and a subset of the batteries can be hot swapped while the robot is operating. The batteries incorporate several types of safety features to avoid overheating, and produce System Management Bus telemetry about battery health and charge status. The EPS subsystem includes an individual charging circuit for each battery, as well as under-voltage cutoff circuitry, and also forwards battery telemetry to the main processors. With four batteries, Astrobee can operate for up to $\sim$2 hours of flying time (varying depending on what components and payloads are active).

Each Astrobee can run in a hibernation mode where only the EPS subsystem is active. All other components, including the main processors and cooling fans, are turned off to minimize power consumption and wear. While hibernating, the EPS waits for either the wake button to be pressed by an astronaut, or for the dock to send a wake signal on the I^2C bus, relayed from a ground operator. The same I^2C bus provides the link for the Astrobee to command the dock to retract its retention magnets during undocking.

D. Computing

Name	**Function**	**Model**	**Cores**	**Max Clock (GHz)**	**Max Power (W)**
Low-Level Processor (LLP)	Reliably run high-rate control code in isolation from other software.	Wandboard Dual	2	1.0	3
Mid-Level Processor (MLP)	Run most Astrobee software, including compute-intensive computer vision code, in isolation from Guest Science.	InForce IFC6501 (Qualcomm Snapdragon 805)	4	2.4	10
High-Level Processor (HLP)	Primarily for Guest Science. Also manages SciCam video compression, touch screen, and other less critical devices.	InForce IFC6601 (Qualcomm Snapdragon 820)	4	2.2	10

TABLE III: Astrobee main processors

Astrobee's computing balances design drivers of low volume and power consumption with high computing performance. Primary computing is split over three main processors (Table III) to better isolate robust high-rate control code from less predictable software.

The main processors, all ARM processors delivered in system-on-module format and derived from smart phone technology, ride on two custom carrier boards that are designed to enable a possible on-orbit upgrade by astronauts later in the project life cycle (almost all I/O is routed to the carrier boards through stack connectors to simplify swapping).

Each Astrobee also carries several PIC microcontrollers for power management and distributed control of the propulsion modules, signal lights, and perching arm.

E. Sensors

Each Astrobee carries a suite of six commercial off-the-shelf (COTS) cameras. Their functions are summarized in Table IV. Three cameras point forward, two point aft, and one points up (Fig. 11). In addition to the cameras, each Astrobee carries an Epson M-G362 6-axis Inertial Measurement Unit (IMU) mounted near its geometric center.

To estimate the unified extrinsic calibration for these sensors, we collect simultaneous data logs that include vigorous motion with calibration targets in view, and use the Kalibr toolbox [47]. We also take advantage of the fact that the LIDAR sensors can produce amplitude images of calibration targets [27].

Name	Primary Purpose	Type	Sensor Make/Model	Pixels	Field of view (°)
NavCam	General navigation	RGB	TheImagingSource DFM 42BUC03-ML	1280 × 960	130 × 98
DockCam	Dock-relative navigation	RGB	TheImagingSource DFM 42BUC03-ML	1280 × 960	120 × 90
SciCam	High-quality video	RGB	InForce ACC-1H70	5360 × 4032	55 × 41
HazCam	Obstacle detection	LIDAR	PMD CamBoard Pico Flexx	224 × 171	62 × 47
PerchCam	Handrail-relative navigation	LIDAR	PMD CamBoard Pico Flexx	224 × 171	62 × 47
SpeedCam	Over-speed propulsion cutoff	Greyscale + Rangefinder + Accel.	PixHawk PX4FLOW + TeraRanger One + AdaFruit MMA8451	752 × 480	25 × 19

TABLE IV: Camera specifications

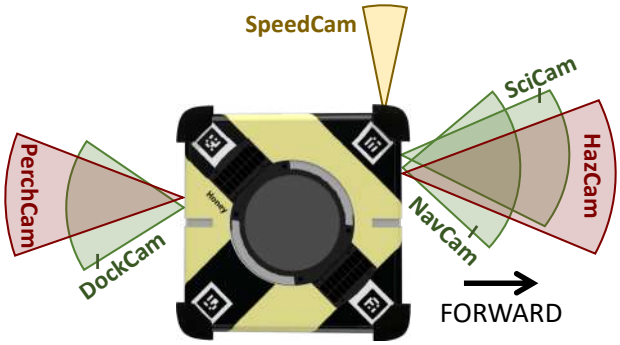


Fig. 11: Astrobee camera fields of view, seen from the Astrobee's right side.

F. Thermal

Astrobee's thermal subsystem must reject heat in spite of the lack of gravity-driven thermal convection in the zero-gee environment. Each heat source exposed to astronaut touch must satisfy ISS safety touch temperature limits either by being passively safe (incapable of overheating an exposed surface), or single-fault-tolerant if active cooling is required. We can take advantage of higher safety limits where only brief incidental skin contact is expected and the surface material is thermally insulating (e.g. Ultem, Nomex fabric), thus slower to cause discomfort.

Each Astrobee has core avionics cooled by a pair of cross-flow blowers that run at all times (except when hibernating). Air is drawn in at intakes on the Astrobee's forward and top faces, directed to flow between boards of the avionics stack and over a heat sink linked to the Mid-Level Processor (MLP) and High Level Processor (HLP), then out exhaust vents at the aft end of the payload bays. The two blowers counter-rotate to cancel the gyroscopic moment and minimize disturbance to navigation. While air flow in the aft direction produces a net disturbance force, it is constant and small (<5% of propulsion system thrust).

Besides the core avionics, the other main sources of heat are the propulsion impeller motor and nozzle servos. The impeller motor is exposed to the avionics box cooling fan air flow, but also in contact with the propulsion module plenum base, so it can take advantage of free cooling from the propulsion air flow itself. (Even when the nozzles are closed, there is residual air flow due to leakage.) The nozzle servos are sufficiently isolated from exposed surfaces to be passively safe, and likewise get free cooling from propulsion air flow.

To address the need for single-fault-tolerance with respect to faults such as stalled motors, blocked air flow, failed fans, etc., there are several independent over-temperature cutoff circuits throughout the system.

The propulsion system could have been designed to draw air through Astrobee's core computing to provide free cooling. However, this coupling would have introduced additional design constraints on both sides (e.g. components in the core would need to be carefully arranged for efficient air flow; propulsion fans would need to continue operating even while docked, to provide cooling). Our selected modular approach eliminates these coupling concerns, and has simplified our prototyping and integration and test processes.

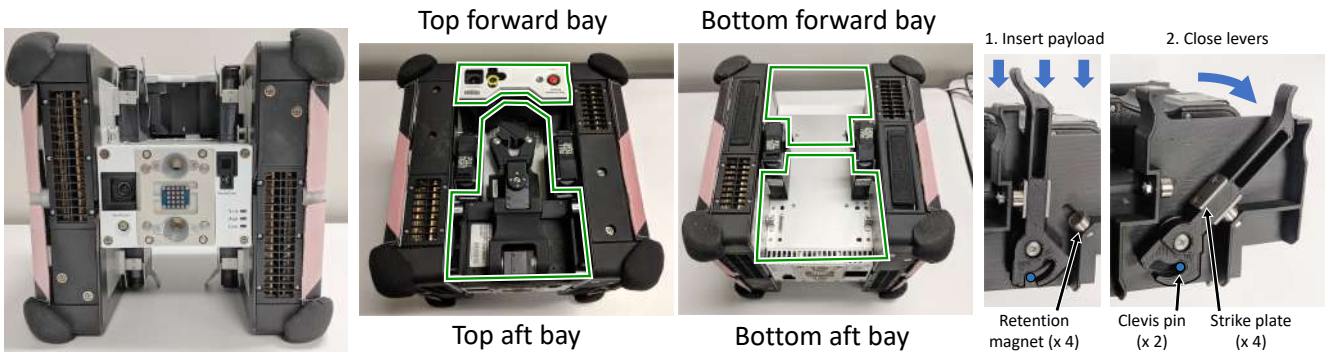


Fig. 12: Left: Aft view, no arm, showing open volume for payloads on the Astrobee's top and bottom. Middle: Payload bays from above and below. Right: Quick-release lever mechanism.

G. Payload Hardware

Astrobee has four payload bays, two on the top and two on the bottom (Figs. 12). Each provides the same interface for payloads. The top forward bay is not available for Guest Science because it is always occupied by Astrobee's baseline Top Forward Module, which contains four of Astrobee's six cameras. Up to three payloads can be operated simultaneously in the other three bays. The top aft bay is usually occupied by Astrobee's baseline arm, but the arm is easily swappable for a Guest Science payload. The two bottom bays are normally unoccupied. Guest Scientists are advised to design their payloads to stay entirely within the volume of the payload bays, where they are protected from collisions by the Astrobee's bumpers.

A payload port in each bay provides raw battery power at 14.4 VDC nominal and USB 2.0 data using a 31-pin blind-mate connector. We recommend that payloads connect to Astrobee's payload bay using our quick-release (QR) lever mechanism that was first implemented for Astrobee's baseline arm (Fig. 12), enabling an astronaut to swap the payload quickly with no tools. The levers can be actuated one at a time, important for astronauts who often have only one hand free. Magnets lock the levers at either Open or Closed until actuated.

H. Arm

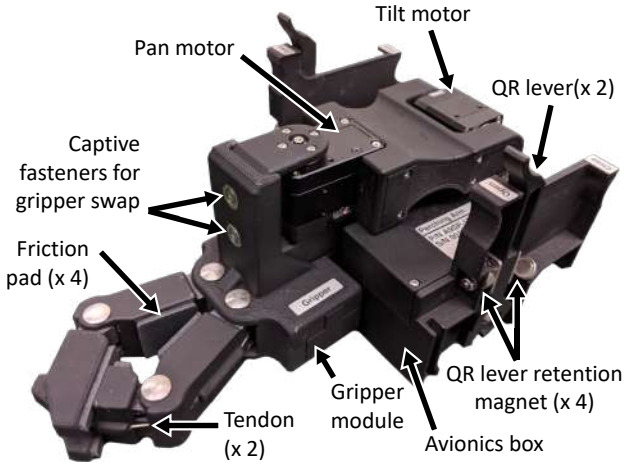


Fig. 13: The arm in stowed configuration.

Each Astrobee can carry an arm in its top aft payload bay (Figs. 2, 13) [81]. The arm's primary function is to enable autonomous perching on ISS crew handrails. While perched, the arm acts as a pan/tilt unit for forward-facing cameras like the SciCam (Fig. 14). Perching allows propulsion to be turned off, saving power and minimizing disturbance to astronauts. A secondary function is to enable future Guest Science research on robotic manipulation.

Each arm has three actuated degrees of freedom: two for the arm pitch joints, and one for the gripper. The arm length and joint arrangement are designed to maximize the range of pan/tilt motion when Astrobee is grasping a handrail, while also allowing the arm to stow fully inside the top aft payload bay, within the volume protected by the bumpers. While the arm is stowed and powered off, a pair of permanent magnets prevent it from drifting open; to unstow, the tilt motor can easily overcome the light retention force.

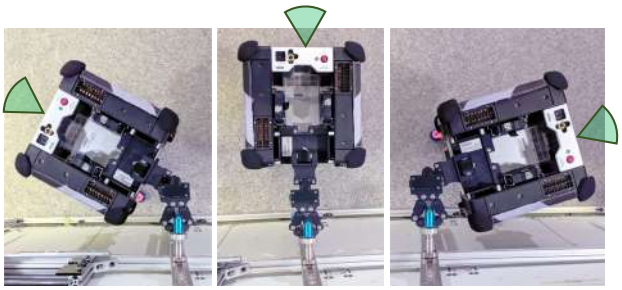


Fig. 14: The arm grasps a handrail and acts as a pan-tilt unit for pointing the SciCam (field of view indicated in green) and other forward-facing cameras.

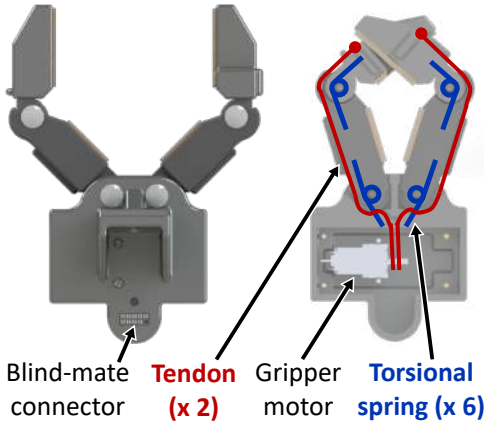


Fig. 15: The gripper. Left: Open configuration. Right: Closed configuration, cross-section exposing internal components.

The gripper uses a passively under-actuated compliant design (Fig. 15). For flexion, it uses torsional springs in the finger joints. For extension, a single motor reels in two tendons, one for each finger. This design means the springs passively hold the gripper closed when perching for extended periods. When the gripper is open, the passive back-drive resistance from the gripper motor is sufficient to overcome the springs and keep it open. The gripper is sized for grasping a handrail, but the geometry is also optimized to successfully grasp a wide variety of object shapes [24].

The arm mounts to Astrobee using the standard payload QR lever mechanism (§III-G).

Astrobees must tolerate astronaut pushoff forces of up to 556 N. To handle these forces without breaking, the arm and gripper are designed to be back-driveable: the arm joints disable themselves when a strong torque is detected, allowing them to be manually repositioned; the gripper will release its grasp when its springs flex under load, or when an astronaut manually opens or closes it. Together, these features also enable an astronaut to easily push a perched Astrobee out of the way in an emergency, or to manually perch an Astrobee on a handrail for convenient temporary stowage.

The avionics box in the arm houses a custom controller board based on a dsPIC33EP512MC806 microcontroller. The arm controller links to the MLP via USB 2.0, receives high-level commands (move to pan/tilt, open/close gripper), and publishes telemetry at 10 Hz. It communicates with the arm joints (Robotis Dynamixel XH430-W210-R servos) via daisy-chained RS485, and directly controls the gripper motor (Pololu 3057 brushed DC motor) by sending the drive waveform and receiving encoder ticks. Because the gripper encoder only provides relative position information, the gripper must periodically self-calibrate by fully extending the fingers to a hard stop, with full extension detected using motor current monitoring. The arm thermal design is discussed in [80]; it ensures compliance with ISS touch temperature safety standards by implementing passive and active thermal controls. Passive control uses bimetallic thermostats to prevent overheating in arm motors, while active control employs a current limiter circuit to protect the gripper motor from overcurrent conditions.

The design goal for arm pan/tilt max speed was 90 degrees of turning within 15 seconds. The joint motors, with 2.2 Nm of stall torque, provide more than sufficient acceleration. Based on ground testing, the factor limiting speed is the grip strength on the handrail; foam friction pads on the inside of the fingers act to minimize slip.

I. Human-Robot Interaction

Astrobees are designed to work well with astronauts. Several hardware components are included specifically to support co-located Human-Robot Interaction (HRI), including the touch screen, signal lights, speaker and microphone, and a laser pointer.

Beyond providing tools for future Guest Science, the touch screen and signal lights enable Astrobees to signal their operating state and motion intentions, so they become more predictable and less disruptive to astronauts. Multiple signal light animations were evaluated in a user study to determine how they would be perceived and interpreted [54]. In a special nod to astronaut privacy, blue lights on all four sides of an Astrobee turn on when its microphone is recording audio. To minimize distraction to astronauts, ISS human interface guidelines recommend that the signal lights should not use blinking or jerky motion; therefore, we designed signal light animations using a custom domain-specific language called LightFlow that encourages smooth spatial and temporal interpolation.

The touch screen's default behavior is to display animated virtual eyes. The eyes serve to make Astrobee's otherwise impersonal industrial appearance somewhat more engaging, and reinforce which side is its forward "face" (echoed by chevron patterns on the Astrobee's sides that point forward). Eye animations can signal an idle state (sleeping eyes), success (excited eyes) or a neutral ready state (attentive looking around and blinking). Additional animations can mirror and reinforce motion signals conveyed by the signal lights.

J. Safety

The highest priority of the ISS program is the safety of astronauts and the ISS itself. We discuss several key safety challenges. Touch temperature limits were discussed previously in §III-F.

Early in the Astrobee development process, we committed to avoiding any use of software to mitigate safety hazards. That commitment forced us to implement additional hardware safety controls, but it also gave us the great benefit that Astrobee software is not considered safety-critical per NASA Procedural Requirement 7150.2. Because Astrobee software is not safety-critical, many requirements related to formal software engineering practices and verification are relaxed, and it is much easier to allow future Guest Scientists to modify Astrobee's baseline software.

In the remainder of this article, we draw a distinction between *safety* issues, which affect astronaut or ISS health and must be mitigated, versus *mission assurance* issues, which threaten only Astrobee mission goals, for example, causing Astrobee damage, or causing operational disruptions that result in the Astrobees being grounded. Software controls *are* used in some cases to address mission assurance issues.

1) Collision Safety: For mission assurance, Astrobee attempts to avoid obstacles and limit its speed in software, which will be discussed later.

For safety for astronauts and the ISS itself, Astrobee's approach is to be light, slow, and soft. "Light" refers to an Astrobee's mass (∼10 kg, depending on configuration). "Slow" refers to the fact that an Astrobee can't accelerate to faster than ∼2.1 m/s, given physical limits on its thrust capability and room to maneuver inside the ISS. "Soft" refers primarily to the Astrobee's bumpers, but also to thinner foam and fabric skins covering most of the exposed area of the propulsion modules. Together, these factors allow us to bound the maximum impact kinetic energy and peak force.

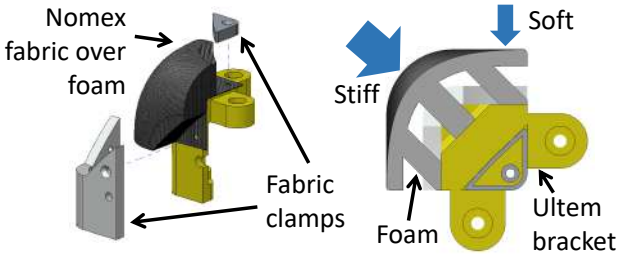


Fig. 16: Bumper. Left: Exploded view. Right: Cross section showing foam shape and non-isotropic stiffness.

Considerable effort went into designing and testing Astrobee's bumpers (Fig. 16). The bumpers are made of ARTi-LAGE, an impact-absorbing open-cell polyurethane foam with good wear properties, typically used for products such as seat cushions. The foam is wrapped in non-flammable Nomex fabric. The bumper's purpose is to keep impact peak force to below the astronaut crew pushoff force of 556 N in a 50 cm/s collision, achieving this with a bumper thickness limited to 13 mm.

An Astrobee's impact peak force depends on both speed and orientation (Fig. 17). Due to the Astrobee's cubic shape, if it strikes a flat or concave object, it always makes first contact at one or more bumpers. With a flat object, the peak force is maximized in orientations where the Astrobee's center-of-mass is in perfect alignment to distribute the load across one, two, or four bumpers impacting simultaneously. (In other orientations, some of the initial kinetic energy is translated to rotational energy and peak force is reduced.) The three "perfect" impact orientations were carried throughout the design and test process.

A challenge for bumper design is setting stiffness. A bumper that is too stiff will obviously exert more peak force; however, one that is too soft will bottom out before all of the impact energy is absorbed. Furthermore, once an optimal *total* target stiffness is calculated, the optimal *individual* target stiffness of each bumper depends on how many bumpers are involved in the impact. Astrobee's bumper design therefore uses inward-facing ribs to achieve non-isotropic stiffness when compressed from different directions (Fig. 16).

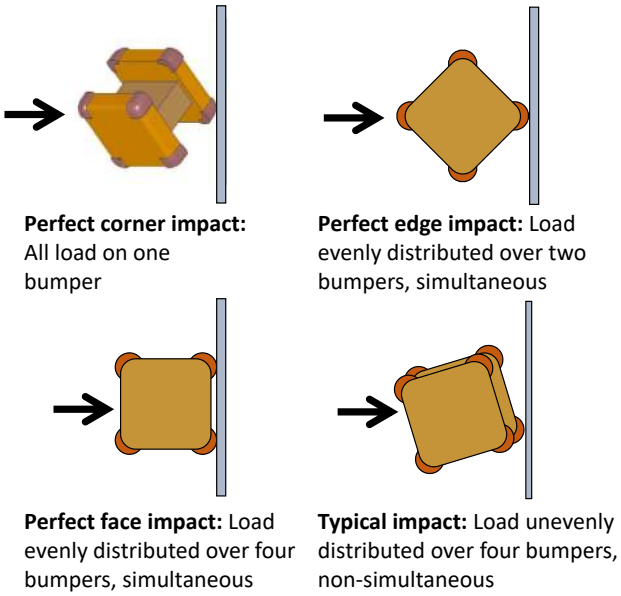


Fig. 17: Collision orientations

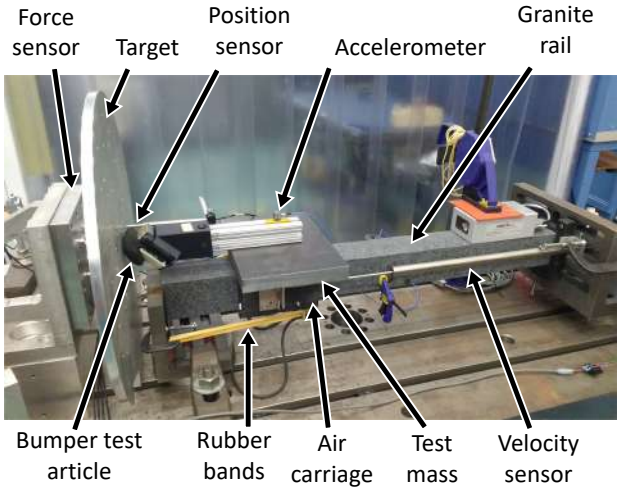


Fig. 18: Collision test rig

It was important to test bumper performance while placing the minimum amount of prototype hardware at risk. We evaluated bumpers using a collision test rig (Fig. 18). To simulate each collision orientation test case, an individual bumper was integrated with an appropriate test mass and mounted in the desired orientation. The integrated impactor sled rode an air carriage, and rubber bands were used to accelerate it to the specified test velocity and ram it into an aluminum plate mounted on a force table. The primary result was the measured impact peak force. Simultaneous logging with other sensors (position, velocity, acceleration, high-speed video) was used to provide context and validate the experimental setup.

Having verified the design performance using prototype bumpers, we proceeded to order the foam parts in bulk from the vendor, but a further challenge was a variety of quality control problems in the vendor's injection-molding process. The bumper stiffness varied significantly, requiring us to improvise a per-bumper acceptance test with a force-measuring press.

The analysis for third-tier collision controls was split according to what type of object the Astrobee collides with:

- **An astronaut.** Astronauts are much faster and more agile than Astrobees, and could theoretically injure themselves by running into an Astrobee (for example, if a sensitive area such as the orbit of the eye strikes an Astrobee corner). Unfortunately, there is no practical way to design Astrobee to prevent these collisions. We can only partially mitigate the risk with rounded shape and padding, and also by taking steps to improve astronaut awareness of Astrobees (e.g. mentioning Astrobee activities in the daily crew conference, and each Astrobee announcing its presence with visual and audible cues as appropriate). On that basis, the ISS program formally accepted the risk.
- **A glass window.** A typical ISS window has four glass panes. The two thicker middle *pressure panes* keep ISS cabin air

from escaping; there are two to provide redundancy. The pressure panes are sandwiched between two thinner panes that protect them from impacts; of these two, the *scratch pane* faces inward toward the cabin and incorporates a polycarbonate film that prevents glass from being released if it is fractured. Damage to the scratch pane would not put astronauts at risk, though it would certainly be taken very seriously; damage to one of the pressure panes would be much worse, since it would remove a level of redundancy protecting against catastrophic cabin depressurization.

Through extensive bumper testing and analysis, we established that the absolute worst-case impact force on a window would be less than 2700 N, sufficient to risk fracturing the scratch pane, but not a pressure pane. Furthermore, the lead-up to such a collision would involve a string of several independent unlikely events, making it highly implausible.

Next, as a mission assurance issue, we evaluated what over-speed cutoff threshold would protect the scratch pane. We needed the impact peak force to be below the 556 N maximum astronaut pushoff force that the scratch panes were verified to tolerate. The fastest successful test speed was 53 cm/s, leading us to specify a round number of 50 cm/s for the cutoff.

- **ISS structure.** Damaging ISS structure could be very serious. As a worst-case reference object for analysis, we took a structural panel 1.6 mm thick made of aluminum 7075. We used a finite-element analysis to estimate the strain energy at which the panel would rupture, and showed that between an Astrobee's limited impact kinetic energy and its internal structural compliance, even an absolute worst-case collision would not transfer enough energy into the panel to break it.
- **Projecting objects.** Arbitrarily shaped objects projecting from the walls may directly strike an Astrobee's rigid exterior without first contacting a bumper. However, this hardware is typically less critical than windows or structure, and in many cases it has sufficient compliance that impact peak force would be greatly reduced (e.g. ISS crew laptop mounted on a flexible Bogen arm).

2) Additional Safety Considerations: **Glass Parts** Because Astrobee's touch screen is glass and some of its cameras use glass lenses, it was important to mitigate the hazard of glass shattering in a collision (considered catastrophic, since glass fragments could blind an astronaut). As a result, every exposed glass part on an Astrobee is either protected by a transparent acrylic shield, contained by Teflon tape, or recessed sufficiently that typical projecting obstacles like an ISS astronaut handrail can't contact the glass in a collision. These features significantly complicated packaging all the cameras in the limited available volume of the robot.

Battery The hazards of lithium-ion batteries are well-known in the aerospace community. We selected our Inspired Energy batteries in part based on their multiple levels of safety features. Beyond the manufacturer's baseline guarantees, the ISS program did a rigorous safety analysis, directed us to change the batteries' factory firmware settings to mitigate a potentially hazardous undervoltage edge case, and performed additional lot testing to certify a pool of batteries for launch.

Electrical The Astrobee dock draws 120 VDC from the ISS power system and is subject to extra scrutiny around issues of grounding and isolation in order to avoid electric shock. The dock berth 20-pin connector is unusual in that its pins are always exposed in the ISS cabin; loopback pins are used to detect the presence of an Astrobee and enable charging current, preventing a short if conductive debris drifts onto the pins. Some Astrobee connectors that operate at higher current (e.g. battery terminals) are required to have a "scoop-proof" sleeve to prevent molten metal escape in a worst-case short.

Flammability and Off-gassing In order to operate without astronaut tending, Astrobee must not be considered flammable (unlike SPHERES). The usual ISS approach for managing flammability is simply to use materials that have been found to be non-flammable when tested in air up to 24.1% O_2. Workarounds are available, for example when the amount of flammable material is sufficiently small, or it is protected from ignition sources. Similarly, given the closed ISS environment, Astrobee materials must not off-gas noxious chemicals, with exceptions made for small quantities.

Following the standard approach, most of Astrobee's exposed surface is made of non-flammable 3D printed Ultem 9085 (forward and aft bezels), aluminum (payload bays), stainless steel (intake screens), or Nomex fabric (propulsion module skin and bumpers). Internally, the biggest flammable components are the propulsion module plenum and impeller, made from Windform XT 2.0, a flammable carbon fiber composite 3D printing material chosen for its high strength-to-weight ratio. These parts are isolated from the ISS cabin (e.g. exposed surfaces coated in non-flammable Kapton tape) and from possible internal ignition sources in the core avionics box. Many of Astrobee's smaller COTS avionics components are also flammable or include unknown materials, but are either too small to be considered a problem, or can be protected, for example by using Kapton tape or heat-shrink tubing.

Eye Astrobee's laser and flashlights are bright enough that nearby astronauts should not stare at them, but they are considered eye-safe due to the human blink reflex. Both components had to undergo brightness measurements. Although the flashlight brightness is comparable to a typical pocket flashlight, verifying its eye safety required analysis taking into account the LED's output spectrum, viewing angle, viewing distance, and lens dispersion.

K. Dock

The Astrobee dock has two berths, each providing power and Ethernet data to one Astrobee (Fig. 19). When the Astrobees are idle, typically two hibernate on the dock and one is stowed, powered off and with batteries removed.

The dock berth interface to the Astrobee is designed to enable docking with significant error tolerance and with an insertion force lower than the Astrobee maximum thrust. Each berth has two lances that mate with matching cups on an Astrobee,

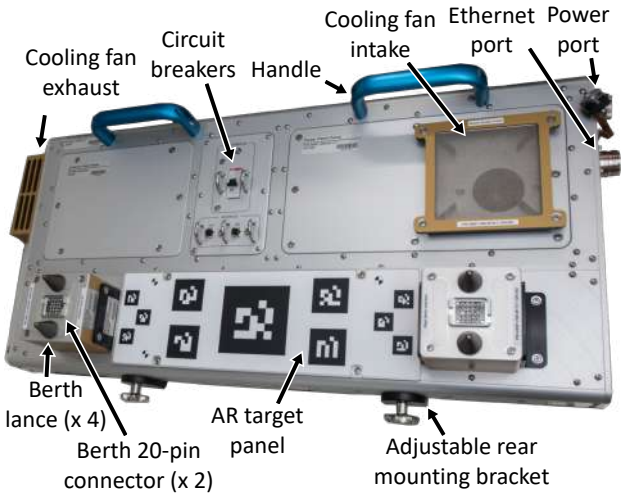


Fig. 19: Astrobee dock

accommodating and correcting up to ∼1 cm of alignment error. When fully mated, four permanent magnets just behind the surface of the berth generate retention force by attracting strike plates in the Astrobee, and spring-loaded pogo pins in the berth 20-pin connector contact pads in the Astrobee 20-pin connector. Compliance in the pogo pins accommodates any remaining alignment error. To enable undocking, the Astrobee can command the dock to retract the magnets further into the berth; they automatically extend again after a few seconds.

Design of the magnetic retention system evolved over the course of the project. Our requirements were that the magnets must detach when an astronaut tries to move the Astrobee (>22 N force), for example to clear an egress path in an emergency, but should not detach in cases of light incidental astronaut contact (<4.5 N force). As a result, an Astrobee's maximum thrust is far too low to overcome the magnets on its own. An earlier design iteration kept the magnets fixed and extended two push rods from the dock to disengage the Astrobee; however, during prototype testing we observed that this imparted unwanted rotation on the Astrobee as the rods extended. The final design has the benefits that the Astrobee tends to stay in place during magnet retraction, and all berth moving parts stay fully enclosed behind a sealed panel.

"AR target" fiducials on the dock are in view of the DockCam during docking approach, enabling more precise and robust localization.

Free space on the ISS is a precious resource. The dock's current installed location is near the port end of the Japanese Kibo module of the ISS. This location was selected to minimize disruption to astronaut activities; it is a low-traffic area near a dead end and far from the docking ports that are used for visiting cargo vehicles. (SPHERES operations happen nearby for the same reason.) In case the dock needs to be moved later, its design includes several options for astronaut reconfiguration. In its current location, four feet on the back of the dock attach to the wall using velcro. Each foot's position can be adjusted both horizontally and vertically so as to align with a good mount point on the wall. (In the unlikely event of a fire in the rack behind the dock, an astronaut can quickly move the dock out of the way by pulling the handles to detach the Velcro.) As another mounting option, a different set of adjustable brackets can thread into the seat track rails found throughout the ISS. Finally, the berth posts can be tilted up or down in case the dock is mounted in a confined space such that a level approach path would take the Astrobee too close to an obstacle.

L. Ground Facilities

Two gravity-offset testing labs were used to simulate Astrobee motion in zero-gee during testing. The Granite Lab, a flat table with air-bearings, allows an Astrobee to move freely in 3-DOF (x, y, and yaw). It is still in use, and it is described in detail in VI-B. Astrobee was also tested in the Micro-Gravity Test Facility (MGTF), shown in Fig. 20. In the MGTF, an Astrobee rode an active motion platform built from a 3-DOF active gimbal mounted on a 3-DOF active gantry. The gantry and gimbal allowed full 6-DOF motion testing Astrobee's localization system.

Because the Astrobee was rigidly mounted in the gimbal, its propulsion system was disabled. However, the Astrobee still ran most of its normal mobility software stack. The force allocator module was disabled, and the linear and angular acceleration outputs from the controller module were sent to an Astrobee physics simulation instead. The simulation calculated the resulting position and velocity to target for gantry and gimbal motion, and sent those targets to the MGTF's controller. Alternatively, an MGTF operator could simply execute pre-planned motion sequences without Astrobee involvement. Although the MGTF supported the initial phases of mapping and localization development, it unfortunately became impractical to use as a precision mobility platform as its actuators were not capable of fine control required to test docking manuevers, for instance. It was difficult to maintain as the gimbal would require frequent calibration and repairs; it was ultimately decommissioned in 2021.

Fig. 20: Micro-Gravity Test Facility (MGTF)

Zero-gravity parabolic flights were considered, but failed the cost/benefit analysis. Past parabolic experience from the SmartSPHERES project showed that between the short duration of each zero-gee episode and the significant residual accelerations deviating from true zero-gravity, it would not provide a very useful test of Astrobee motion control [61]. Instead, Granite Lab and MGTF hardware testing were complemented with software simulation testing in 6-DOF.

IV. Astrobee Robot Software

The Astrobee Robot Software (ARS) is available open source for use by Guest Scientists and others [74]. Astrobee also relies on software that runs on the ground, which is described in §V. The key software tasks required to support Astrobee's functions include:

- Localize throughout the U.S. Orbital Segment of the ISS without extra infrastructure.
- Precisely plan and execute motions without collision.
- Provide control and monitoring from the ground with resilience to communication loss.
- Support multiple control modes, including remote teleoperation, autonomous plan execution and on-board control by Guest Science (external researchers) software.
- Autonomously dock for battery recharging and wired communications.
- Autonomously perch on handrails to conserve energy while providing pan/tilt camera functionality.
- Manage Guest Science software, hardware payloads, and user interface components.
- Simulation and data replay for testing.

The LLP and MLP run Ubuntu (currently 20.04) because of its widespread use and the availability of software packages, notably Robot Operating System (ROS). The HLP, however, runs Android (Nougat 7.1) because it is the only OS supporting some key hardware for Astrobee (the high resolution camera, video encoder and touchscreen). Android allows for the encapsulation of Guest Science software for the HLP as Android Packages (APKs), avoiding custom deployment and management methods. Astrobee does not require any real-time kernel extensions, since the control loop runs at a relatively low frequency of 62.5Hz. Actual servo and motor control is achieved with dedicated microcontrollers.

Building on ROS middleware [84], the robot software consists of $\sim$46 lightweight nodelets, grouped into $\sim$14 nodes (corresponding to UNIX processes), which are distributed across the three main processors (LLP, MLP, HLP). Nodelets grouped within the same node (e.g. camera driver and vision algorithm) communicate efficiently using zero-copy message passing. Communication between nodes of a single Astrobee uses ROS messaging (TCP) on the Ethernet bus that connects the main processors. Communication on unreliable links outside the Astrobee is done through the Robot Application Programming Interface Delegate (RAPID) using the Data Distribution Service (DDS) (details in Section §IV-A).

In this section, we discuss in detail a few particularly interesting aspects of the robot software. For more details, visit the Astrobee's open source software repository [74] or read a detailed introduction to incorporating a new package into Astrobee's software stack [2].

A. Communication

An overview of Astrobee data flow is shown in Fig. 21. Astrobees connect to the ISS Payload Local Area Network (LAN). While flying, Astrobees link to the LAN via the ISS WiFi network. When docked, Astrobees communicate to the LAN via wired Ethernet through the dock, which increases bandwidth, reduces contention with other WiFi network users, and serves as a backup link in case of a WiFi problem.

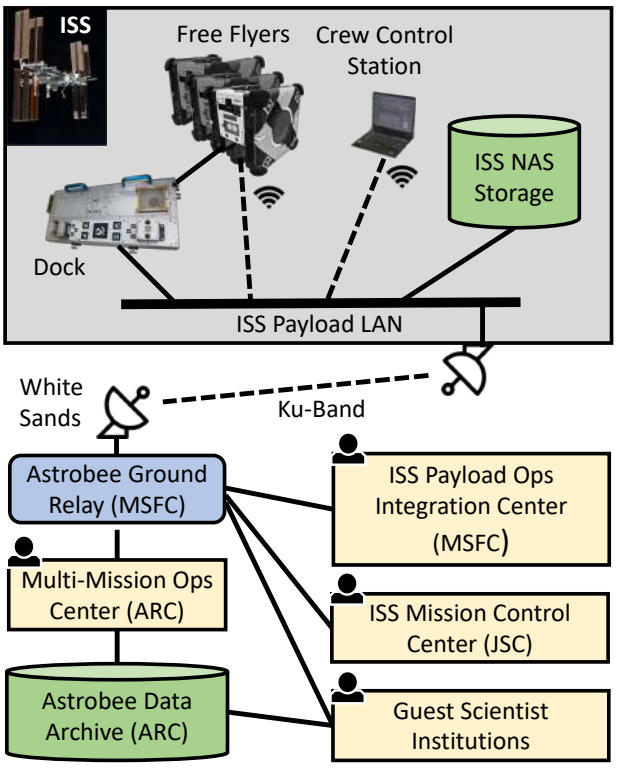


Fig. 21: Astrobee data flow

The LAN also enables Astrobees to communicate locally with each other and with the Crew Control Station installed on a crew laptop.

Astrobee exchanges high-volume batch data in both the uplink direction (occasional large software updates) and downlink direction (video recorded onboard). Beyond the storage available on the Astrobees themselves (up to 30 GB), data can be staged locally to the storage on the dock (up to 30GB) and an ISS Network Attached Storage (NAS) shared server. Software updates are uploaded once to the dock, then propagated to multiple Astrobees.

The Astrobee ground relay receives a single Ku-band downlink telemetry stream from Astrobees on the ISS and can replicate the telemetry for monitoring by multiple operators at different operating centers. Video on the space-to-ground link is H.264 compressed and streamed to the ground via the Real-Time Streaming Protocol. Commands and telemetry use NASA's RAPID message conventions [97] and are delivered by RTI DDS middleware, which offers tunable quality-of-service to enable reliable operation over high-latency and degraded networks [42]. RAPID/DDS has been used to control robots on Earth from the ISS [18] as well as robots on the ISS from Earth [43].

Because the WiFi and Ku-band networks have frequent dropouts, Astrobees log all streaming data onboard the robot so that a complete copy can be downloaded later. After a Guest Science experiment, Guest Scientists can access their data through a data archive web server at ARC.

B. Software Updates and Configuration Management

Astrobee software and firmware is periodically updated on-orbit without astronaut support or external equipment. All the microcontrollers run a custom bootloader that allows firmware to be safely updated from the host processor. The HLP processor running Android is updated via network from the MLP using Android *fastboot*. The MLP and LLP processors have a more complex update mechanism using a rescue partition that is updated by a wired network connection to the dock. The Linux image creation for the ARM processors is based on a set of custom tools that guarantee consistency between the development environment and the images running on the embedded processors. For reliability purposes, the MLP and LLP use a read-only file-system with robot-specific customizations, like IP addresses and hardware serial numbers, applied using an overlay partition.

Software on the MLP and LLP is managed via Debian packages. All Astrobee software is packaged into Debian packages which can easily be removed and upgraded. Configuration files are kept in a separate Debian package, so that they can be updated without changing other software.

Finally, most of the software is stored in file systems that are mounted read-only during operations, making them less susceptible to corruption. Ephemeral overlay partitions enable these files to be modified on the fly when needed, but any changes will revert automatically on reboot.

C. Command Interfaces

Astrobee offers the following command interfaces that can be mixed during a single session:

Teleoperation from a control station located at either ground control or on-board the ISS.
Autonomous execution either via a sequence of planned actions in a plan file, or commanded via software on the robot.
Guest Scientist control of the robot's behavior by an Android application running on the HLP.

To support the multiple control interfaces we use a common command dictionary. This dictionary is processed by translators that create a set of DDS commands (using the RAPID framework) to support external operator control, and a Java Application Programming Interface, called the Astrobee API, for Guest Scientists. Fig. 22 illustrates how commands and telemetry flow to and from ARS.

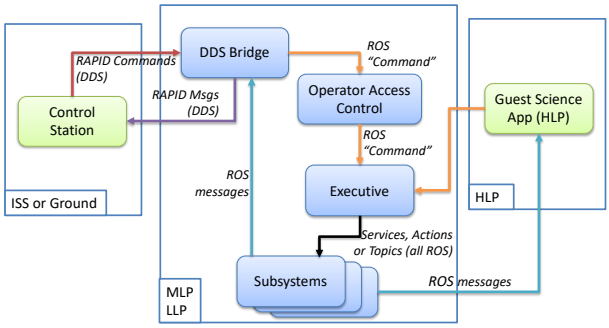


Fig. 22: Unified Command and Telemetry within Astrobee core system (LLP and MLP) and external applications (Control Station and/or HLP).

D. Execution Management

The **executive** filters incoming commands as a function of their source and the current operating mode (teleoperation, plan execution or Guest Science). DDS commands issued from the ground are transformed into ROS commands by the **DDS bridge** (see Fig. 22). The DDS bridge also subscribes to useful ROS messages for real time monitoring, and transmits them to the ground at a controllable rate as DDS messages.

ARS uses a distributed fault management framework. Each subsystem is therefore responsible for detecting and communicating its local faults to the system monitor. The **system monitor** uses information encoded in heartbeats from other nodes to aggregate fault information and responds according to a set of actions that are defined in the **fault table**. Some example responses include Stop and Propulsion Idle, as discussed in IV-E.

ARS also supports guest science Android Java applications ("apps") developed by external users to run on the Astrobee HLP. The ARS Guest Science manager running on the HLP enables ground operators to manage starting and stopping apps as well as route app-specific commands and telemetry for real-time experiment management. Guest Scientist app developers can leverage the ARS Guest Science library when building their apps to get easy onboard access to the Astrobee API and to Astrobee ROS telemetry.

E. Mobility

The **mobility** subsystem is responsible for planning, executing and monitoring all free-flyer motion. It is the highest level of Astrobee's navigation architecture, as shown in Fig. 23. Trajectories can be synthesized on the ground using the control station, or calculated on-board using trajectory **planners**. Either way, the command is rejected if the trajectory approaches too close to pre-defined keepout zones. The trajectory is encoded as a second-order polynomial spline; spline subintervals are called "set points". The mobility module passes set points to the controller to execute, and monitors the execution for any errors.

The system uses a plug-in architecture to switch between path planners. The default trapezoidal path planner generates straight-line trajectories with trapezoidal angular and linear velocity ramps. One may also add an optional face-forward condition, which ensures the robot always faces forward as it moves. This condition ensures that the depth camera faces in the direction of motion, which enables the robot to "see" upcoming collisions over short horizons. A second Quadratic Programming (QP) planner creates smooth, optimal trajectories around obstacles [101] and can be selected at runtime without affecting the rest of the system. The QP planner is appropriate for elaborate moves in free space, but not necessarily for

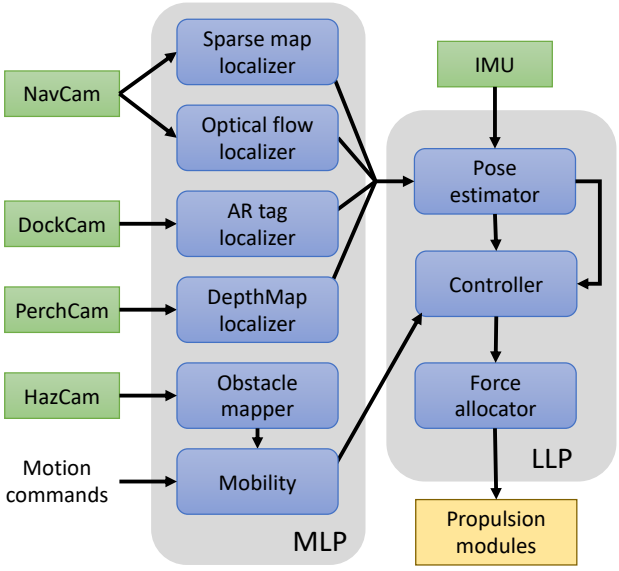


Fig. 23: Navigation architecture

simple linear moves that are required for docking or perching. Guest Scientists have also designed their own planners, such as RATTLE, an online information-aware motion planner [4].

The mobility subsystem is also responsible for detecting obstacles. The **mapper** uses OctoMap [50] to build an octree-based occupancy map by fusing the HazCam depth camera with pose estimates. It then validates trajectories against this map and predefined keep-in and keep-out zones for each mission scenario. During ISS operations, however, it is not used due not only to computational cost but the slow speed of Astrobee vs. astronauts, and operational checks done to make sure the paths are clear before motion occurs.

When an Astrobee receives a "Stop" command, its mobility mode first transitions to "Stopping", during which it suspends position control and attempts to zero its 6-axis velocity. When the velocity drops below a small threshold value ($0.02m/s^2$), it automatically transitions to "Stopped" mode, which latches its current pose for station keeping. When Astrobee receives a "Propulsion Idle" command, its mobility mode switches to "Idle", which closes all nozzles to generate zero thrust. (If an idle Astrobee is not restrained, it will drift in zero gravity.)

The mobility subsystem can automatically trigger a Stop command to Astrobee. Conditions that trigger a stop include:

- the current motion command approaches too close to a mapped obstacle. The mapper relies on hazcam measurements that aggregates collision locations as soon as it detects them. If it identifies a collision point within the robot's trajectory, it will command the robot to stop.
- the Astrobee's motion diverges too far from its desire trajectory. The divergence is a parameter that can be adjusted according to the operating environment and modes of operation e.g. nominal, quiet, lomo, or difficult modes. For instance, if the robot is in the ISS, the end position may be at 10 cm and the current position at 25 cm; if the robot is operating at the Granite Laboratory, the end position may be at 10 cm and the current position at 20 cm. Astrobee's motion can diverge from its trajectory if an astronaut manually moves the Astrobee while it is in flight.

In other cases, the mobility system will idle Astrobee's propulsion and allow it to drift. For instance, when the pose estimator diverges (has a high covariance), Astrobee idles propulsion because it lacks reliable position and velocity information to use for control. Also, if Astrobee detects an over-speed condition, it idles propulsion to avoid the chance of making the situation worse.

F. Docking and Perching

Specialized behaviors control both docking and perching. Both behaviors are similar: we first describe the docking behavior, and then explain the changes for perching. First, the robot is positioned in a location where the AR tags on the dock are visible to the DockCam. Localization is switched from localizing based on map features to using features from the AR tags. Assuming the relevant features are detected, the robot moves to an approach point a fixed relative distance from the target, switches to a docking mobility mode with a higher fan speed for tighter control approaching the small target, and moves with the mobility system to a point slightly beyond where contact with the dock would occur. Once a connection with the docking mechanism is detected electronically, propulsion is disabled. If connection with the docking mechanism is not detected, the robot returns to the approach position.

Perching differs in that handrail detections are done with the PerchCam (depth camera) using the algorithm presented in [57] rather than AR tag features. At the approach point, the arm is extended and opened. Once the approach motion is completed, the gripper is closed. However, the gripper does not have appropriate feedback to detect if it is grasping the handrail. So the robot then attempts to back up. If it succeeds, grasping has failed, and it tries again up to three times. If it fails to back up, it is assumed grasping has succeeded, and the propulsion is turned off. Docking very rarely requires a retry; however, retries are common when perching due to greater sensing challenges.

G. Control

Astrobee's motion control subsystem runs on the LLP. By limiting the number of processes running on the LLP, closed-loop control has a jitter well below the accepted tolerance. The control subsystem is divided into two components:

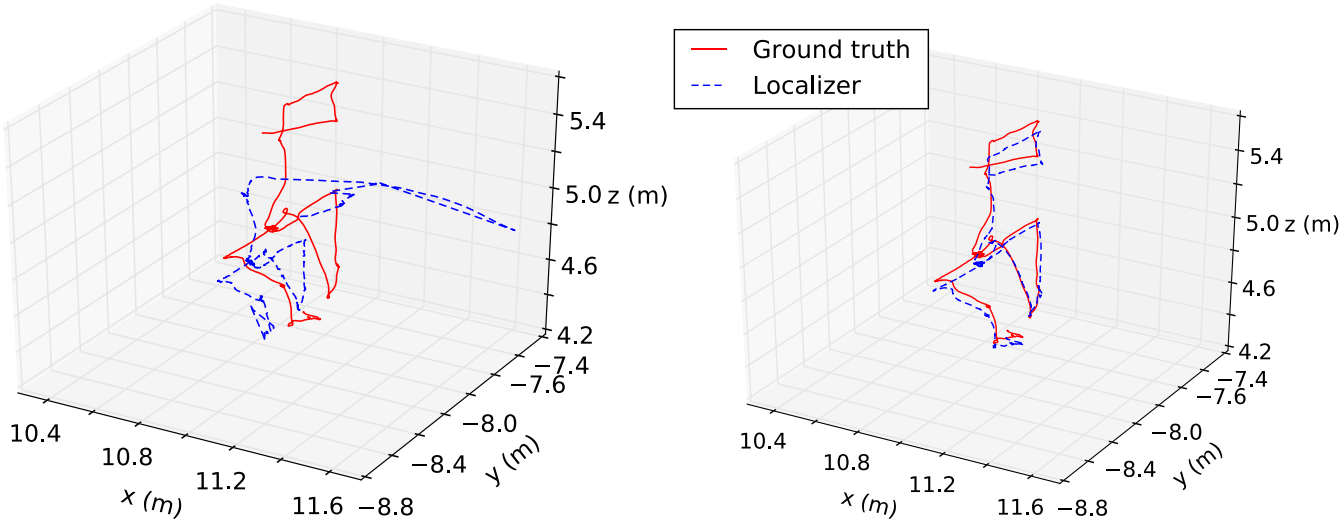
1) A Control (CTL) loop that calculates a desired force/torque that drives the estimated state (from localization) towards the goal state. The controller implements PID loops for position and attitude control. The attitude control portion includes feed forward linearization.
2) A Force Allocation Module that translates forces and torques to propulsion nozzle commands as discussed in §III-A3.

This decomposition has allowed users to replace a single component (typically CTL) with their own algorithm, while still benefiting from the other components [34].

H. AstroLoc Localizer

Astrobee used AstroLoc [90] for graph-based localization, as described in the following sections. The localizer has since been upgraded to AstroLoc2, which is discussed in more detail in [91].

The Astrobee localizer [25] preceeding AstroLoc was based on the Multi-State Constraint Kalman Filter (MSCKF) [69] and had successfully completed many tasks on the ISS since its introduction. However, it often suffered from large localization drift or lost robot events that forced the robot to be reset or re-docked during an activity. Due to occlusions from objects such as cargo bags, lighting changes, and other environmental discontinuities on the ISS, map landmarks are also not always available, so Astrobee relies heavily on visual-inertial odometry (VIO) for navigation. Additionally, free-flying in microgravity prevents the use of a gravity vector and requires highly accurate localization and velocity estimation to control and station keep the robot.



(a) Previous EKF-based Localizer **(b)** AstroLoc

Fig. 24: AstroLoc reduces drift and more accurately tracks Astrobee's position compared to the previous EKF-based localizer.

To improve upon the performance of the MSCKF-based localizer, we use graph-based optimization, which performs relinearization during each iteration compared to the single linearization point used by EKF-based systems.

Existing graph-based approaches rely on sliding window bundle adjustment for VIO [38] [96] [59] or a parallel mapping process [71] [83] for SLAM that are too expensive to run on Astrobee, whose compute platform [88] is roughly 10 times slower than an Intel i9-9980HK 2.4 GHz CPU and is running a suite of other processes, leaving only a single core for the graph-based localizer.

We therefore use a monocular visual-inertial graph-based localization system for the Astrobee robots.

The localizer includes:

- A novel localization system for a free-flying robot that intelligently incorporates visual odometry measurements, performs sliding window marginalization, and limits factors and optimization times to perform graph-based optimization in a resource constrained environment.
- Methods for handling visual odometry and localization factor cheirality issues to improve robustness.
- Approaches to separately evaluate the accuracy of tracked velocity and IMU bias estimates.
- A modular software framework for state estimation.

The AstroLoc system is tuned for the Astrobee robots but many of the parameters in §IV-H3 can be adjusted for other computationally limited platforms.

We have released the code[2] to the public, as well as an ISS localization dataset, including comparisons to other state of the art localization algorithms [51].

1) Related Localizers: **EKF-based Approaches** The MSCKF [69] uses marginalized visual odometry errors along with map-based features in an augmented-state EKF for localization, while ROVIO [11] performs monocular VIO using an EKF with a direct photometric error.

Graph-based VIO Approaches OKVIS [59] performs monocular or stereo VIO using BRISK features with a graph-based optimizer. SVO [45] and DSO [38] are both direct graph-based visual odometry methods with support for monocular and stereo systems and VI-DSO [100] adds IMU support. Each of these methods require costly sliding window bundle adjustment, and direct methods often rely on camera frame rates that are much faster than the Astrobee rate of 15 Hz.

Smart factors [20] marginalize out 3D feature points to more efficiently perform graph-based visual odometry, but they correlate state parameters and require an expensive SVD process, as described further in §IV-H3.

Graph-based SLAM Approaches PTAM [55] introduced the concept of parallel tracking and mapping that many modern SLAM methods rely on. ORB-SLAM [71] uses ORB features along with the DBoW2 bag-of-words library [48] to perform feature-based loop-closures. Basalt [96] and Kimera [86] perform VIO and SLAM, where Basalt uses bidirectional patch tracking with LSSD costs and ORB feature loop-closures and Kimera adds semantics to produce a 3D mesh construction of an environment with optional semantic segmentation, but each of these require a stereo camera. VINS-Mono [83] uses a combination of direct and indirect features for monocular visual odometry and adds loop-closures for 4 DOF map optimization, but it relies on a gravity vector to reduce the dimensionality of its map optimization and it requires bundle adjustment for 3D feature point estimation.

Free-flyer Localizers The SPHERES [76] satellites required ultrasonic beacons for localization while the JAXA Int-Ball [65] needs stereoscopic markers placed throughout the ISS. The previous localizer for Astrobee [25] used map-based image features but relied on the MSCKF for sensor fusion.

Other Aerospace Localizers

VISINAV [70] performs localization for planetary entry, descent, and landing using natural terrain and an MSCKF. Relative pose estimation is calculated for autonomous orbital rendezvous and proximity operations in [52] but requires an EKF and either fiducials or an existing 3D CAD model for tracked objects.

To take advantage of graph-based optimization while also using our prebuilt image feature map, we designed a localization system for Astrobee's limited compute platform that we present in the following sections.

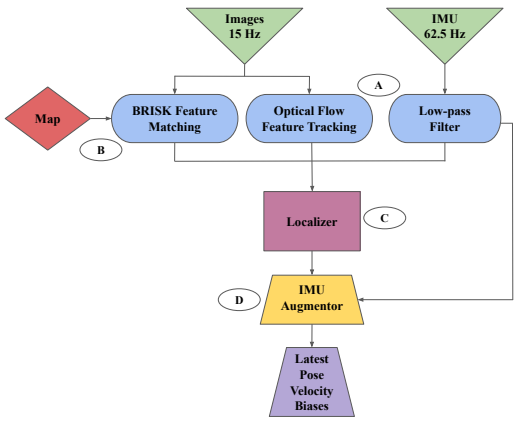


Fig. 25: Localization pipeline showing map-dependent and independent data used by the localizer, along with the IMU Augmentor, which extrapolates the navigation state produced by the localizer with the latest IMU measurements.

2) AstroLoc System Overview: The AstroLoc localization pipeline processes map-dependent and map-independent data to generate a navigation state S_i consisting of the pose, velocity, and IMU biases of the robot. Fig. 25 shows the pipeline in more detail and Table V summarizes the inputs to localization.

[2]https://github.com/nasa/astrobee

Source	Approach	Used for	Sensor	Rate (Hz)
Sparse map	Recognize features from pre-built sparse map	General navigation	NavCam	≈2
Optical flow	Track feature motion from image to image	General navigation	NavCam	≈15
AR tag	Recognize unique IDs of AR tag fiducials and extract corners	Docking	DockCam	≈15
DepthMap	Extract handrail and wall geometry from 3D point cloud	Perching	PerchCam	≈5
IMU	Measure 6-axis linear acceleration and angular velocity	All motion	IMU	62.5

TABLE V: Localization sources

The pipeline processes images and IMU measurements at 15 Hz and 62.5 Hz respectively as shown in Part A of Fig. 25. Vibrational noise from the impeller fan is removed by passing the IMU measurements through low-pass and notch filters. Optical flow feature tracks are generated using the pyramidal Lucas-Kanade algorithm [17], where forward and backward passes are conducted on a sequence of images to limit outliers.

The pipeline uses mapped 3D BRISK features stored in a DBoW2 library [48] and sends measurements and their associated map features from each image to the localizer.

The mapping procedure is described in more detail in [25].

Next, the graph-based localizer in Part C processes the measurement inputs and outputs a navigation state S_i. Its design is discussed further in §IV-H3.

Since the localizer is used with a controller that expects frequent navigation state updates at a rate similar to that of the IMU, S_i is subsequently passed to the IMU Augmentor in Part D. This is also discussed in more detail in §IV-H3.

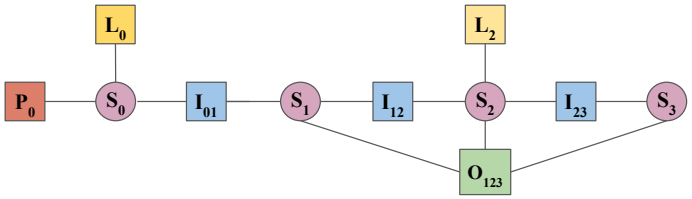


Fig. 26: Factor graph representation for the AstroLoc localizer. Navigation states are represented by nodes S_i. IMU preintegration factors I_{ij} are used to connect neighboring navigation states. Factors for map-based landmarks are represented by L_i, and visual odometry smart factors by O_i. Priors for the oldest navigation state are shown using P_i.

3) Efficient Localization: AstroLoc performs graph-based nonlinear optimization with the factor graph representation shown in Fig. 26. It adds IMU measurements as preintegration factors [44] to avoid reintegrating IMU data during each optimization iteration, and uses smart factors [20] to incorporate optical-flow feature tracks as visual odometry constraints and remove the costly estimation of 3D features that most graph-based visual odometry and SLAM approaches require. AstroLoc inserts map-based feature measurements using projection factors with the error function in (5) where the map feature is not included in the optimization problem to avoid increasing graph solve times. Optimization is performed using the GTSAM library [32].

Limiting Cost of Visual Odometry Smart Factors Visual odometry smart factors are the most expensive component of the AstroLoc graph optimization process. The factor error is formed by projecting a triangulated feature point f into a set of images containing feature point measurements per (5).

$$\mathbf{p}_i = \mathbf{u}_i - \pi({}^{\mathbf{C}}_{\mathbf{B}}\mathbf{T}\,{}^{\mathbf{B}}_{\mathbf{W}}\mathbf{T}\mathbf{f}) \tag{5}$$

The projection function π for a camera model represents the mapping $\pi\colon \mathbb{R}^3 \mapsto \mathbb{R}^2$ where $\mathbb{R}^2$ is image space, transform ${}^{\mathbf{C}}_{\mathbf{B}}\mathbf{T}$ is the extrinsic calibration of the camera, transform ${}^{\mathbf{B}}_{\mathbf{W}}\mathbf{T}$ is the inverse of the robot body pose in the the world frame, and $\mathbf{u}_i$ is the feature measurement in image i. The smart factor forms the error vector $\mathbf{e}$ by stacking the errors per (5) for each feature measurement in the measurement set. The error function is the same as a bundle-adjustment approach, but unlike in bundle-adjustment procedures, the 3D feature points are not included in the optimization problem. These are removed with a marginalization procedure mirroring that of [69] that requires the singular value decomposition of the feature Jacobian matrix.

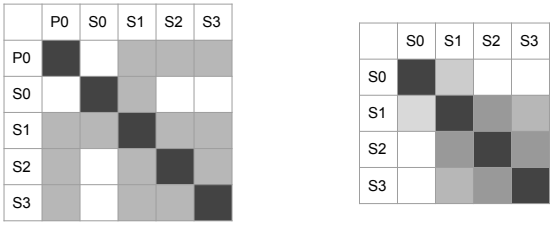
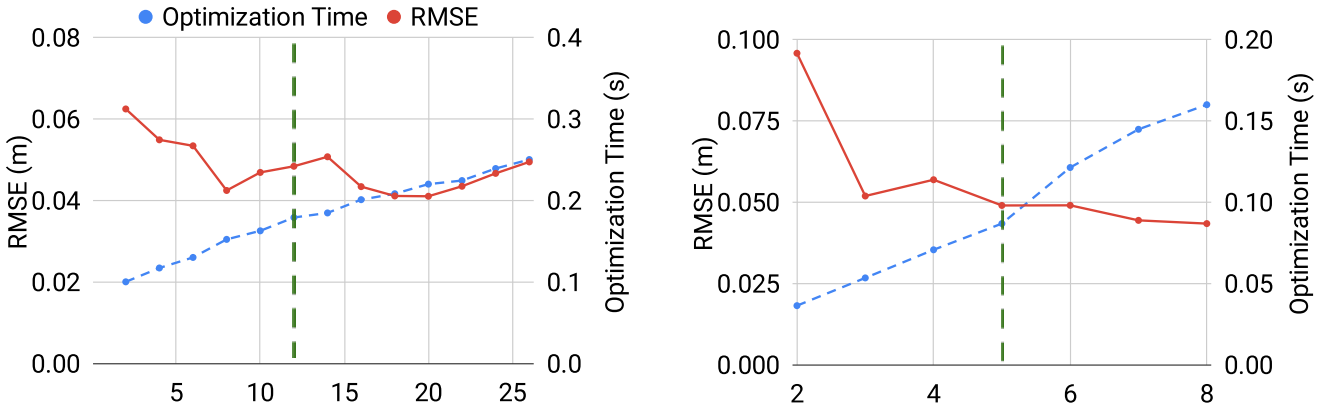


Fig. 27: Example Hessian before and after smart factor marginalization, for a graph containing feature track measurements at states S_1, S_2, and S_3 and the feature point P_0.

While the smart factor marginalization procedure reduces the dimensionality of the factor as shown in Fig. 27, many off-diagonal components of the Hessian may still be present that increase the optimization time for the graph, indicated by the non-empty blocks of the Hessian matrix in the same figure. Additionally, each smart factor in the graph performs the singular value decomposition marginalization during each optimization iteration, greatly reducing the speed of the localization procedure.



(a) Maximum number of smart factors

(b) Max measurements per smart factor

Fig. 28: Trade-offs in accuracy versus computation time for the number of smart factors included in the graph and the number of measurements added per smart factor. AstroLoc uses a maximum of 12 smart factors and five measurements per factor.

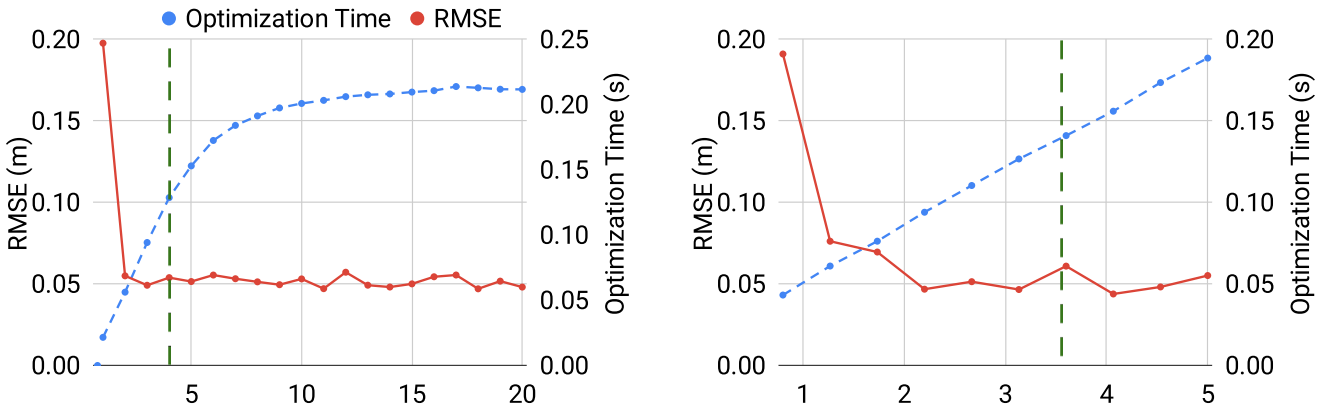
AstroLoc therefore limits both the overall number of smart factors included in the graph and the measurements included in each smart factor to reduce their computational cost. The impact of limiting the total number of smart factors on the graph solve time is shown in Fig. 28a, where the data is generated using a parameter sweep with the evaluation procedure described in §IV-H5.

AstroLoc also performs measurement selection to reduce the number of off-diagonal terms in the graph's Hessian matrix. Each feature track includes measurements from the same subset of timestamps, which are evenly spaced over the duration of the graph. This limits the correlation between navigation state nodes in the graph and increases the sparsity of the optimization problem, while also increasing the portion of the graph constrained by visual odometry factors. Fig. 28b shows the tradeoff in optimization time and accuracy versus the number of measurements included per smart factor. In practice, AstroLoc uses five measurements per smart factor and includes a maximum of 12 smart factors in its factor graph.

Sliding Window Marginalization While AstroLoc supports adding marginalization factors when sliding the graph window using the linearization of each removed factor [59], it uses a more efficient method instead that inserts priors for the oldest state parameter nodes remaining in the graph. Marginalization with linearized factors correlates state parameters that shared a factor with marginalized parameters and introduces many off-diagonal elements in the resulting Hessian matrix, increasing optimization times. Additionally, when using smart factors, adding marginalization factors becomes difficult as only a subset of each smart factor's measurements are removed when the window is slid rather than the entire smart factor. AstroLoc therefore adds priors to the oldest remaining state parameters in the graph using their current estimated values and the covariances generated by the most recent optimization iteration's inverted Hessian matrix. This limits the density of the optimization problem and yields faster solve times.

Limiting Optimization Runtime AstroLoc's convergence criteria include both a threshold for the minimum change in relative error while optimizing, and a limit on the maximum number of optimization iterations performed. As Fig. 29a shows, error reduction quickly saturates while the iteration threshold and resulting computation time continue to increase. To prevent outlier optimization times that use many iterations, AstroLoc limits the maximum number of optimization iterations to four.

In addition to limiting the total number of smart factors, AstroLoc limits the total factor count for map-based localization



(a) Maximum optimization iterations

(b) Sliding window duration

Fig. 29: Trade-offs in accuracy versus computation time for the maximum optimization iterations and sliding window duration of the graph. AstroLoc uses a maximum of four optimization iterations and a sliding window duration of 3.5 seconds.

factors to around 40. The localization factors are much more efficient to compute, as they are image projection factors that only depend on one state parameter node. AstroLoc limits the sliding window graph duration to 3.5 seconds, which provides some extra buffer time for outlier map-based feature measurement delays. The trade-off in accuracy versus runtime for graph duration is depicted in Fig. 29b. These constraints help limit the maximum runtime for AstroLoc and enable it to provide localization updates at a frequency of around 5 Hz. The localization updates are then fed into the IMU Augmentor described in the following section, to provide even higher frequency updates to the controller.

High Frequency Updates using the IMU Augmentor The IMU Augmentor delivers high frequency 62.5Hz localization updates that the controller requires by extrapolating the latest localization estimates provided by the localizer with the latest IMU data.

It performs extrapolation by maintaining a queue of IMU measurements and integrating them on top of the most recent navigation state, using its IMU bias estimates. Without the IMU Augmentor, feedback to the controller occurs at 5Hz instead of 62.5Hz, causing Astrobee to over- and undershoot its desired poses and velocities while path planning.

4) Robust Localization: To prevent lost robot and high drift events during localization, AstroLoc uses a robust loss function as well as introducing several techniques.

Robust Cost using Huber Loss To avoid drift due to outlier measurements, AstroLoc feeds each factor through a Huber loss function per (6).

$$h(n) = \begin{cases} n & \text{if } n < k \\ n\sqrt{k/n} & \text{otherwise} \end{cases} \tag{6}$$

Here $n = \mathbf{e}^T\mathbf{e}$ is the error norm for a factor. AstroLoc uses the standard threshold $k = 1.345$. The Huber loss assumes that large factor errors are often correlated with outlier data.

Preventing Cheirality Errors for Map-based Projection Factors using Pose Factor Fallback AstroLoc inserts map association pairs p_i, consisting of an image measurement $\mathbf{u}_i$ and its associated 3D map features $\mathbf{f}_i$, using the error function in (5). If every feature point in the set of pairs P for an image suffers a cheirality error, the localizer instead adds a pose prior factor so the measurements can still be included in the optimization problem. The pose ${}^{\mathbf{W}}_{\mathbf{C_e}}\mathbf{T}$, where $\mathbf{C_e}$ is the estimated camera frame, is found using the perspective-three-point algorithm [49] with a RANSAC selection procedure described in [25]. Four randomly selected pairs from P are used to estimate ${}^{\mathbf{W}}_{\mathbf{C_e}}\mathbf{T}$ and the rest of the associations are checked for consistency. This is repeated for several iterations and the pose with the most inliers is used for ${}^{\mathbf{W}}_{\mathbf{C_e}}\mathbf{T}$. The error function for the factor is shown in (7), where ${}^{\mathbf{W}}_{\mathbf{B}}\mathbf{T}$ is the pose estimate in the graph at the image timestamp and ${}^{\mathbf{C}}_{\mathbf{B}}\mathbf{T}$ is the transform from the body to camera frame.

$$\mathbf{e} = \log({}^{\mathbf{W}}_{\mathbf{B}}\mathbf{T}^{-1}\,{}^{\mathbf{W}}_{\mathbf{C_e}}\mathbf{T}\,{}^{\mathbf{C}}_{\mathbf{B}}\mathbf{T}) \tag{7}$$

This fallback procedure helps avoid localization drift, as more map-based measurements can be included in the graph even with poor current position and orientation estimates that often lead to cheirality errors.

Preventing Cheirality Issues in Visual Odometry Smart Factors using Measurement Pruning

In addition to providing a fallback for cheirality issues for map-based localization factors, AstroLoc also introduces a measurement pruning procedure to help eliminate cheirality issues for visual odometry smart factors. Smart factors are more susceptible to cheirality issues, since if one measurement in the smart factor suffers a cheirality issue, the entire factor is ignored. Recognizing that cheirality issues are often the result of a faulty pose estimate or sequence of faulty pose estimates that have been recently added to the localizer, AstroLoc tests each smart factor before an optimization cycle. It fixes smart

factors with cheirality errors using the following approach. It first removes the most recent feature measurements and checks the resulting smart factor for errors. The measurements are checked individually, and if the cheirality error is removed by discarding a measurement the factor is added without the faulty measurement. Otherwise, measurement sequences are checked, starting again with the most recent measurements and additionally removing previous measurements until the error is removed. If the cheirality error still remains the factor is discarded.

Sanity Checking The resulting navigation state from the localizer is checked using a covariance-based sanity checker and pose history sanity checker. If covariances for S_i exceed a set threshold, the localizer is reset. Additionally, if a predetermined number of poses from the localizer drift too far from pose estimates derived using map-based associations as described in §IV-H4, the localizer is also reset.

5) AstroLoc Experiments: We evaluate AstroLoc on a dataset of 12 ISS activities and compare results with the previous Astrobee localizer and a base GTSAM localizer.

Ground truth is created using our mapping pipeline described in [25].

AstroLoc runs on Astrobee's Inforce 6501 Micro SoM featuring a Qualcom Snapdragon SoC mobile processor and utilizes a single processing core, which as mentioned in IV-H runs $\sim$10 times slower than an Intel i9-9980HK 2.4 GHz CPU.

The reliance on a gravity vector and lack of a prebuilt feature-based map prevent the comparison with many of the methods from §IV-H1, in addition to their increased computational cost that prohibits their use on Astrobee's limited compute platform. We therefore compare AstroLoc to both the previous Astrobee localizer [25] and a base GTSAM implementation. The base GTSAM localizer provides a graph-based localization reference that still incorporates the IMU preintegration and visual odometry smart factors used by AstroLoc, but does not include the efficiency improvements discussed in §IV-H3 or the robust techniques of §IV-H4.

We evaluate the localizer in two modes: (1) VIO mode, using only VIO information, with map-based measurements suppressed. (2) Localization mode, using both VIO and map-based measurements. VIO mode is useful for simulating the localizer behavior when prior map features are not recognized, as often occurs due to environment changes.

As an additional performance metric, we measure the number of lost events that occur on each dataset, defined as when the robot has drifted by more than 1.5 meters. Lost events can require crew intervention during activities, and avoiding them is therefore a high priority for AstroLoc.

To more precisely track performance of velocity and IMU bias estimates, we integrate graph estimates for each of these and compare the results with ground truth data. Integrating velocities utilizes the time difference between successive velocity estimates to add position updates to the starting position of the robot per (8).

$$\mathbf{p} = \mathbf{p}_0 + \sum_{i=1}^{N} \Delta T \mathbf{v}_i \tag{8}$$

Here ΔT is the time difference between successive velocity estimates $\mathbf{v}_i$ and $\mathbf{v}_{i-1}$, and $\mathbf{p}_0$ is the initial position of the robot. Accurate velocity estimation is critical for the controller to station keep and execute precise maneuvers in microgravity.

To evaluate the IMU biases estimated by the localizer, we integrate each IMU measurement in a recording using the latest IMU bias estimate occurring before or at the measurement's timestamp. Then we generate a set of relative integrated IMU pose increments ${}^{i\text{-}1}_{i}\mathbf{T}$, which are added to the initial robot pose per (9), similar to the velocity integration procedure.

$${}^{w}_{i}\mathbf{T} = {}^{w}_{i\text{-}1}\mathbf{T}\,{}^{i\text{-}1}_{i}\mathbf{T} \tag{9}$$

Here ${}^{w}_{0}\mathbf{T}$ is the initial pose of the robot in the world frame. While some degree of noise will still be present in the integration, accurately tracked biases will produce less overall drift. Tracking IMU biases is especially important as we use these to extrapolate localization estimates as described in §IV-H2.

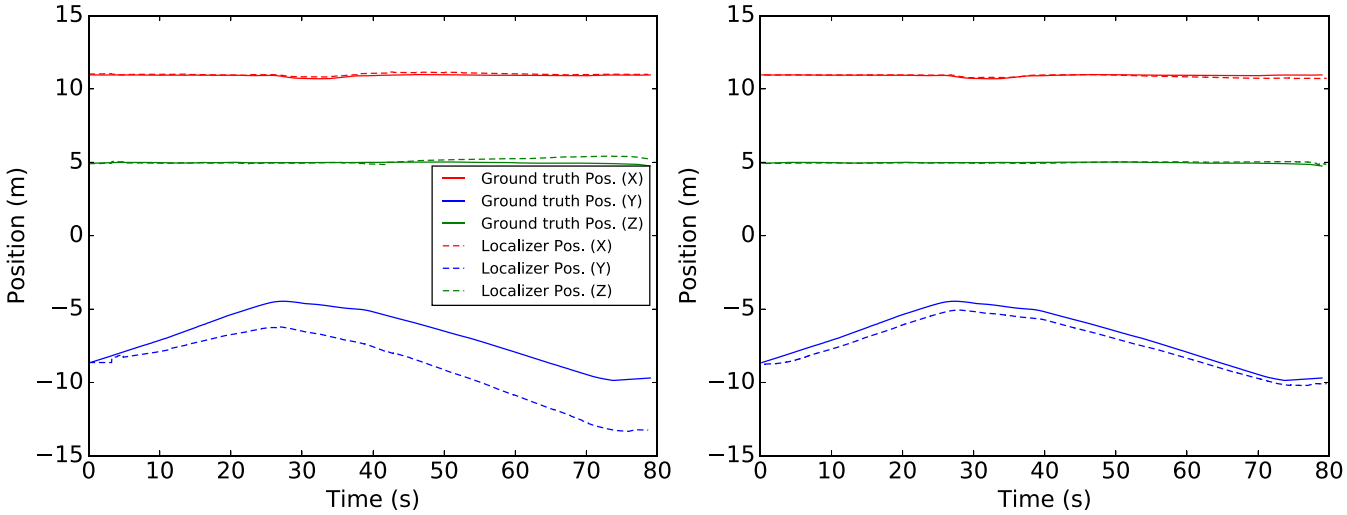
6) Results: We evaluate AstroLoc on data collected on-orbit. AstroLoc exhibits large improvements in VIO mode compared to the previous localizer and base GTSAM localizer as displayed in Table VI. RMSEs are reported for all activities except the two lost events suffered by the previous localizer and base GTSAM implementation, while the last row shows the position RMSE including these outliers.

Both the base GTSAM implementation and AstroLoc outperform the previous localizer in most VIO categories, demonstrating the improved accuracy of graph-based approaches. However, AstroLoc further improves upon the base GTSAM method in position, velocity, and IMU bias estimation. The base GTSAM method naively includes every visual odometry measurement, and without the AstroLoc methods for smart factor spacing and handling cheirality errors, it accrues more drift. Base GTSAM also gets lost twice, akin to the previous localizer. Additionally, it suffers larger errors in IMU bias estimation and velocity estimation, as shown in Table VI. These errors can cause the live controller to perform erroneous velocity and position corrections during an activity. Fig. 30 shows an example of the reduced drift for AstroLoc in VIO mode compared to the previous localizer.

AstroLoc greatly improves upon the base GTSAM implementation's runtimes, running roughly six times faster, as shown in Table VII. The combination of intelligent visual odometry measurement selection, factor and iteration limiting, and efficient

TABLE VI: VIO Results

RMSE	Prev Loc	BaseGT	AstroLoc
Pos. (m)	0.6212	0.5328	0.2777
Orientation (rad)	0.0730	0.0783	0.0696
Rel Pos. (m)	0.2788	0.2080	0.1440
Rel Orientation (rad)	0.0662	0.0632	0.0659
Integrated Vel. Pos. (m)	0.7953	0.9614	0.2931
Rel Integrated Vel. Pos. (m)	0.2624	0.2179	0.1470
IMU Bias Pos. (m)	83.9615	4.0982	2.1535
Rel IMU Bias Pos. (m)	12.3075	0.7425	0.4033
Lost Events	2	2	0
Pos. w/ Outliers (m)	2.4179	0.7508	0.3721



(a) Previous Localizer **(b)** AstroLoc

Fig. 30: AstroLoc reduces drift in VIO mode by utilizing an efficient graph-based localization system, while the previous localizer steadily veers off course.

sliding window marginalization, prevent excessive computation that would lead to large delays for Astrobee. The previous localizer is not included as it runs at a fixed 62.5Hz.

Table VIII displays AstroLoc's significant improvements in position, orientation, velocity, and IMU bias estimation compared to the previous localizer and base GTSAM implementation.

AstroLoc and the base GTSAM localizer suffer no lost events compared to the two experienced by the previous localizer. However, AstroLoc yields a further 80% and 15% decrease in position RMSE, with and without outlier events, compared to the base GTSAM localizer. The combination of AstroLoc's fallback approach for map-based measurement factor cheirality errors, and improved VIO performance when map-based features are sparse or unavailable, help prevent it from getting lost and accruing a large drift. Results from one activity where AstroLoc accurately tracks localization position while the previous localizer gets lost are displayed in Fig. 31.

Table IX displays the improved runtime of AstroLoc compared to the base GTSAM localizer. Similar to the VIO results, AstroLoc demonstrates significant runtime improvements. Runtimes are only slightly increased compared to VIO as localization map-based factors do not yield as significant of a performance hit as the smart factors do.

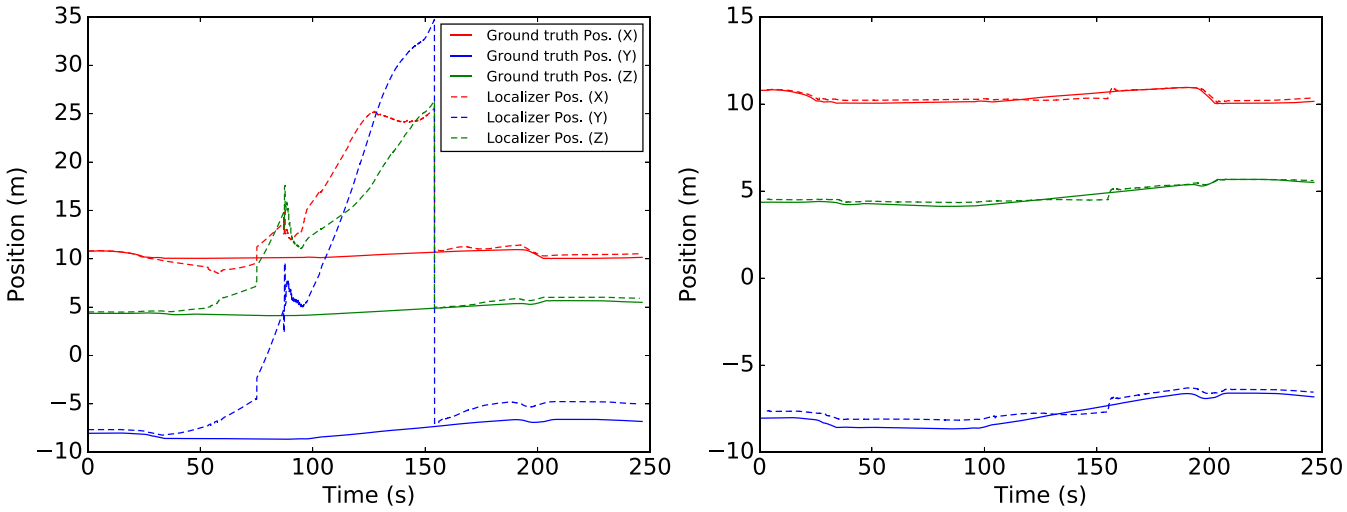
7) AstroLoc Conclusion: While the AstroLoc system is carefully tuned for the Astrobee robots, the parameters in §IV-H3 including number of smart factors, measurements per smart factor, sliding window duration, and maximum optimization

TABLE VII: VIO Runtimes

Localizer	Avg. Runtime (s)	Opt. Time (s)
BaseGT	1.0588	0.8525
AstroLoc	0.1785	0.1259
Rel Decrease	83%	85%

TABLE VIII: Localization (VIO+Map) Results

RMSE	Prev Loc	BaseGT	AstroLoc
Pos. (m)	0.0948	0.2417	0.0491
Orientation (rad)	0.0439	0.0505	0.0275
Integrated Vel. Pos. (m)	0.2681	0.3228	0.1417
Rel. Integrated Vel. Pos. (m)	0.1200	0.1534	0.0656
IMU Bias Pos. (m)	83.5386	2.4744	2.4173
Rel. IMU Bias Pos. (m)	12.2643	0.4682	0.4784
Lost Events	2	0	0
Pos. w/ Outliers (m)	1.0277	0.3105	0.2636



(a) Previous Localizer **(b)** AstroLoc

Fig. 31: Previous EKF-based localizer accruing position errors and ultimately getting lost during an activity, whereas AstroLoc accurately tracks the robot position.

iterations can be adjusted for other resource limited platforms.

From its commission, Astrobee regularly executed multiple ISS activities with over two hours of flying time. Since AstroLoc was deployed on the ISS, Astrobee's improved navigation when map-based updates are unavailable has dramatically improved operational reliability. Reduced reliance on the prior map has allowed the Astrobee team to invest much less effort in running ISS activities specifically to collect new map imagery (∼40% fewer activities and 30% fewer images per activity).

While this work primarily addresses limiting localization and VIO drift for Astrobee and has enabled longer duration and more robust experiments on the ISS, we additionally investigate using semantics for mapping and localization in [62] and in future work we wish to explore using more life-long mapping approaches to further reduce the impact of environmental changes on localization performance.

For a full, detailed analysis of Astroloc's performance, and details on Astrobee's public localization dataset, see [51]. This includes an evaluation against multiple state of the art algorithms, showing that AstroLoc has similar accuracy to top approaches at significantly reduced computational costs. AstroLoc has since been further refined and improved to create AstroLoc2, presented in [91].

V. Ground Software

The Astrobee Ground Data System (GDS) comprises the Control Station, which is a graphical application to monitor and control the Astrobees, and ground servers for relaying and archiving data generated during on-orbit activities (§IV-A).

A. Control Station Overview

There are two variants of the Control Station. The *Engineering* Control Station has the ability to issue all available commands and view all available telemetry types. This variant is used by highly trained operators and engineers on the ground. The *Crew* Control Station offers a simplified interface to a subset of the Engineering Control Station functionality, and it is designed to be used when a Guest Science experiment must be operated locally by an astronaut with minimal training.

TABLE IX: Localization (VIO+Map) Runtimes

Localizer	Avg. Runtime (s)	Opt. Time (s)
BaseGT	1.1335	0.8919
AstroLoc	0.2050	0.1389
Rel Decrease	82%	84%

Both variants use the Eclipse 4 Rich Client Platform (RCP) and are derived from the Visual Environment for Remote Virtual Exploration (VERVE) code base, which has a long history of providing 3D interfaces for space-analog testing with multiple robots [58], and was also the basis of the Smart SPHERES Workbench previously used to control the SPHERES free flyers inside the ISS [43]. The Control Station has been ported to run both on Ubuntu Linux (used internally by developers) and Windows (supported release for external users).

The Control Station has several tabs, each of which displays a different set of controls. The Crew Control Station has four tabs (Overview, Run Plan, Teleoperate, and Guest Science), and the Engineering Control Station includes an additional six tabs. Since most use cases have an operator for each Astrobee, most tabs connect to one Astrobee at a time. The exceptions are the Overview tab and the Guest Science tabs, which can monitor and control up to three Astrobees at the same time.

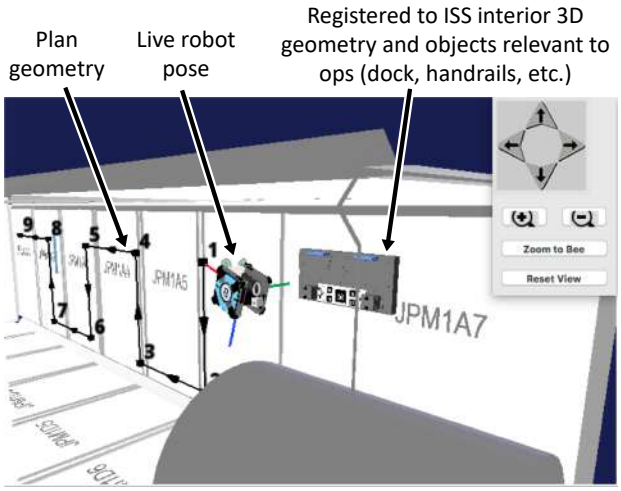


Fig. 32: Control Station 3D View

B. 3D View

Most Control Station tabs contain a 3D view, showing Astrobee's current pose within a simplified 3D model of the modules in the ISS where Astrobee is allowed to fly (Fig. 32). Astrobee keepout zones are displayed as gray boxes.

The interior of the actual ISS can be obstructed with stowage and other items that are moved around often. To avoid cluttering the model with details that quickly become outdated, the model in the Control Station shows plain interior walls of each module labeled with their standard ISS location code. For instance, "JPM1A6" denotes the Japanese Pressurized Module, aft wall, sixth bay. The only objects included in the current model are the ones particularly relevant to Astrobee: the dock, the large airlock nearby, and handrails that Astrobee can perch on. If other interior objects become relevant later, models can be added by changing configuration files. The Control Station can also be used to control Astrobees in our ground testing facilities, using alternate model configuration files.

C. Interactive Commanding

The Control Station offers three ways to specify the target pose of a movement command: the user can type in coordinates (absolute or relative to a named ISS rack), drag a preview model in the 3D view (Fig. 33), or select from a list of pre-defined positions (such as standard "dock approach points" in front of the dock berths). If the commanded position is outside the ISS or too close to a keepout zone, the Control Station prevents the operator from sending the command.

The normal conops is for an Astrobee to face forward during translation, so that any unexpected obstacles in its path can be detected by the HazCam. The Control Station supports this with a "Face Forward" feature that automatically turns Astrobee to face toward the target position before beginning translation. This feature can be disabled by the operator when needed. For example, when viewing live video from the forward-facing SciCam, the operator may want to command a small lateral move

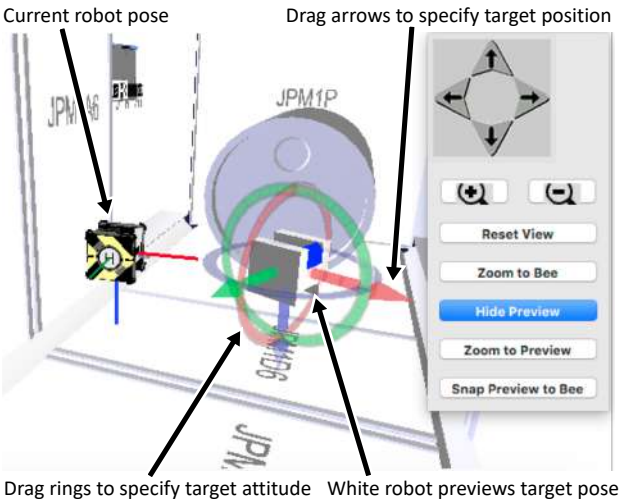


Fig. 33: The operator can specify a target pose in the 3D view, then send the motion command.

to improve the view without disrupting the video by turning the robot (assuming the operator is confident there is sufficient clear space to make the move).

Many other commands are available, including: dock/undock, perch/unperch, setting camera parameters, start/stop camera recording, and operating the flashlights and laser. (Some of these are hidden in the Crew Control Station variant, for simplicity.) The name and parameters for each command are defined once in Astrobee's command dictionary, and from there the command becomes available for interactive commanding and plan editing in the Control Station as well as interactive commanding from Guest Science Applications (GSAs) running onboard the robot.

D. Editing and Running Plans

Astrobee plans are specified in a dialect of the Exploration Plan JavaScript Object Notation (XPJSON) format [89], which has been used successfully for multiple robotic applications [18], [37], [77]. An XPJSON plan consists of a series of stations, which are locations for Astrobee to visit; commands to be executed at each station; and segments, which are moves that connect the stations. Although the plan format can encode arbitrary holonomic 6-DOF curved trajectories and Astrobee's flight software can follow them, for simplicity, the Control Station generates plans in which segments are straight lines and the Astrobee comes to a stop at every station.

The Control Station plan editor shows the current plan as 3D geometry (Fig. 32) and as a list (Fig. 34). From these views, any plan element (station, segment, or command) can be selected and its parameters edited.

The position of a station in the plan editor can be specified in the same ways as during interactive commanding, with one addition: in a feature inspired by planetary rover 2D traverse planners, the plan editor can display a horizontal reference plane in the 3D view; operators can adjust the height of the plane, then repeatedly click on the plane to create stations.

Before running a plan, the operator selects a plan file to load onto an Astrobee. A plan preview is displayed, similar to the view in the plan editor. The operator can then run or pause the plan, and skip steps while the plan is paused. While the plan is running, the 3D view and plan list view update to show the duration and completion status of each step (Fig. 34).

E. Images and Video

The Control Station can view live image frames (NavCam, DockCam) or H.264-encoded HD video (SciCam) from the selected Astrobee. The imagery viewers can be inset in the main Control Station window or opened as separate windows to view full size on another screen.

F. Guest Science

Guest Science is the only Astrobee function that requires interaction with multiple Astrobees at the same time (for example, to trigger a formation flying experiment). Accordingly, the Guest Science tab displays the status, current pose, and plan geometry for all three Astrobees. The operator can also command all the Astrobees simultaneously (Fig. 35).

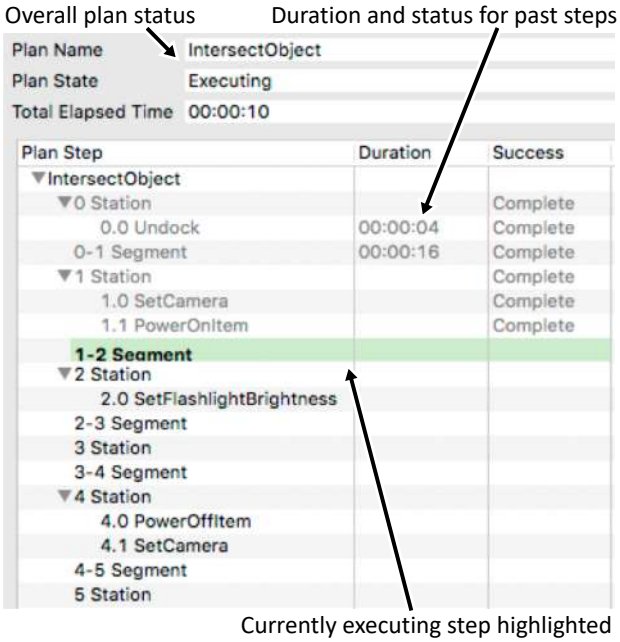


Fig. 34: Plan list view, showing a plan partway through execution

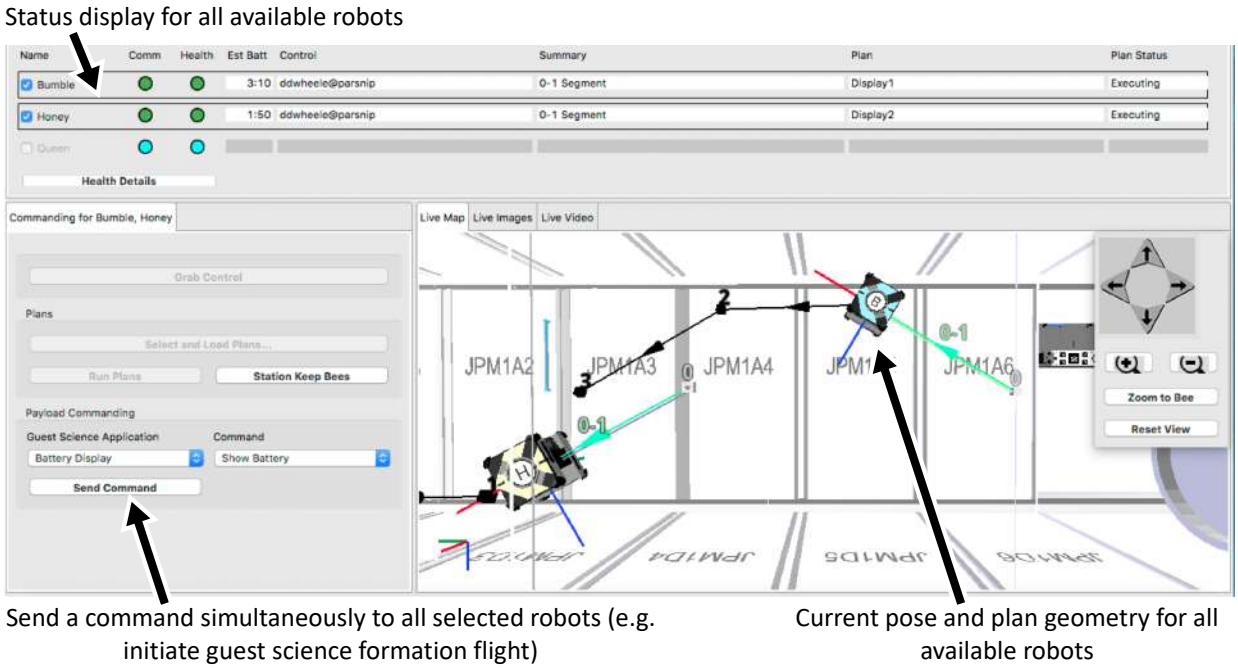


Fig. 35: The Control Station can manage Guest Science on up to three Astrobees simultaneously.

G. Configurability

Because the Crew Control Station runs on an ISS astronaut laptop, requiring months of lead time for software certification, we knew from the beginning that the software would need to be finalized early, even before the Astrobee flight hardware itself was built and tested, and of course long before we had details of novel Guest Science experiments that will continue to be developed for years into the future. It was a challenge to build an operator interface that was intuitive and clear for a novice operator, without restricting possible future use cases and without having a complete system to test on.

To support flexible Guest Science, the Control Station subscribes to custom Guest Science command definition messages published by a GSA running onboard the robot. These definitions include pre-defined default parameters for the Guest Science commands, and are used to populate the Payload Commanding tool, enabling a novice operator to send any Guest Science command at the click of a button. Because Guest Science command definitions are published by the GSA, they do not need

to be finalized until the GSA is prepared for upload.

In the Crew Control Station variant, many less-frequently-used commands have been suppressed for simplicity, but if needed for a particular activity, they can easily be re-enabled by adding them to a "Miscellaneous Commands" configuration file. The Crew Control Station reads this configuration file and automatically constructs an interface widget that allows the operator to edit the command parameters and send the command. Changes to this configuration file do not require software recertification.

The Engineering Control Station variant provides additional flexibility. Its "Configurable Commands" feature subscribes to command definition messages published by Astrobee's flight software, and enables the operator to send any of the defined commands. This feature keeps the commanding capabilities of the Engineering Control Station in sync with any new capabilities offered by the flight software, but at the cost that the Configurable Commands interface widget for each command is automatically generated based on the command parameter definitions, and cannot be optimized by an interface developer to provide a better user experience (e.g., the operator might have to manually specify coordinates rather than picking a point in the 3D view). This was considered acceptable because the Engineering Control Station is expected to be used mostly by highly trained operators: either staff employed by the Astrobee Facility, or Guest Scientists who can rehearse their specific experiments in advance. The Engineering Control Station also provides a Guest Science Telemetry subtab that automatically displays in tabular form whatever fields a GSA encodes in its custom telemetry messages.

Guest Scientists with more sophisticated interface requirements can also modify the Engineering Control Station to add more interface tabs. This capability is facilitated by the modularity features of the Eclipse RCP framework, and we have released the Control Station code base as open source in order to facilitate Guest Scientist contributions. Notably, RFID-Recon create a complex plugin that was implemented as part of GDS to command the robots and their payload during ISS Ops. Other Guest scientists who used GDS for telemetry visualization include JAXA, ROAM, and MIT.[3]

As engineers use the Control Station regularly to command Astrobee on the ISS, we expect many new insights that will drive future design iterations.

H. GDS Helper

GDS Helper is a text-based interface created to facilitate commanding and monitoring of an Astrobee robot directly from an SSH terminal session. Its batch mode enables complex command scripting while retaining some interactive capabilities. The GDS Helper complements the Astrobee Control Station by allowing a secondary operator to assist with an ISS activity, by quickly sending short commands to the Astrobee and by reading out data to the primary operator, such as the number of marked landmarks recognized by the localization system.

Figure 36 illustrates the layout and different components of the GDS Helper.

The command line interface also helps with repeatability. With increasing ops experience, we improved the reliability of Astrobee activities by specifying repeatable ground procedures for operators to follow, especially during activity setup and teardown. It is more convenient to document these procedures as shell commands (for operator copy/paste) than to specify detailed actions to take in a graphical interface.

A further improvement is that we can easily incorporate these shell commands into reusable higher-level scripts that encode things like success verification and retry logic. For future robot software development to get the benefits of both approaches, we recommend starting with a fully featured robot command API binding for a flexible scripting language, and building graphical interfaces that invoke these scripts in a customizable way. This strategy would allow easy iterative procedure improvement during activity preparation as well as maximum flexibility for operators to improvise on the fly, reordering procedure building blocks during the unpredictable ops of a science experiment.

VI. Six Years of On-Orbit Operations

Astrobee has reached six years of continuous operations since the commissioning of the Dock in early 2019. The Dock was followed by Astrobee units named Bumble and Honey, the perching arms, and finally the Astrobee unit Queen. Table X provides a high-level overview of the operations of the Astrobee elements since their commissioning.

Including commissioning, Dock repair, and general commanding sessions, the Astrobee Facility has completed over 165 on-orbit real-time operations on the ISS to date. Since the completion of commissioning, the Astrobee Facility has completed over 90 Guest Science Test Sessions. In total, these real-time operations involved over 1200 hours of console support of which approximately 220 hours involved crew. Therefore, nearly 80% of console time did not require crew. Of these crew hours, 25 unique ISS crew members have been trained and participated in Astrobee operations. After its commissioning on October 30, 2019, Honey had to be returned to NASA Ames Research Center to repair a corrupted SD-card. It was returned to the ISS on August 30, 2023; the delay was due to the on-site working limitations in place during the COVID-19 pandemic. Astrobee has supported over 16 unique Guest Scientists from academia, industry, and NASA centers, and it has engaged in international collaboration with other space agencies such as JAXA. The Astrobee Facility has transferred over 2.0 Terabytes of Guest Science ISS Operations data to users as of May 2025.

[3]https://github.com/nasa/astrobee_gds

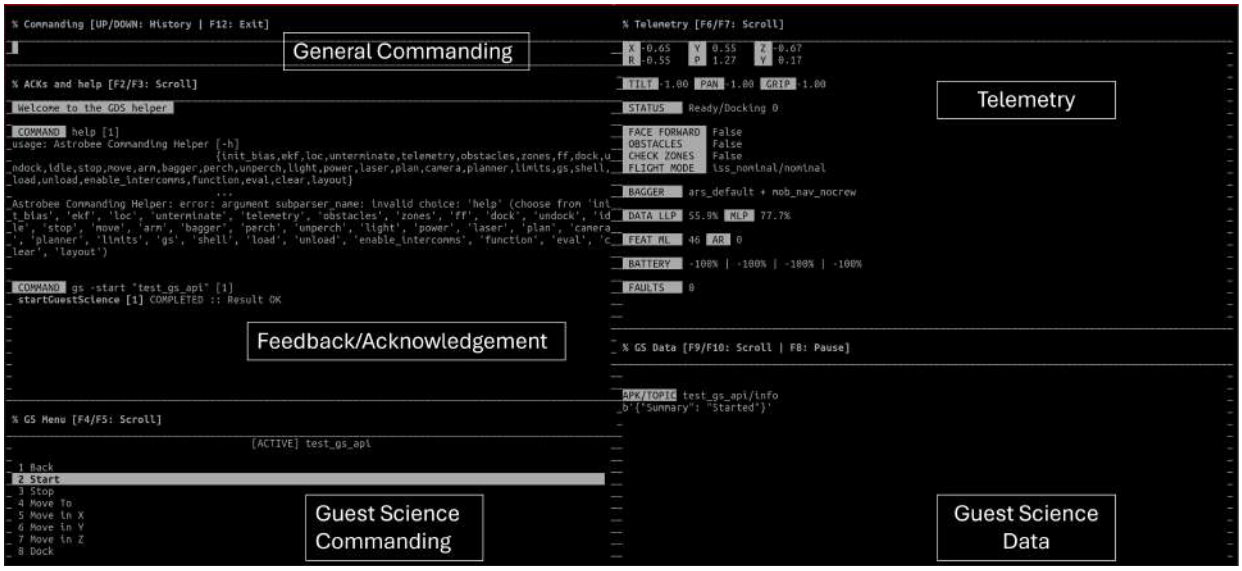


Fig. 36: GDS-Helper: Complementary text-based interface to Astrobee operators.

Robot	Commissioned	No. Sessions	HW Problems	Returned to Earth	Returned to ISS
Dock	2019-02-11	164	2	-	-
Batteries	2019-04-30	164	0	-	-
Bumble	2019-04-30	135	0	-	-
Honey	2019-10-30	59	1	2021-10-01	2023-08-30
Perching Arms	2021-02-04	23	0	-	-
Queen	2021-09-20	21	1	2023-06-30	-

TABLE X: Astrobee on-orbit activities summary

A. Astrobee Commissioning

Astrobee conducted on-orbit commissioning activities through the end of September 2019. The objective of commissioning was to fully validate the operational system prior to beginning Guest Science and utility operations. The commissioning activity schedule took an incremental approach, gradually demonstrating increasing capability, with repeat activities written into the schedule to account for the inevitable problems exposed when any robotic system is tested in a new environment. The activities relied on astronaut support, especially in the beginning, before Astrobee flew on its own. This section describes the commissioning activities.

1) L1-A5: Launch to Checkout: The installation and checkout procedures for the Dock and the Astrobee units systematically tested the functionality of every significant hardware component of the system. After months of effort on readiness testing and procedure maturation, these activities executed essentially flawlessly. There was a brief scare the first time the power button was pressed and Astrobee didn't boot, but it turned out the astronaut hadn't pressed the button hard enough. It was a tremendous relief to the team to know that the dock and Bumble Bee were still fully functional after enduring the stresses of launch.

Fig. 37: Part of the sparse map of the Kibo module, viewed top-down, with port end-cap at left. Each dot is a map feature available for localization. Map patchiness is attributed to areas with little texture and dim illumination.

2) A7-A8: Calibration and Mapping: The calibration and mapping (C&M) activity had an astronaut move an Astrobee around to collect imagery data, with two main tasks: (1) **Calibration** checked that the camera and IMU calibration taken on the ground had not changed significantly due to launch vibration. We decided to include this task after we found that the calibration changed slightly for at least one camera (NavCam) during vibration environment testing on the ground, presumably due to slip at the joint between the sensor board and the lens mounting bracket. To perform the calibration, the astronaut carried the Astrobee in scripted motions with a calibration target in view of the cameras. (2) **Mapping** collected NavCam data for building Astrobee's initial sparse map of the Kibo module where the dock is installed. The astronaut carried the Astrobee up and down Kibo several times, with the cameras facing different walls at different angles.

3) A10-A12: Localization and Mobility: The localization and mobility (L&M) activity had these main tasks: (1) **Localization** checked that Astrobee could localize itself using the map built in the C&M activity. An astronaut carried Astrobee in several motions through Kibo, similar to the C&M activity, but this time with its sparse map localizer running using the new map. (2) **Mobility** was Astrobee's first independent flight. The Astrobee performed a series of increasingly complicated maneuvers, culminating in flying a rectangular plan and performing an autonomous docking approach. (3) **Robustness** tested Astrobee's reaction to a few specific anomalies: (a) While Astrobee was station keeping, an astronaut applied a disturbance force. Astrobee resisted the force and stopped itself. (b) While the Astrobee was station keeping, an astronaut partially blocked the NavCam field of view. Astrobee continued station keeping. (c) While the Astrobee was moving forward, an astronaut moved into its path. Astrobee detected the obstacle and stopped.

Fig. 38: Bumble Bee on the dock. Credit: NASA / Christina Koch.

Fig. 39: Bumble Bee in flight. Credit: NASA / David St.-Jacques.

B. The Astrobee Facility

The Astrobee Facility at ARC manages all operations with the Astrobees. In particular, the facility is in charge of the Guest Science program, and it provides Guest Scientists with an ecosystem that includes software, hardware, test facilities, and operations and engineering teams [99], [98]. In this section we describe the components of the Astrobee Facility and the ways they support Guest Scientists.

Software Simulator. Guest Scientists interact with Astrobee software by writing APKs called Guest Science Applications. To facilitate development of Guest Science Applications, ARS includes the Astrobee Simulator, which simulates both the Astrobee hardware and the ISS environment. Guest Scientists develop APKs and test them in the Astrobee Simulator before running their code on ground units.

Ground Units. The Astrobee Facility includes three high-fidelity Astrobee ground units, which were built using the same parts and integration processes as the flight units. The ground units always run new code before anything is uplinked to the Astrobees on ISS.

Granite Lab. The Astrobee ground units are operated in the Granite Lab test facility. In the Granite Lab (Fig. 40), an Astrobee rides on an air bearing that glides over an extremely flat 1.2 $\times$ 1.2 m granite table surface, providing 3-DOF motion

(x, y, and yaw). The Granite Lab enables testing of Astrobee's propulsion system and motion control. The air bearing works by producing a CO_2 cushion under its feet, fed by two liquid CO_2 cartridges that last for $\sim$15 minutes of continuous use. A minor challenge is that the air bearing mass has a significant effect on motion control, and can't be modeled precisely because the mass of the cartridges varies as the CO_2 is consumed.

Astrobee can be mounted on the air bearing in a variety of orientations, including z-down (top up, most common for testing), y-up (on its side, enabling pitch rotations), and variant orientations using a goniometer cradle that adds up to 30° of tilt. A high-fidelity Dock ground unit is mounted in the Granite Lab to allow testing of the autonomous docking approach and Astrobee/dock avionics interaction. High-fidelity ISS handrail mockups enable perch testing.

The Granite Lab roughly simulates the visual environment of the ISS with backdrop posters printed from photos of the actual ISS interior. The lab has been mapped using Astrobee's sparse mapping software, and it can be used to test the computer vision localizer modules during integrated system testing. A tracking system in the Granite Lab can produce ground truth tracking of Astrobee's pose by combining custom software with repurposed HTC Vive consumer-grade virtual reality tracking hardware [16]. The Granite Lab also includes controlled lighting with a tunable intensity and color spectrum, which can simulate various ISS lighting conditions.

All potential on-orbit activities are performed in the Granite Lab prior to launch, to enable Guest Scientists to evaluate their experiments on the ground. Ground testing is a fundamental step in validating the results obtained through simulation and determining if the software deployed on an actual Astrobee unit performs as expected in a real-world environment.

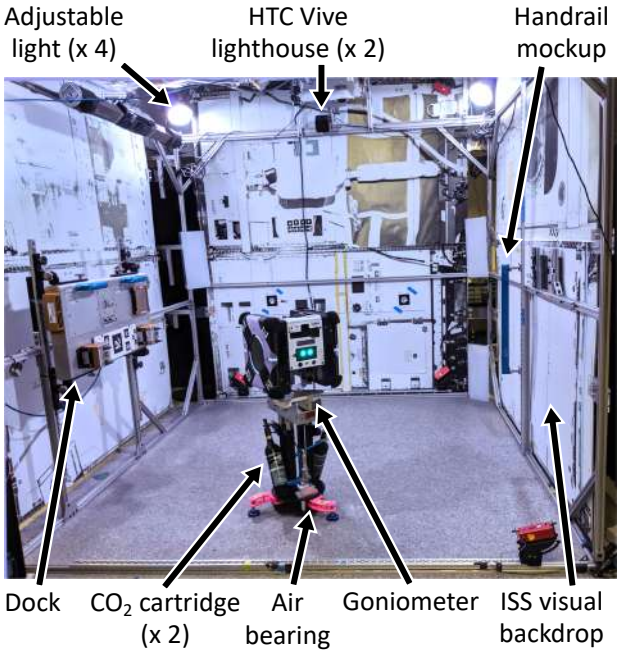


Fig. 40: Granite Lab

Operations and Engineering Teams. Completing an Astrobee Guest Science project requires developing software and possibly payload hardware for Astrobee, as well as working with the ISS office to schedule time and resources for any on-orbit activities. The Astrobee Ops and Engineering Teams support Guest Scientists through this process, providing technical expertise with software and hardware interfaces, as well as helping them with the certification and planning process described in VI-C. The Astrobee Teams also interface with ISS controllers and crew during Astrobee activities, and distribute data to the Guest Scientists after the activities.

Operations Center. The Astrobee Facility operates the Astrobees using a control room at ARC configured to connect to the ISS. In addition to workstations with Astrobee's Engineering Control Station installed, the control room provides standard resources such as voice loop headsets, shared big screens capable of viewing ISS camera feeds, and enough space to host Astrobee Guest Scientists if they choose to operate at ARC. Since the pandemic, operators have typically worked remotely (especially since the ISS operates on GMT and activities typically occur in the middle of the night California time).

C. Astrobee Guest Science

Astrobee's primary objective is to enable Guest Scientists to conduct research on a free-flying robotic platform inside the ISS. To become a Guest Scientist (GS), an individual, academic institution, or company starts by contacting the Astrobee Facility and communicating their interest in having an Astrobee payload or running an experiment with Astrobee. After initial contact is

established, there are four additional phases: Strategic, Tactical, Operations, and Post-Operations. The Strategic phase defines, at a high-level, who, what, where, and how the GS's research will be done. The Tactical phase includes technical planning, development, and evaluations. During the Operations phase, the hardware and/or software developed during the Tactical phase is run on the ground at the Granite Lab or on the ISS to complete the GS's science. The GS receives experimental data from the Operations phase during the Post-Operations phase.

1) Strategic Phase: The funding and timeline for an Astrobee payload vary greatly depending on complexity. In order to operate on the ISS, the investigator must be sponsored as an official ISS payload and listed in the ISS Integrated Payload List (IPL). There are various sponsorship routes. Investigators who are government funded (e.g. by NASA, National Science Foundation, or Defense Advanced Research Projects Agency) work with the ISS Program's Technology Development Office to get listed on the official ISS IPL. Foreign researchers are welcomed and require an international agreement between the GS's country's space agency and NASA. Commercial GSs can apply for ISS support through the non-profit Center for Advancement of Science in Space organization. Typical timelines to have a payload approved to be launched are on the order of six months to two years. A summary of this scheduling with approximate time includes: requirements development (two months), design (six months), and a science plan (four months). Some work on integration and hardware delivery (eight months) and simulation and software (four months) may also begin in the Strategic phase, depending on when the payload is listed in the IPL.

Once an investigator has an official payload on the IPL, they are considered a Guest Scientist and can proceed to the Tactical phase. An additional benefit to becoming a GS, besides access to the Astrobee Facility, is the Astrobee Technical Interchange Meeting. This semi-annual meeting provides an opportunity for information sharing across the Astrobee user community, including facility assessments on available resources, scheduling, and an open forum for developing new joint user opportunities.

2) Tactical Phase: In the Tactical phase, the GS develops their own software and possibly payload hardware, and tests them in simulation without using the Astrobee ground facilities. Figure 41 presents a conceptual interaction between the multiple components a GS works with, and represents a sequential approach to the development of experiments using the Astrobee research platform. The GS first learns about ARS [41] and develops their Guest Science Application in the form of an APK. The GS then uses the Astrobee Simulator to evaluate their APK. If they have developed their own payload hardware, they will also test their software with their hardware. These steps comprise the Tactical phase.

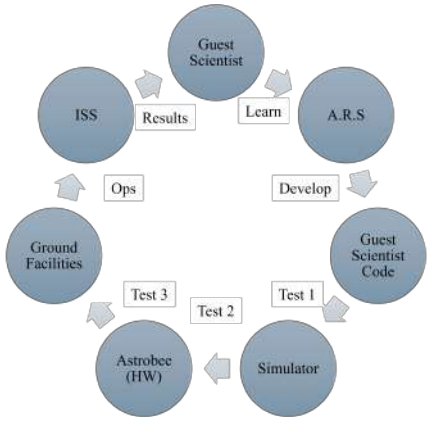


Fig. 41: Guest Science development lifecycle conceptual sequence

The Guest Science Application. GS applications are Android APKs that run on Astrobee's HLP. They use the Astrobee API or ROS to communicate with the rest of ARS. ROS gives GS apps access to all the data generated by ARS nodes, since the APKs run on the HLP which connects to the other processors. In contrast to ROS, the Astrobee API is a simple way for APKs to access a predefined set of telemetry, and it also includes predefined functions for APKs send commands to Astrobee.

GS apps that run on Astrobees on the ISS must not use Astrobee's touch screen to display data or accept inputs. The apps must take commands and retrieve telemetry only through the Astrobee Ground Data System (a software interface for astronauts and ground controllers).

There are two types of GS applications: primary and secondary. Primary GS apps are allowed to command the robot using the Astrobee API. They may also get telemetry from the Astrobee API or from ROS. Primary applications are ideal for executing trajectories and other active experiments.

Secondary GS applications are only allowed to get telemetry from the robot. They can get telemetry from the Astrobee API or from ROS, but they may not command the robot. Secondary applications are ideal for listener and monitoring software. They can also be used to get data generated by other applications running simultaneously.

3) Operations Phase: When a GS is satisfied with the results from the Tactical phase, they can test their APK and potential payload hardware on Astrobee robot hardware at the Astrobee ground facility (Granite Lab). The Astrobee Engineering Team

supports the GS during testing in the ground facility, and they also develop Ground Procedures to be executed during the on-orbit test sessions. Many Astrobee activities can be commanded from the ground without astronaut involvement.

While ground testing happens, the Astrobee Ops Team develops the required Flight Products that must pass stringent review from stakeholder teams at Johnson Space Center (JSC) in Houston, Texas and the NASA Marshall Space Flight Center (MSFC) in Huntsville, Alabama. Flight Products include detailed planning products that specify all of the resources (hardware, crew, network time, etc) required for each activity. If an astronaut is needed for the activity, Crew Procedures and On-Board Training materials are also developed. [19] The Flight Product review process culminates in a formal Engineering Change Request meeting attended by the JSC and MSFC teams. Once the Flight Products are approved and thus baselined, they become part of the ISS library of official Flight Documents.

If the GS has developed hardware, it must be certified by the ISS program and manifested on a launch vehicle to ISS. Once performance on the ground units is shown to be satisfactory, the Astrobee Ops Team uploads the Guest Science APK to the Astrobee robot(s) on board the ISS and works with MSFC to plan and schedule operations during which the science is conducted.

During a test session, depending on the science being performed, the GS may have live data feeds, or they may receive data after the operation. The GSs may also have the opportunity to participate in the live operation of their science. If an astronaut is required during real-time Astrobee operations, the Astrobee Ops Team is responsible for interacting with the crew member. Whether working directly with a crew member or not, the Astrobee Ops Team is also responsible for maintaining all communications with the Payload Operations Integration Center (PID) at Marshall Space and Flight Center and the Mission Control Center (MCC) at Johnson Space Center, managing real-time schedule deviations, and assisting the Astrobee Ground Engineering Team as needed during ISS operations. The Astrobee Ops Team is the focal point for all ISS real-time operations.

4) Post Operations Phase: During the Post Operations phase, the Astrobee Team archives all video taken during the Test Session, documents and tracks real-time statistics, and works with MSFC to implement any changes to Flight Procedures that may be needed for the next Test Session. Such changes are implemented and tracked through the Operations Change Request process.

GSs receive their experimental data during or shortly after the on-orbit test session in a private data pipeline or from a secure data transfer from the Astrobee Ops Team. As illustrated in Figure 41, the GSs can use the data to inform their next investigation.

D. Examples of Astrobee Research

Astrobees supports several types of research including manipulation, computer vision, and human-robot interaction. As of May 2025, the Astrobee Facility has supported 17 unique Guest Scientists in 22 different projects, and it plans to support four additional Guest Scientists and seven additional projects between 2024 and 2025. A comprehensive list of completed and pending Guest Science investigations, and the Astrobee capability used by each, is presented in Figure XI. For a list of over 30 publications that resulted from the Astrobee Guest Science program, see [75].

For Guest Scientists interested in developing hardware payloads that interface with the Astrobee robots, the mechanical interface can be found at Astrobee's website (https://nasa.gov/astrobee). In addition, Astrobee's flight software is hosted publicly in a Github repository (https://github.com/nasa/astrobee). These interfaces are an example of how the Astrobee Facilities enable and support Guest Scientists' own hardware and software payload design and development.

In this section, we highlight a few of the interesting research directions Guest Science has taken Astrobee research.

Autonomous Spacecraft Rendezvous and Docking. The Astrobee robots have demonstrated autonomous spacecraft rendezvous and docking. One robot tumbled in place, while a second approached to capture it [6], [5], [78], planning for the rendezvous with model predictive control [92].

Mapping and Navigation. As localization has been one of Astrobee's largest challenges, significant research has focused on improving robotic navigation, mapping, and localization techniques, leveraging Astrobee for experimental validation. This research has included semantic mapping and localization [63], approaches for robustness to changing lighting conditions [53], joint visual and time-of-flight camera calibration [28], ground truth evaluation with HTC Vive [16], detecting changes in the ISS [33], and creating acoustic maps [15].

Orbital Debris Removal. Astrobee has served as a platform to develop and test techniques for orbital debris removal using innovative materials and robotic systems [10].

Control and Path Planning. Astrobee has served as a platform for the development of multiple novel path path planning approaches, including a quadratic programming algorithm [101], an algorithm based on sequential convex programming [13], [14], and a kinodynamic-RRT [3]. Online information-aware motion planning with inertial parameter learning for robotic free-flyers has also been explored for single [36] and multiple robots [67].

Manipulation and Motion. An under-actuated hand design using mechanically-realizable manifolds was designed for Astrobee [23], as was a gecko-inspired gripper [21]. A novel form of movement was studied, where instead of using the propulsion modules, Astrobee uses its perching arm to toss itself between ISS handrails [7], [56], [8].

Educational Outreach. Astrobee also works as an educational platform through student competitions such as JAXA's Kibo Robotic Programming Challenge (Kibo-RPC, https://jaxa.krpc.jp/) intended for undergraduate university students and

	Investigation	Payload				HLP	MLP	LLP	Arm	Replace Gripper	Replace Skins	GDS	Sci Cam	Dock Cam
		Attachment		Power	Data									
		Lever	Screw											
Completed	Apiary						■	■						
	Astrobatics					■		■	■		■			
	Astrosee					■		■						■
	Clingers	■		■	■	■								
	Gecko					■	■		■	■				
	Gecko SCP						■							
	ISAAC					■	■					■	■	
	Kibo-RPC y1					■								
	Kibo-RPC y2					■						■		
	Kibo-RPC y3					■						■	■	
	Kibo-RPC y4					■						■	■	
	Kibo-RPC y5					■						■	■	
	MIT-ZR UAE					■								
	MIT-ZR y1					■							■	
	MIT-ZR y2					■							■	
	MIT-ZR y3					■							■	
	MRS	■		■	■	■						■		
	REACCH	■		■	■	■								
	REALM	■		■	■	■			■		■	■		
	REGGAE		■			■								
	RESWARM					■	■		■					
	ROAM					■	■		■					
	SoundSee	■		■	■	■								
	SVGS	■				■							■	
Pending	Kibo-RPC y6					■						■	■	
	MIT-ZR y4					■							■	
	SVGS2	■				■							■	

TABLE XI: Summary of Guest Scientists and corresponding Astrobee capability usage as May, 2025.

Massachusetts Institute of Technology (MIT)'s Zero Robotics (ZR) intended for middle and high school students [72]. Kibo-RPC alone has positively impacted thousands of students across the Asia Pacific region, and MIT-ZR has reached thousands of students across the United States.

a) b) c)

Fig. 42: Examples of different Guest Scientist payloads as those detailed in XI. a) CLINGERS [22]. b) MRS [30]. c) REACCH [68].

VII. Lessons Learned

In this section, we discuss some elements of Astrobee's design and development approach that did or did not prove to work well.

- **Early Prototyping.** Astrobee invested in building multiple generations of integrated prototypes even before the formal system requirements review with stakeholders. The experience gained during that early testing was vital for fleshing out the architectural assumptions and subsystem interfaces that allowed us to develop relevant and verifiable requirements.
- **Keeping an Open Mind as Objectives Change.** The Smart SPHERES project had, as one of its more ambitious secondary goals, demonstrating SLAM to build a map of portions of the ISS interior while controlling the flight of a SPHERES satellite. That effort had promising initial results, but was not fully realized. When the Smart SPHERES veterans on our team started to design the vision-based navigation for Astrobee, it was natural to see the Smart SPHERES SLAM code base as the obvious core technology to build on. However, we kept an open mind and realized that, because Astrobee operates in the bounded and mostly static ISS environment, it is acceptable to collect on-orbit imagery data with astronaut support, build a map on the ground (where it can be manually registered to the ISS CAD model used in the Control Station), then uplink the map for on-orbit use. This approach resulted in simpler and more robust flight software.
- **Total Life-Cycle Impact of Complexity.** Systems implemented purely for research can tolerate a much greater degree of complexity than systems that must support end users over an extended lifespan. This is particularly true for human

spaceflight systems that have challenging additional requirements related to the launch vibration environment and astronaut safety. Every additional part triggers another safety analysis and is another potential point of failure for the deployed system. In retrospect, given the hurdles we encountered already during development and verification, some of the "nice-to-have" components of the Astrobee design (e.g. flashlights, laser pointer, microphone) perhaps should not have made the cut for inclusion in the baseline Astrobee hardware, especially given we have the ability to add them as part of future payloads.

- **Importance of Modularity in Large Projects.** With multiple subsystem teams working in parallel, it was important to use a modular design with clear interface definitions so that component-level verifications would assure the system would work when integrated. Some Astrobee design choices that improve modularity include: (1) Propulsion airflow is decoupled from the core avionics. (2) Computing is distributed across three processors to isolate robust real-time code from experimental code. (3) Astrobee's flight software embodies a modular architecture with clean separation between ROS nodes. However, in some cases our best efforts at modularity failed due to interfaces we overlooked. For example: (1) Electromagnetic interference (EMI) injected onto the main power bus by the propulsion modules caused communications failures, discovered late in the integrated testing process, triggering a redesign that added months of delay to the project schedule. One cause was that "power quality" EMI requirements at the module interface had never been defined or verified. (2) Although the acoustic noise generated by individual nozzles subjectively seemed insignificant, it was unexpectedly amplified when the nozzles were mounted on the plenum, whose base is a thin aluminum sheet that apparently acts as a sounding board; the nozzle mounting had to be modified to include acoustic isolation.
- **Schedule Overlap Trade-Offs.** The Astrobee project had initially planned to build a fully flight-like certification unit that would go through formal verifications prior to starting integration and test (I&T) of the flight units. During the final phases of development, we made a difficult decision to add one more generation of integrated prototype testing prior to starting the certification unit, which unavoidably slipped the certification unit I&T schedule to overlap substantially with flight unit I&T. The result was that when lessons learned during certification forced design changes, in some cases we had to rework the flight units—a problem we had multiple times. Nevertheless, with hindsight, we believe we took the correct approach to ensure the design was sufficiently mature before starting the certification unit.
- **Trade-Offs in Testing Rigor.** There is often a tension between rigorous testing to fully validate robustness versus the desire to avoid costly delays due to damaging one-of-a-kind prototype hardware. An example is Astrobee's heat sink assembly for the MLP and HLP, which had a design flaw that caused it to damage the processors during launch vibration testing of the certification unit, forcing a difficult late redesign. We missed an early opportunity to discover the flaw when we conducted an informal risk reduction vibration test on the avionics stack, but (with the best of intentions) chose to run the test at a reduced amplitude relative to the final verification test, and so avoided triggering the problem. With better discussion and clarity about test objectives, we would have tested at full amplitude and caught the problem early.
- **Limits of Designing for Repair.** Whenever field-expedient repair is possible, there is a question of how to design to enable that repair. Astrobee started with an ambitious approach of requiring many components to be ORUs. That approach complicated the design, requiring the use of captive fasteners, constraining the geometry to enable astronaut access, and increasing size and weight. In retrospect, it would have been better to focus on replaceability of only a few larger sub-assemblies, achieving most of the benefit at lower cost.
- **3D Printing Considerations.** Through many failures involving 3D printed plastic parts, our team collected 3D printing rules of thumb that other projects may find useful, including: (1) Prefer assembly using brass threaded heat-set inserts that avoid galling. (2) Incorporate increased tolerances for 3D printing variability, and complement with post-print machining when tighter tolerances are needed (e.g. drill out holes for inserts). (3) If appearance is important, bead blasting and painting work well, but manual finishing steps greatly impact cost and schedule, and transitioning to painted parts may cause new problems with fit. (4) Use caution when transitioning from in-house prototyping to an outside vendor, as differences in printing processes can cause new problems.
- **COTS Part Obsolescence.** The development cycle for spaceflight projects tends to be much longer than that for consumer electronics devices; projects should plan accordingly. Astrobee benefited greatly from availability of COTS integrated parts like the IMU and cell-phone-derived system-on-module MLP and HLP processors, but obsolescence caused problems, including: (1) A few parts (e.g., the original SciCam) reached end-of-life and became unavailable before we stocked sufficient inventory to last for the life of the project, forcing us to redesign with an alternative part. (2) In a cascading effect, the new SciCam part required a new HLP part for compatibility, which in turn triggered significant effort porting parts of the flight software to the new HLP. (3) Long-anticipated Linux drivers enabling Graphics Processing Unit (GPU)-accelerated computer vision on the MLP never materialized because the vendor lost interest after they released their next-generation processor board.
- **Balancing Research with Deliverables.** Many components of ARS, such as the localization and planning algorithms, are novel in the field of robotics and are therefore the outcome of significant research. The innovative nature of the contribution means that work itself is unbounded – there are many research directions that might lead to an incremental performance improvement. Balancing the requirement to deliver a product with the desire to explore further research avenues is challenging.

Function	Coder	Eigen	Improvement
of_residual_and_h	87.0s	2.0s	98% (43x)
delta_state_and_cov	22.5s	3.5s	84% (6x)
covariance_multiply	18s	< 2s	89% (~10x)

TABLE XII: Performance of some time-consuming functions using the Mathworks Simulink **Coder** compared to hand written code using the **Eigen** library.

- **Hardware Constraints.** Astrobee, and in particular its propulsion system, was designed in-house across several groups at NASA using a combination of off-the-shelf and bespoke hardware. Developing hardware as a series of prototypes in lock-step with software necessitates communal access, which required careful planning and coordination. In the future, procuring sufficient quantities early in development would mitigate the risk of unavailable parts.

  Furthermore, as a result of design constraints and hardware availability, the processing modules[4] within Astrobee differ from one another, with the exception of the LLP and dock processor. Even though various boards' kernel versions match, numerous customizations were required for each module. In addition, Astrobee runs both Linux and Android which forces the team to acquire expertise with two development environments and create different tools to maintain each system. This heterogeneity taxes software development effort.

  Finally, the pace of the smartphone industry is much faster than that of a project like Astrobee, which spans multiple years. Products like processors and cameras become obsolete before the project is mature enough to commit to acquisition in sufficient quantities. This situation forces necessary software adaptations, which cost substantial time.
- **Software Optimization.** The use of an embedded computing platform for computationally intensive algorithms necessitates optimization. For example, we replaced certain auto-generated Simulink C code with equivalent Eigen implementations, which led to drastic performance increases which we show in Table XII. The times in the table are totaled over the run of a recorded data set. The performance increases are possible because Eigen exploits the accelerated NEON instruction set on ARM, whereas Simulink does not. Simulink code also proved difficult to maintain and has now been entirely removed.

  The MLP also offers a GPU that may be used to optimize other parts of flight software. Image processing algorithms are typically well-suited to this type of hardware acceleration. In the future, we will consider using the GPU to improve performance.
- **ISS Operations.** Astrobee has been operating for six years and has seen many sessions and thousands hours of operations. The overall performance has surpassed expectations, and the greatest performance improvements have been in localization. Astrobee continues to be used by various investigations, and each investigation is unique with its own requirements and parts of the systems used. The most used parts of the system are the functional capabilities, and the computing resources. Astrobee continues to be useful for meaningful science while striving to improve the core capabilities. A summary of the lessons learned from an operations perspective are:
  - The ISS is a dynamically changing environment and maintaining reliable maps became a task requiring more frequent updates than initially estimated.
  - Localization algorithms need to be able to function stably for a time without mapped landmarks, the longer the better.
  - Until a map has good features over the entire area, maneuvers will always need to point periodically to areas with good features.
  - After several activities dedicated to gather imagery data towards mapping, less reliance on crew assistance was required and accomplished thanks to improvements in the localization upgrade from the initial extended Kalman filter method.
  - Given the unexpected high frequency in ISS activities and the correspoding high amount of data collected (over 2.0 Terabytes to date), the Astrobee Facilities team had to develop a mechanism to quickly treat it and obtain the required clearance to distribute it back to the user.
  - Users that developed software for the MLP required much more laboratory testing and development time. Guest Scientists such as AstroSee and Astrobatics, who developed software running at the MLP also developed software targeting the LLP.
  - The HLP is indeed a useful and highly used processor. The HLP running Linux would have been easier for user integration and even off-loading the MLP. Users continue to have interest in running Linux on the HLP. MIT and Gecko projects used the HLP to send commands and telemetry
  - Frequent anomalies during ISS operations drove most users to rely on the Astrobee team for command and control. Special data display requirements led to users needing telemetry directly.
  - Operator training for poor localization situations took longer than the planned one hour of training. The Astrobee facility is running a study to better understand the impact GDS has in the operator's cognitive load during an ISS activity; GDS Helper serves as a comparison baseline in this study.

[4]A processing module contains a CPU, GPU, memory and peripherals like I2C, USB and network adapters.

- The GDS on a station laptop is not needed when ground commanding is possible.
- Human interaction elements are unlikely to be incorporated by a user not specifically studying interaction. Only JAXA's Kibo-RPC has use the laser pointer, flashlights, and signal lights as means of communicating the robot's state or intent during their competition. MIT-ZR has also used the flashlights for this same purpose.
- Hardware resources include power, data, the mechanical attachment options (quick-release levers or screws) available for the payloads, and the perching arm. Most users prefer the quick-release lever option when it is shown to work. Only one user, REGGAE, used screws, and they did not use power or data. Most users prefer Ethernet over USB for communicating with their payload.
- Matlab's Simulink controller generated code ran slower than what Astrobee required and had to be replaced by in-house developed software which also facilitated maintenance.

VIII. Related Work

A. Past ISS Free Flyers

A major inspiration for Astrobee was the highly successful SPHERES satellites that operated inside the ISS for 13 years. Astrobee and SPHERES were compared in §I. Some of us were part of the ARC team that managed the SPHERES Facility on the ISS and took over management of the Astrobee Facility after Astrobee completed commissioning. Some of us were also part of the Smart SPHERES project [43], which demonstrated leveraging COTS smart phone hardware to integrate a variety of sensors and enable the SPHERES satellites to perform inspection tasks. Among many connections, Astrobee's Control Station draws significant heritage from a similar interface evaluated by Smart SPHERES.

Our team also includes veterans from the Personal Satellite Assistant (PSA) project [35], an ambitious early effort to develop a robotic assistant for astronauts inside the ISS. Although multiple generations of prototypes were built, PSA never flew. Astrobee benefited from PSA propulsion design heritage and inherited the MGTF gantry originally built for PSA.

B. Other ISS Free Flyers

Two other free flyers whose development overlapped with Astrobee have recently launched to the ISS. Japan Exploration Module **IntBall 2**, developed by JAXA, is an IV free-flying camera designed to observe astronaut activities [95], [64], [103]. Its impressively small size (IntBall1: 15 cm sphere, 1 kg mass, IntBall2: 20cm sphere, 3 kg mass) is enabled by JAXA's miniaturized all-in-one CPU plus IMU plus 3-axis reaction wheel module developed with small-sats in mind. The Phase 1 Int-Ball had a rapid 18-month development cycle and launched in 2017. A Phase 2 Int-Ball is currently operational at the ISS. Although Astrobee and Phase 2 Int-Ball share the free-flying camera use case, they are otherwise quite different; Astrobee 6 cameras for navigation and task monitoring whereas Int-Ball2 has 2 cameras; Astrobee has 12 nozzles with $\sim$2 times higher maximum propulsion thrust, and includes many unique features designed to support Guest Science research. JAXA and NASA are currently collaborating to develop joint K-12 education activities involving Int-Ball and Astrobee.

The Crew Interactive Mobile Companion (CIMON), developed by the German Aerospace Center (DLR)/Airbus, is another IV free flyer, intended to provide assistance and companionship to astronauts [87]. It is backed by ground-based artificial intelligence for dialog management and autonomous HRI. In size it is closer to Astrobee (32 cm diameter sphere, 5 kg). It excels at human interaction, but again does not pursue many Astrobee capabilities, such as Guest Science support, autonomous docking for resupply, and a robot arm. An agreement between NASA and DLR has enabled CIMON to share use of the pool of Inspired Energy batteries certified for the ISS by NASA's GeoCam Space and Astrobee projects.

Figure 43 shows mockups of three free-flyers on board of the ISS: NASA's Astrobee, JAXA's IntBall, and DLR/Airbus' CIMON in display at the International Astronautical Conference 2018 in Washington, D.C.

C. Extra-Vehicular Free Flyers

Beyond IV free flyers, most spacecraft can be considered EV free flyers, and there is considerable heritage for proximity operations relevant to Astrobee. Without going into familiar large spacecraft examples such as Gemini and Soyuz, more recently several smaller spacecraft projects are pursuing proximity operations. AERCam Sprint, an EV mobile camera designed to be joysticked by a Space Shuttle astronaut with line-of-sight through a window, was successfully flight tested on STS-87 in 1997. A follow-on project to develop a miniaturized version called Mini AERCam reached the prototype stage, but was never launched [46]. The U.S. Air Force Research Lab Experimental Satellite System-11 demonstrated in-space inspection with a semi-autonomous 100 kg spacecraft [9]. In September 2019, NASA's Seeker 3U CubeSat demonstrated in-space inspection of a Cygnus vehicle after it undocks from the ISS, prior to disposal; it happened to launch on the same NG-11 cargo flight as the first two Astrobees [82]. Astrobee is an ideal low-risk testbed to evaluate technologies for future deployment to EV free flyers like these.

Fig. 43: Three Friends: NASA's Astrobee, JAXA's IntBall, and DLR/Airbus' CIMON on display at the International Astronautical Congress 2018.

D. Terrestrial Free Flyers

Astrobee's development has also benefited by leveraging intense research interest in terrestrial micro aerial vehicles (MAVs) such as quadrotor drones. A few examples are: (1) In Astrobee's navigation trade study, our selected option of using vision-based navigation with a monocular camera was supported by heritage such as [1], although our actual implementation is somewhat different [26]. (2) The PX4FLOW component of Astrobee's SpeedCam is a COTS device intended to provide optical flow-based navigation and control for a MAV. Its availability greatly simplified implementing an independent over-speed cutoff. (3) Astrobee's signal light design was influenced by past work on HRI for MAVs [94].

IX. Astrobee's Future

Name	Developer	Description	Launch
Gecko Gripper	Stanford	Replacement gripper for Astrobee arm uses gecko-inspired adhesives to cling to flat surfaces [39]	SpX-18 (7/2019)
RFID Recon	NASA JSC	Mobile RFID reader to track location of critical items on ISS [40]	NG-12 (10/2019)
SoundSee	Astrobotic, Bosch USA	Beam-forming microphone array for mapping acoustic anomalies such as failing mechanisms or leaks	NG-12 (10/2019)
Astrobatic	Naval Postgraduate School	Astrobee demonstrates propellant-less mobility using its arm to throw itself from handhold to handhold [85]	Software only
Zero Robotics	MIT	K-12 students write code to fly simulated Astrobee through a game; finalist code runs on the ISS [60]	Software only
Kibo Robot Programming Challenge	JAXA, NASA	JAXA-led student programming competition; may also involve Int-Ball in the future	Software only

TABLE XIII: Astrobee Guest Science activities that have operated and remain operational

Hopefully, like the long-lived SPHERES satellites, Astrobee will continue operating on the ISS for years to come. As a Guest Science platform, the Astrobee Facility has a registry of more than 40 projects that have expressed interest in using Astrobee. Some of the projects that have operated and maintained annual operations on board the ISS are listed in Table XIII. Multiple researchers have also investigated potential improvements to Astrobee path planning and replanning in dynamic environments [102], [12]. In Summer of 2025, NASA selected a commercial company, Arkisys, to continue to operate Astrobee as a research platform. Arkisys will provide expanded opportunities for commercial companies to develop, test and operate in the ISS environment, and achieved their first customer activity in November 2025. Astrobee is expected to continue operations until ISS decommission. For information about how to become an Astrobee Guest Scientist, see [73].

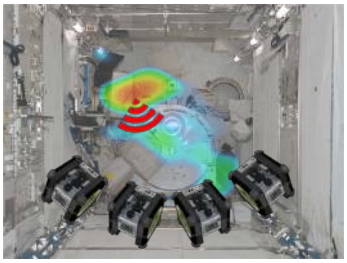


Fig. 44: Leak localization concept: As it moves, an IVR free flyer samples with a beam-forming ultrasonic microphone array, gradually narrowing its probability distribution over possible leak locations (heat map).

Intra-vehicular robots (IVR) will play a key role in the future missions of NASA's Moon-to-Mars exploration architecture. For example, current plans for the lunar Gateway space station call for it to be uncrewed more than 85% of the time [29]. Some of us are involved in the Gateway program's IVR working group, which is studying possible robotic caretaking use cases during uncrewed periods, including fault recovery (Fig. 44) and transferring cargo from visiting uncrewed cargo vehicles into the Gateway. Astrobee will be useful as a reference implementation to derive requirements for a possible Gateway IVR mobile inspection robot.

We are also working toward realizing Astrobee's potential to support ISS operations, related to the *free-flying camera* and *sensor survey* use cases. As a free-flying camera controlled by ground operators, an Astrobee can observe astronaut activities, run a visual status check in case of an anomaly, or help search for a missing item. An example sensor survey application is automating the repetitive task of collecting video surveys of the ISS interior, which astronauts currently perform using a camcorder. In addition to saving astronaut time, an Astrobee video survey with associated camera pose tracking can more readily be processed into a 3D model to enable spatial browsing, providing greater value to analysts. Many other useful survey data sets can be collected with Astrobee's baseline hardware or by carrying miniaturized sensors already being developed for ISS use, including: acoustic noise, CO_2 concentration, radiation, and WiFi signal strength [19]. However, across applications, a business case must be made that replacing the status quo with a new robotic survey approach will provide positive return on investment within the service life of Astrobee and the ISS. We are embarking on those discussions now.

X. Conclusion

Astrobee was designed specifically to facilitate Guest Science on ISS; it requires minimal crew attention, can navigate and dock itself to recharge, and it has payload bays, cameras, and computing power. With six years deployed in space, Astrobee has supported over 21 unique Guest Science investigations from academia, industry, other NASA centers, and international partners such as JAXA and DLR, making Astrobee the most utilized research platform on board of the ISS. Astrobee has served as a successful platform for Guest Science, a feasibility demonstration and reference implementation for IVR free flyers on future exploration missions, and a tool to support ISS operations.

Acknowledgments

We thank the many others who contributed to the Astrobee project, including but not limited to: Stephen Battazzo, Jeffrey Blair, Michael Dille, Ryan Goetz, Ravi Gogna, Robert Hanson, Ali Kashani, Hyunjung Kim, Brian Koss, Dong-Hyun Lee, Dongmeng Li, Jason Q. Lum, Ruojuan J. Ma, Nghia Mai, Donald Morr, Robert Nakamura, Gregory Paulson, Cedric Priscal, Estrellina Pacis Rius, Arno Rogg, Corey Snyder, Donald Soloway, Antoine Tardy, Recaredo J. Torres, Shang Wu, and Allison Zuniga.

Many thanks to our interns for their energy, fresh perspective and contributions; our guest scientists for their endless creativity; and our colleagues in the ISS program for their patience and support. Astrobee development is funded by NASA's Game Changing Development program. The Astrobee Facility sustaining funding was provided by NASA's Advanced Exploration Systems program through 2019 and has since been funded by the ISS Program Research Integration Office.This article is dedicated to the memory of our beloved friends and colleagues Darryl LeVasseur and Ted Morse, who passed away in 2017 and 2024, respectively. Ad astra, Darryl and Ted.

References

[1] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[2] K. Albee, M. Ekal, and C. Oestreich, "A brief guide to astrobee's flight software," 2020.

[3] K. Albee and B. Coltin, "Kinodynamic-rrt for robotic free-flyers: Generating feasible trajectories for on-orbit mobile manipulation," in *Proc. IEEE Int. Conf. Rob. Automation (ICRA) Workshop: Toward Online Optimal Control of Dynamic Robots*, 2019.

[4] K. Albee, M. Ekal, B. Coltin, R. Ventura, R. Linares, and D. W. Miller, "The rattle motion planning algorithm for robust online parametric model improvement with on-orbit validation," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 946–10 953, 2022.

[5] K. Albee *et al.*, "Autonomous rendezvous with an uncertain, uncooperative tumbling target: the tumbledock flight experiments," in *16th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2022)*, 2022.

[6] K. Albee, C. Oestreich, C. Specht, A. Teran Espinoza, J. Todd, I. Hokaj, R. Lampariello, and R. Linares, "A robust observation, planning, and control pipeline for autonomous rendezvous with tumbling targets," *Frontiers in Robotics and AI*, p. 234, 2021.

[7] K. P. Alsup, "Robotic spacecraft hopping: Application and analysis," Master's thesis, Naval Postgraduate School, 2018.

[8] K. P. Alsup, J. Virgili-Llop, J. Komma, and M. Romano, "Analysis and simulation of robotic hopping maneuvers inside the International Space Station with Astrobee," in *Proc. American Astronautical Society, Space Flight Mechanics Meeting*, Maui, 2019.

[9] H. Baker, "XSS-11 micro satellite fact sheet," https://www.kirtland.af.mil/Portals/52/documents/AFD-111103-035.pdf?ver=2016-06-28-110256-797, 2011, accessed: 2019-07-09.

[10] M. K. Ben-Larbi *et al.*, "Orbital debris removal using micropatterned dry adhesives: review and recent advances," *Preprint submitted to Elsevier*, 2022.

[11] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

[12] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE Conf. Rob. Automation (ICRA)*, 2019. [Online]. Available: https://arxiv.org/pdf/1903.00155.pdf

[13] R. Bonalli *et al.*, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE Int. Conf. Rob. Automation (ICRA)*, 2019.

[14] ——, "Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach," in *Proc. Robotics: Science and Systems (RSS)*, 2019.

[15] L. Bondi, G. Chuang, C. Ick, A. Dave, C. Shelton, B. Coltin, T. Smith, and S. Das, "Acoustic imaging aboard the international space station (iss): Challenges and preliminary results," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 5108–5112.

[16] M. Borges, A. Symington, B. Coltin, T. Smith, and R. Ventura, "HTC Vive: Analysis and accuracy improvement," in *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[17] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm," *Intel corporation*, vol. 5, no. 1-10, p. 4, 2001.

[18] M. Bualat, D. Schreckenghost, E. Pacis, T. Fong, D. Kalar, and B. Beutter, "Results from testing crew-controlled surface telerobotics on the International Space Station," in *Proc. Int. Symp. on AI, Robotics, and Automation in Space (i-SAIRAS)*, 2014.

[19] M. G. Bualat, T. Smith, E. E. Smith, T. Fong, D. Wheeler, and the Astrobee Team, "Astrobee: A new tool for ISS operations," in *Proc. SpaceOps (AIAA 2018-2517)*, 2018.

[20] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 4290–4297.

[21] A. Cauligi, T. G. Chen, S. A. Suresh, M. Dille, R. G. Ruiz, A. M. Vargas, M. Pavone, and M. Cutkosky, "Design and development of a gecko-adhesive gripper for the astrobee free-flying robot," *arXiv preprint arXiv:2009.09151*, 2020.

[22] S. E. R. Center, "CLING and CLINGERS - Space Engineering Research Center — isi.edu," https://www.isi.edu/centers-serc/research/rendezvous-and-proximity-operations-rpo/cling-and-clingers/, 2026, [Accessed 13-01-2026].

[23] T. Chen, M. Haas-Heger, and M. Ciocarlie, "Underactuated hand design using mechanically realizable manifolds," in *Proc. IEEE Int. Conf. Rob. Automation (ICRA)*, 2018.

[24] M. Ciocarlie, F. M. Hicks, R. Holmberg, J. Hawke, M. Schlicht, J. Gee, S. Stanford, and R. Bahadur, "The Velo gripper: A versatile single-actuator design for enveloping, parallel and fingertip grasps," *Int. J. Rob. Research*, vol. 33, no. 5, pp. 753–767, 2014.

[25] B. Coltin, J. Fusco, Z. Moratto, O. Alexandrov, and R. Nakamura, "Localization from visual landmarks on a free-flying robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4377–4382.

[26] ——, "Localization from visual landmarks on a free-flying robot," in *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 4377–4382, video.

[27] T. Conceição, B. Coltin, A. Symington, L. Flückiger, and T. Smith, "Joint visual and time-of-flight camera calibration for an automatic procedure in space," in *Proc. Int. Symp. on AI, Robotics, and Automation in Space (iSAIRAS)*, 2018.

[28] T. Conceição *et al.*, "Joint visual and time-of-flight camera calibration for an automatic procedure in space," in *Proc. Int. Symp. on AI, Robotics, and Automation in Space (i-SAIRAS)*, 2018.

[29] J. C. Crusan, R. M. Smith, D. A. Craig, J. M. Caram, J. Guidi, M. Gates, J. M. Krezel, and N. B. Herrmann, "Deep Space Gateway concept: Extending human presence into cislunar space," in *Proc. IEEE Aerospace*, March 2018.

[30] CSIRO, "Multi-resolution mapping revolution in space — csiro.au," https://www.csiro.au/en/news/all/articles/2024/march/multi-resolution-scanning, 2024, [Accessed 13-01-2026].

[31] C. Davis, "Theory of positive linear dependence," *American Journal of Mathematics*, vol. 76, no. 4, pp. 733–746, 1954.

[32] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.

[33] H. Dinkel, J. Di, J. Santos, K. Albee, P. V. Borges, M. Moreira, R. Soussan, O. Alexandrov, B. Coltin, and T. Smith, "Astrobeecd: Change detection in microgravity with free-flying robots," *Acta Astronautica*, 2024.

[34] B. Doerr, K. Albee, M. Ekal, R. Ventura, and R. Linares, "The reswarm microgravity flight experiments: Planning, control, and model estimation for on-orbit close proximity operations," *Journal of Field Robotics*, 2023.

[35] G. A. Dorais and Y. Gawdiak, "The Personal Satellite Assistant: An internal spacecraft autonomous mobile monitor," in *Proc. IEEE Aerospace*, 2003.

[36] M. Ekal *et al.*, "Online information-aware motion planning with inertial parameter learning for robotic free-flyers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[37] R. C. Elphic, J. L. Heldmann, M. M. Marinova, A. Colaprete, E. L. Fritzler, R. E. McMurray, S. Morse, T. L. Roush, C. R. Stoker, M. C. Deans, and T. Smith, "Simulated real-time lunar volatiles prospecting with a rover-borne neutron spectrometer," *Adv. Space Res.*, vol. 55, no. 10, pp. 2438–2450, 2015.

[38] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[39] M. A. Estrada, H. Jiang, B. Noll, E. W. Hawkes, M. Pavone, and M. R. Cutkosky, "Force and moment constraints of a curved surface gripper and wrist for assistive free flyers," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[40] P. W. Fink, T. F. Kennedy, L. Rodriguez, J. L. Broyan, P. H. Ngo, A. Chu, A. Yang, D. M. Schmalholz, R. W. Stonestreet, R. C. Adams, J. Berger, A. K. Merta, F. J. Graffagnino, P. Shenoy, E. Cecchet, and J. Gummeson, "Autonomous logistics management systems for exploration missions," in *Proc. AIAA SPACE*, 2017. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2017-5256

[41] L. Flückiger, K. Browne, B. Coltin, J. Fusco, T. Morse, and A. Symington, "Astrobee robot software: Enabling mobile autonomy on the iss," in *Proc. Int. Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2018.

[42] L. Flückiger and H. Utz, "Service Oriented Robotic Architecture for space robotics: design, testing, and lessons learned," *J. Field Robot.*, vol. 31, no. 1, pp. 176–191, 2014.

[43] T. Fong, M. Micire, T. Morse, E. Park, C. Provencher, V. To, and D. Wheeler, "Smart SPHERES: A telerobotic free-flyer for intravehicular activities in space," in *Proc. AIAA Space*, 2013.

[44] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Proc. RSS*. Georgia Institute of Technology, 2015.

[45] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[46] S. E. Fredrickson, L. W. Abbott, S. Duran, J. D. Jochim, J. W. Studak, J. D. Wagenknecht, and N. M. Williams, "Mini AERCam: development of a free-flying nanosatellite inspection robot," *Space Sys. Tech. Oper.*, vol. 5088, 2003. [Online]. Available: https://doi.org/10.1117/12.498108

[47] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *Proc. Int. Conf. Intelligent Robots and Systems (IROS)*, 2013.

[48] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[49] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[50] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.io. [Online]. Available: http://octomap.github.io

[51] S. Kang *et al.*, "Astrobee iss free-flyer datasets for space intra-vehicular robot navigation research," *IEEE Robotics and Automation Letters*, vol. 9, no. 4, pp. 3307–3314, 2024.

[52] J. Kelsey, J. Byrne, M. Cosgrove, S. Seereeram, and R. Mehra, "Vision-based relative pose estimation for autonomous rendezvous and docking," in *2006 IEEE Aerospace Conference*, 2006, pp. 20 pp.–.

[53] P. Kim *et al.*, "Robust visual localization in changing lighting conditions," in *Proc. Int. Conf. Rob. Automation (ICRA)*, 2017.

[54] Y. Kim and T. Fong, "Signaling robot state with light attributes," in *Proc. IEEE/ACM Int. Conf. on Human-Robot Interaction (HRI)*, 2017, pp. 163–164.

[55] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[56] J. Komma, "Mechatronics: The development, analysis, and ground-based demonstrations of robotic spacecraft hopping with a manipulator," Master's thesis, Naval Postgraduate School, 2018.

[57] D.-H. Lee, B. Coltin, T. Morse, I.-W. Park, L. Flückiger, and T. Smith, "Handrail detection and pose estimation for a free-flying robot," *Int. J. Adv. Robot. Syst.*, vol. 15, no. 1, 2018. [Online]. Available: https://doi.org/10.1177/1729881417753691

[58] S. Y. Lee, D. Lees, T. Cohen, M. Allan, M. Deans, T. Morse, E. Park, and T. Smith, "Reusable science tools for analog exploration missions: xGDS web tools, VERVE, and Gigapan Voyage," *Acta Astronaut.*, vol. 90, no. 2, pp. 268–288, 2013.

[59] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[60] J. Liu, W. Feenstra, M. Hunt, A. Siegel, K. McGrane, and A. Saenz-Otero, "Students touch space in Zero Robotics programming competition with free downloadable curriculum," in *Proc. IEEE Aerospace Conf.*, 2014.

[61] M. Micire, T. Fong, T. Morse, E. Park, C. Provencher, E. Smith, V. To, R. J. Torres, D. Wheeler, and D. Mittman, "Smart spheres: a telerobotic free-flyer for intravehicular activities in space," in *AIAA SPACE 2013 Conference and Exposition*, 2013, p. 5338.

[62] I. Miller, R. Soussan, B. Coltin, T. Smith, and V. Kumar, "Robust semantic mapping and localization on a free-flying robot in microgravity," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[63] I. D. Miller *et al.*, "Robust semantic mapping and localization on a free-flying robot in microgravity," in *International Conference on Robotics and Automation (ICRA)*, 2022.

[64] S. Mitani, M. Goto, R. Konomura, Y. Shoji, K. Hagiwara, S. Shigeto, and N. Tanishima, "Crew-supportive autonomous mobile camera robot on ISS/JEM," in *Proc. Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2018.

[65] ——, "Int-ball: Crew-supportive autonomous mobile camera robot on iss/jem," in *2019 IEEE Aerospace Conference*. IEEE, 2019, pp. 1–15.

[66] S. Mohan, A. Saenz-Otero, S. Nolet, D. W. Miller, and S. Sell, "SPHERES flight operations testing and execution," *Acta Astronaut.*, vol. 65, no. 7-8, pp. 1121–1132, 2009.

[67] M. Moreira, B. Coltin, and R. Ventura, "Cooperative real-time inertial parameter estimation," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1934–1936.

[68] A. Morris, "REACCHing for the Stars — KMI — kallmorris.com," https://www.kallmorris.com/columns/reaccching-for-the-stars, 2024, [Accessed 13-01-2026].

[69] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3565–3572.

[70] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johnson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 264–280, 2009.

[71] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[72] S. Nag, G. Katz, and A. Saenz-Otero, "Collaborative gaming and competition for cs-stem education using spheres zero robotics," in *Acta Astronautica*, vol. 83, no. February-March, 2013, pp. 145–174.

[73] NASA, "IRG-FF029 Astrobee guest science guide," https://www.nasa.gov/sites/default/files/atoms/files/irg-ff029-astrobee-guest-science-guide.pdf, 2017, accessed: 2019-07-09.

[74] ——, "Astrobee robot software," https://github.com/nasa/astrobee, 2019, accessed: 2019-07-18.

[75] ——, "Astrobee publications record," https://www.nasa.gov/astrobee-research-publications/, 2024, accessed: 2024-07-03.

[76] S. Nolet, "The spheres navigation system: from early development to on-orbit testing," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007, p. 6354.

[77] J. Norheim, J. Hoffman, D. Newman, T. E. Cohen, D. S. Lees, M. C. Deans, and D. S. S. Lim, "Architecture of a surface exploration traverse analysis and navigational tool," in *Proc. IEEE Aerospace*, 2018.

[78] C. Oestreich *et al.*, "On-orbit inspection of an unknown, tumbling target using nasa's astrobee robotic free-flyers," in *Conference on Computer Vision and Pattern Recognition*, 2021.

[79] N. O. of the Chief Health & Medical Officer (OCHMO). Eclss technical brief. [Online]. Available: https://www.nasa.gov/wp-content/uploads/2023/07/eclss-technical-brief-ochmo.pdf

[80] I. W. Park, T. Smith, and J. F. Love, "Thermal design of Astrobee perching arm," in *IEEE/SICE Int. Symp. System Integration (SII)*, 2019, pp. 444–449.

[81] I.-W. Park, T. Smith, S. W. Wong, P. Piacenza, and M. Ciocarlie, "Developing a 3-DOF compliant perching arm for a free-flying robot on the International Space Station," in *Proc. IEEE Int. Conf. Adv. Int. Mech. (AIM)*, 2017.

[82] S. Pedrotty, J. Sullivan, E. A. Gambone, and T. Kirven, "Seeker free-flying inspector GNC system overview," in *Proc. American Astronautical Society Guidance Navigation & Control Conference*, 2019.

[83] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[84] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *Proc. Int. Conf. Robotics and Automation (ICRA) Workshop on Open Source Software*, 2009.

[85] M. Romano, "Astrobatic: A hopping-maneuver experiment for a spacecraft-manipulator system on board the International Space Station," in *Proc. International Aeronautical Congress (IAC)*, 2019.

[86] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1689–1696.

[87] V. Schröder, R. Regele, J. Sommer, T. Eisenberg, and C. Karrasch, "GNC system design for the Crew Interactive MObile companioN (CIMON)," in *Proc. Int. Aeronautical Congress (IAC)*, 2018.

[88] T. Smith, J. Barlow, M. Bualat, T. Fong, C. Provencher, H. Sanchez, E. Smith, *et al.*, "Astrobee: A new platform for free-flying robotics on the International Space Station," in *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2016.

[89] T. Smith, T. Cohen, D. Lees, and T. Scharff, "The Exploration Plan JSON (XPJSON) format specification," https://github.com/xgds/xgds_planner2/blob/master/xgds_planner2/xpjsonSpec/xpjson.rst, 2017, accessed: 2019-06-19.

[90] R. Soussan, V. Kumar, B. Coltin, and T. Smith, "Astroloc: An efficient and robust localizer for a free-flying robot," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 4106–4112.

[91] R. Soussan, M. Moreira, B. Coltin, and T. Smith, "Astroloc2: Fast sequential depth-enhanced localization for free-flying robots," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 14 595–14 602.

[92] C. Specht, A. Bishnoi, and R. Lampariello, "Autonomous spacecraft rendezvous using tube-based model predictive control: Design and application," *Journal of Guidance, Control, and Dynamics*, vol. 46, no. 7, 2023.

[93] J. M. Storey, D. Kirk, B. Marsell, and P. Schallhorn, "Progress towards a microgravity CFD validation study using the ISS SPHERES-SLOSH experiment," in *Proc. AIAA/SAE/ASEE Joint Propulsion Conf.*, 2017.

[94] D. Szafir, B. Mutlu, and T. Fong, "Communicating directionality in flying robots," in *Proc. ACM Conf. on Human-Robot Interaction (HRI)*, 2015.

[95] N. Tanishima, S. Mitani, S. Shigeto, Y. Matsumoto, Y. Arai, M. Goto, and S. Suzuki, "Effective and accurate method for ground and on-orbit verification of control systems for free-flying robot with low thrust force," in *Proc. Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2018.

[96] V. Usenko, N. Demmel, D. Schubert, J. Stueckler, and D. Cremers, "Visual-inertial mapping with non-linear factor recovery," *IEEE Robotics and Automation Letters (RA-L) & Int. Conference on Intelligent Robotics and Automation (ICRA)*, vol. 5, no. 2, pp. 422–429, 2020.

[97] H. Utz, "The Robot Application Programming Interface Delegate project," http://robotapi.sourceforge.net/index.html, 2012, accessed: 2019-07-03.

[98] A. M. Vargas, J. Benavides, J. Barlow, H. Orosco, S. Doi, R. G. Ruiz, R. Carlino, J. Cortez, A. Katterhagen, S. Kanis, B. Coltin, R. Soussan, and K. Hamilton, "Astrobee's multi-year activities at the international space station's japanese experimental module," in *Proc. Int. Aeronautical Congress (IAC)*, 2022.

[99] A. M. Vargas, R. G. Ruiz, P. Wofford, V. Kumar, B. V. Ross, A. Katterhagen, J. Barlow, L. Flückiger, J. Benavides, T. Smith, and M. Bualat, "Astrobee: Current status and future use as an international research platform," in *Proc. Int. Aeronautical Congress (IAC)*, 2019.

[100] L. Von Stumberg, V. Usenko, and D. Cremers, "Direct sparse visual-inertial odometry using dynamic marginalization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2510–2517.

[101] M. Watterson, T. Smith, and V. Kumar, "Smooth trajectory generation on se (3) for a free flying space robot," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 5459–5466.

[102] ——, "Smooth trajectory generation on SE(3) for a free flying space robot," in *Proc. IEEE Int. Conf. Rob. Sys. (IROS)*, 2016.

[103] S. P. Yamaguchi, A. M. Vargas, T. Eisenberg, C. Rogon, T. Yamamot, S. Inoue, C. Kössl, B. Coltin, T. Smith, and J. V. Benavides, "Free-flying crew cooperative robots on the iss: A joint review of astrobee, cimon, and int-ball operations," in *International Conference on Space Robotics (iSpaRo)*. Sendai, Japan: Institute of Electrical and Electronics Engineers (IEEE), 1st-4th December 2025. [Online]. Available: https://2025.ieee-icra.org/event/enhancing-dexterity-in-space-environments-the-potential-of-soft-robotic-systems/