Localization from Visual Landmarks on a Free-flying Robot

Brian Coltin[1], Jesse Fusco[2], Zack Moratto[1], Oleg Alexandrov[1], and Robert Nakamura[2]

***Abstract*— We present the localization approach for Astrobee, a new free-flying robot designed to navigate autonomously on the International Space Station (ISS). Astrobee will accommodate a variety of payloads and enable guest scientists to run experiments in zero-g, as well as assist astronauts and ground controllers. Astrobee will replace the SPHERES robots which currently operate on the ISS, whose use of fixed ultrasonic beacons for localization limits them to work in a 2 meter cube. Astrobee localizes with monocular vision and an IMU, without any environmental modifications. Visual features detected on a pre-built map, optical flow information, and IMU readings are all integrated into an extended Kalman filter (EKF) to estimate the robot pose. We introduce several modifications to the filter to make it more robust to noise, and extensively evaluate the localization algorithm.**

I. Introduction

The free-flying SPHERES robots currently operate on board the International Space Station (ISS), where they are used to conduct a wide variety of experiments in microgravity. The SPHERES are one of the most popular projects on the ISS. However, the SPHERES have two major limitations: 1) they operate with gas thrusters and non-rechargeable batteries, requiring a regular upmass of consumable batteries and CO_2 canisters [1]; and 2) the robots localize by triangulating their positions from fixed ultrasonic beacons, restricting their operating range to a 2m cube [2] (although SPHERES payloads have enabled them to localize with stereo vision [3] and Google's Project Tango).

The Astrobee robots are being developed to address these limitations. Astrobee will use electric fans for propulsion, powered by batteries that recharge through a dock. Second, rather than localizing with ultrasonic beacons, or any form of fixed infrastructure, Astrobee will localize with visual features. The current Astrobee prototype is shown in Figure 1. See [4] for details about Astrobee's hardware, including computing capabilities.

We present Astrobee's complete localization approach. Beforehand, a map of the space station is built from visual features. As Astrobee moves, it detects a variety of observations to help localize: visual features from the map, AR tags, handrail measurements from a depth sensor, and optical flow features. It integrates all these measurements with the IMU into an augmented state Extended Kalman Filter (EKF) that estimates the robot's pose. We extensively evaluate Astrobee's localization approach, both in a 2D testing environment and in 3D through a building.

[1]The authors are with SGT, Inc.[1] and NASA[2], NASA Ames Research Center, Moffett Field, CA, 94035, USA {brian.j.coltin, jesse.c.fusco, zachary.m.moratto, oleg.alexandrov, robert.h.nakamura}@nasa.gov

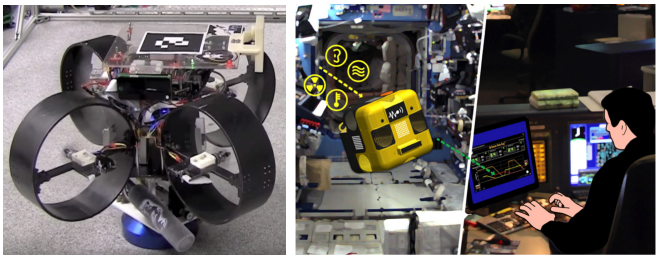

Fig. 1. *Left*: The Astrobee prototype, gliding on a smooth granite table using pressurized gas (a reverse air hockey table). *Right*: Artist's conception of Astrobee flying through the ISS, monitored by ground controllers.

While the fundamental algorithms behind our localization approach are well known, Astrobee presents special challenges as it will be operated in space by people other than its creators. When operating on the ISS for hours at a time, errors such as losing the position fix or colliding with a wall may require assistance from astronauts. Since astronaut time is an extremely valuable and scarce resource, Astrobee must function reliably, or it will simply not be used. Hence, we have tested Astrobee in increasingly extreme conditions—moving and rotating many times faster than its maximum speed, with changing lighting conditions, with numerous occlusions, in environments that differ highly from its map, etc. In this process, we have learned many lessons and made a number of modifications to the localization algorithm to increase robustness and reliability of localization.

II. Background and Related Work

Our localization approach is based on an augmented state EKF initially developed for spacecraft descent and landing [5]. Astrobee matches monocular visual features to a pre-built map, and fuses these observations with optical flow, accelerometer readings, and gyrometer readings. The visual features are replaced with AR tags as Astrobee approaches its dock, and handrail measurements from a depth sensor when Astrobee perches on handrails. Astrobee is equipped with other sensing modalities that have been used for localization, namely WiFi [6] and 3D depth sensors [7]; however, WiFi with our COTS radio and the existing ISS access point geometry does not provide sufficient position accuracy [6]. We chose to use visual features rather than 3D depth data (except for handrails) because visual features can provide a global pose estimate from a single camera frame, which makes recovery from position tracker failures simple.

The localization system consists of three components:

1) **Map Building**. Visual features (i.e., SURF [8], BRISK [9]) are detected and matched across a collection of

images, then their 3D positions are jointly optimized using structure from motion. This approach was most famously applied in [10], but many variants of this fundamental approach exist such as the open source Theia [11] and openMVG [12] libraries.

2) **Visual Observations**. To localize, the robot detects features on an image, matches them to the pre-built map, then filters the matches with a geometric consistency check (such as RANSAC [13]). Much work has gone into faster and better features, such as BRISK [9] and ORB [14]. There has also been work on quickly matching features to images in a map by searching for similar images [10], [15]. Optical flow is also used for velocity estimation; features are tracked locally between frames to provide a velocity estimate [16].
3) **Position Tracking**. All visual observations are integrated with the linear acceleration and angular velocity from the IMU. A Kalman filter is often used for this purpose, of which there are numerous variants, such as adaptive EKFs [17] and unscented Kalman filters [18]. Astrobee uses an augmented state indirect EKF initially designed for spacecraft descent and landing [5]. This filter uses augmented states to keep track of the state when an image was taken to account for the lengthy delay in processing the image. The most popular alternative to an EKF for localization is the particle filter [19], which could be applied to Astrobee localization in combination with sensor resetting for global localization [20]. However, particle filters would scale poorly in computation for Astrobee since more particles would be required to track its six degree of freedom pose compared to the three for ground robots.

We intentionally separate mapping from localization. While approaches for simultaneous localization and mapping (SLAM) such as LSD-SLAM [21] are increasingly capable, they are not as reliable as techniques which rely on a fixed, pre-computed map. Astrobee is confined to a fixed area so the flexibility that SLAM provides is unnecessary.

III. Localization for Astrobee

We discuss the entire localization system on Astrobee in detail. We particularly emphasize our own novel contributions that enable reliable localization with limited computation and minimal human intervention.

A. Mapping

Astrobee localizes based on a sparse map of visual features constructed offline. A sparse map $M = (F, P)$ consists of a list of visual feature descriptors F_i with associated 3D positions P_i. Ultimately, the robot localizes by detecting visual features, matching them to features in F, and triangulating its own position based on the associated feature positions in P. Astrobee's sparse map building process is as follows:

1) **Collect Images**. We begin with a sequence of images I that are processed offline, on the ground. Initially these images will be collected by an astronaut, but once an initial map is built, the robot can collect new images autonomously. The map will need to be updated occasionally as supplies and equipment are moved frequently on the ISS. The pattern of motion is critical for map-building— views from different parts and angles of the ISS are essential to localize from anywhere. We have found that rotating in all directions while moving slowly down a corridor is most effective.
2) **SURF Feature Detection**: Given the images I_i, we detect a list of SURF feature [8] descriptors $F_{i,j}$ at pixel coordinates $L_{i,j}$. We use SURF features because they are high quality and although they are moderately expensive to compute, map building is offline so computational cost is less important.
3) **Feature Matching**: For every pair of images I_i and I_j, we match the feature descriptors in F_i to those in F_j using an approximate nearest neighbors algorithm [22] to generate a partial matching between feature descriptors in the two images. From this matching we estimate the essential matrix $E_{i,j}$ transforming from matched coordinates in L_i to L_j that minimizes the epipolar distance error using AC-RANSAC [23], discarding outlier matches.
4) **Track Building**: We fuse the pairwise matches into multiview correspondences, or tracks [24], resulting in a set T of tracks, where a track is a set of matching feature images and image coordinates which are believed to refer to the same physical landmark.
5) **Initial Map Guess**: From each $E_{i,i+1}$, we estimate a rotation $R_{i,i+1}$ and translation $T_{i,i+1}$ between the two subsequent cameras [25]. Given these rotations and translations, we estimate global camera poses C_i for each image. Then, using multi-view triangulation, we estimate a global position P_j for each track $T_j \in T$.
6) **Incremental Bundle Adjustment**: We refine the C_i and P_j using incremental bundle adjustment to minimize the reprojection error of all the points. Bundle adjustment is performed first on non-overlapping groups of four images, then repeating in powers of two up to 128 images at a time. This step requires that the images are processed sequentially according to the

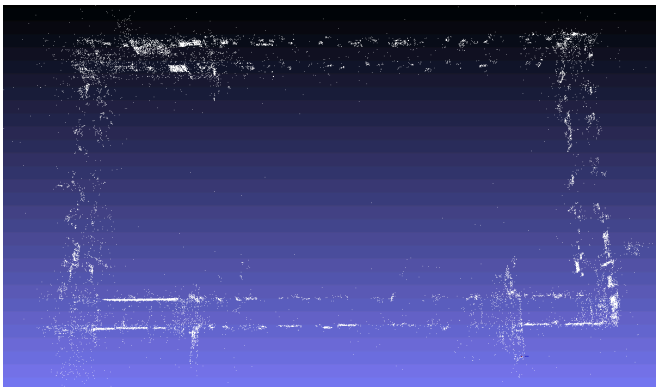

Fig. 2. A closed sparse map built from moving Astrobee in a loop through the hallways of a building. Shown are the 3D coordinates of the BRISK features that compose the final map (right).

robot's motion. Incremental bundle adjustment ensures that the maps are locally consistent.

7) **Global Bundle Adjustment**: The same process is repeated with every image, starting with the C_i and P_j from incremental bundle adjustment. Any loops in which the position drifts will be closed in this step.
8) **Rebuilding with BRISK Features**: The current sparse map uses SURF features. We aim to use BRISK features [9], which are less accurate and robust than SURF (which is critical for map building) but also much faster (which is critical for localization). As such, we rebuild the map using BRISK features. We repeat the detection, matching, track building and global bundle adjustment steps with BRISK features, except we use the camera transforms C_i previously computed from the SURF features. This allows us to combine the accuracy and robustness of a SURF-based map-buildling process with the speed of BRISK feature localization. We also reuse the previous information about which pairs of images will contain matches, eliminating the need for an exhaustive search. The final BRISK map has the same highly accurate camera poses as the initial SURF map, but with faster BRISK features instead of SURF features. To our knowledge this rebuilding step is unique to our approach.
9) **Registration**: A consistent map has been created, but it is in an arbitrary coordinate frame. We take a number of known points on the ISS and find the affine transform which brings these points into the desired ISS coordinate frame. Then we perform global bundle adjustment again with the registration points fixed.
10) **Bag of Words Database**: It is possible to localize by comparing features to every other image in the map, but this is very slow. Instead, we construct a hierarchical database of bag of words features for images [15]. This allows us to quickly look up the most similar images to a given image, so that we can only match features against the most similar images.

The final BRISK map and database allow Astrobee to localize quickly. See Figure 2 for an example map.

B. Visual Observations

Next, we discuss how visual features are observed and filtered before being sent to the EKF for integration into the position estimate. We use four types of visual features: sparse map features, AR tags, handrail features, and optical flow features. See Table I for an overview of all the inputs to the localization process. Optical flow is always used, but only one of the other visual feature types is used at a time.

1) Sparse Map Features: BRISK features are detected in the image. The bag of words database is queried to find the most similar images in the sparse map. Then we match the BRISK descriptors in the query image to the similar map images. A list of potential matches from image coordinates to global ISS coordinates is generated from the sparse map.

Next, RANSAC is applied to remove outliers. A random subset of 4 landmarks is selected, and is used to generate a hypothesis camera pose with P3P [26]. We check the consistency of the matched features with the hypothesis camera pose. This process is repeated multiple times with different random landmark selections, and the camera pose with the highest number of inlier features is selected. The inlier observations— both 2D camera coordinates and matching 3D landmark positions— are sent as inputs to the EKF.

Name	Rate (Hz)	Value
Sparse Map	—	BRISK descriptors and positions
AR Tag Map	—	AR tag IDs and corner positions
IMU Acceleration	62.5	Linear acceleration a_{imu}
IMU Angular Vel.	62.5	Angular velocity ω_{imu}
Sparse Map Features	≈ 2	Coordinates in image and map
AR Tag Features	≈ 5	Coordinates in image and map
Handrail Features	≈ 5	Depth image and global positions
Optical Flow	≈ 5	Multiple image coordinates

TABLE I
INPUTS TO ASTROBEE LOCALIZATION.

2) AR Tags: For even higher accuracy and reliability, Astrobee can localize based on AR tags. Astrobee switches to use AR tags when docking as the dock is tagged. The detected corners of the AR tag are sent to the EKF in the same manner as sparse map landmarks. We use the ALVAR library [27].

3) Handrail Measurements: The Astrobee has an arm that can grip handrails scattered throughout the ISS to perch and observe station activities. The locations of the handrails change frequently and are not known beforehand. A ground operator will direct the Astrobee to a location where a handrail is visible, and then Astrobee will dock autonomously. A time of flight sensor detects the handrail by fitting the plane of the wall with RANSAC and then fitting points to the cylinder of the handrail in front of that plane. A selection of points on the handrail and wall, as well as their associated global positions (relative to the initial EKF pose when the handrail was first detected) are sent to the EKF.

4) Optical Flow: A list of 50 optical flow features is maintained at all times. These features are generated using Good Features to Track [28], which is speedy and effective for optical flow. For each frame, starting with the previous feature coordinates, the features are tracked with pyramidal Lucas Kanade [29]. One trick to remove outliers is to apply the tracking backwards from the new estimated feature coordinates— if the backwards tracked position does not match the original, the feature is removed. We also delete features that are close to the border of the image, as this results in many features which leave the image being incorrectly tracked at the border. We record feature coordinates over multiple frames. Once a feature has been tracked across four frames, all four coordinates are sent to the EKF.

C. Extended Kalman Filter

Astrobee integrates its visual observations and IMU readings into an Extended Kalman Filter (EKF). Our EKF is largely based on the approach in [5], but with key extensions

that make it more robust to errors in map building and camera image acquisition timing, and incorporate additional inputs.

1) Overview: The state vector estimated by the EKF is

$$x = \left[{}^B_G q \; b_g \; {}^G v_B \; b_a \; {}^G p_B \; ({}^{C_1}_G q \; {}^G p_{C_1}) \dots ({}^{C_n}_G q \; {}^G p_{C_n}) \right]$$

where ${}^B_G q$ is the quaternion representing the robot body in the global frame, b_g is the gyrometer bias, ${}^G v_B$ is the robot body's velocity in the global frame, b_a is the accelerometer bias, and ${}^G p_B$ is the robot body's position in the global frame. The ${}^{C_i}_G q$ are the rotation quaternions and the ${}^G p_{C_i}$ are the positions of the camera forming the augmented states. The filter stores these augmented states when an image is taken and uses them once the image is processed to account for the lengthy processing time. We store five augmented states, one for mapped landmarks / AR tags / handrails, and four for optical flow. The EKF also maintains a covariance P of the error state.

We briefly outline the EKF (see [5] for full details).

1) **Predict Step**: The state estimate x and covariance P are propagated forward in time based on the measured IMU acceleration a_{imu} and angular velocity ω_{imu}. The predict step runs at the same rate as the IMU.
2) **State Augmentation**: When a new camera image is taken for processing, the appropriate augmented state is updated. The first augmented state is for sparse map landmarks, AR tags, and handrails (only one can be used at a time, which reduces the state size and required computation), and four states are maintained for optical flow. When a new augmented state is added, the appropriate entries in P are also updated.
3) **Update Step**: The visual measurements are applied to update the state and covariance. Let $\mathbf{z}$ be all the measurements (in pixel coordinates) and $\hat{\mathbf{z}}$ be the expected measurements, given the estimated state $\mathbf{x}$. Then we linearize the residual $\mathbf{r}$ in terms of the error state $\tilde{\mathbf{x}}$ and a Jacobian H

$$\mathbf{r} = \mathbf{z} - \hat{\mathbf{z}} \approx \mathbf{H}\tilde{\mathbf{x}} + \mathbf{n}$$

where $\mathbf{n}$ is a noise vector with covariance R. See [5] for full details of how $\mathbf{r}$ and H are computed for both sparse map landmarks and optical flow landmarks. In the original formulation of the filter, it is assumed that every element in the residual is independent and identically distributed, and hence that $\mathbf{n}$ has a constant diagonal covariance $R = \text{diag}\left(\sigma^2\right)$.

2) Extensions and Discussion: Next, we detail key changes we made to the EKF for Astrobee, and discuss insights on making the most effective and robust filter.

Registration Timing Errors: In the update step, R models the expected observation error. However, in [5], R only considers a constant error in pixels for all observations. In particular, the original approach assumes the registration pulse is received exactly when the image is taken. Without dedicated hardware, this is impractical— image acquisition itself is not instantaneous, and there is a lengthy pipeline from the camera hardware to the camera driver to the user application. For Astrobee, the pipeline is even longer as the EKF runs on a separate processor from the image processing code, connected over Ethernet. Hence, we modify R based on the robot's current estimate of its linear and angular velocity, as the timing errors have larger impacts at higher velocities.

Let T_G^C be the affine transform from the global coordinate frame to the camera coordinate frame, $A(\mathbf{v}, \epsilon)$ be the affine transform that rotates by Euler angles ϵ and translates by $\mathbf{v}$, Δt be the estimated registration delay in seconds, and $\mathbf{v}_{aug}$ and ϵ_{aug} be the estimated linear and angular velocities when the augmented state was stored. Let L be the set of global coordinates of the detected sparse map features. For each subvector $\mathbf{r}_i = \mathbf{z}_i - \hat{\mathbf{z}}_i$ of $\mathbf{r}$, we have

$$\hat{\mathbf{z}}_i = \frac{1}{\hat{\mathbf{c}}_{i,3}} \begin{bmatrix} \hat{\mathbf{c}}_{i,1} \\ \hat{\mathbf{c}}_{i,2} \end{bmatrix}, \; \hat{\mathbf{c}}_i = T_G^C \begin{bmatrix} L_i \\ 1 \end{bmatrix} \text{ [5].}$$

Then, we compute the expected registration error σ_{reg}.

$$\sigma_{reg,i} = \hat{\mathbf{z}}_i - \frac{1}{\mathbf{n}_{i,3}} \begin{bmatrix} \mathbf{n}_{i,1} \\ \mathbf{n}_{i,2} \end{bmatrix}, \; \mathbf{n}_i = A(\Delta t \mathbf{v}_{aug}, \Delta t \epsilon_{aug}) \hat{\mathbf{c}}_i$$

We combine σ_{reg} with the original constant σ_{const} by setting $\sigma = \sigma_{reg} + \sigma_{const}$. This same addition to the covariance is also applied to optical flow, except with different camera matrices and velocities for each augmented state, and using the estimated 3D points.

Map Building Errors: Another significant source of error is the map-building process, which in the case of Astrobee involves centimeter-level errors. If the sparse map feature positions P_i have errors, the error in camera coordinates (which r is measured in) increases as the camera moves closer to the features. We add another term σ_{map} to R to account for this, where $\sigma_{map,i} = K_{map}/\hat{\mathbf{c}}_{i,3}$. K_{map} is a constant, and the final covariance $R = \text{diag}(\sigma^2)$ is constructed with all three terms, $\sigma = \sigma_{map} + \sigma_{reg} + \sigma_{const}$. This source of error is most important when the camera is very close to the landmarks.

Handrail Localization: We extended the EKF to localize based on points from a depth sensor. To keep the process as close as possible to the update from sparse map landmarks, a collection of depth sensor observations and their expected global positions are input to the EKF. These points are located both on the handrail itself and on the wall behind the handrail. The EKF update step is largely the same as for sparse map landmarks, except the residual includes three dimensions rather than two. The main challenge is that as the robot approaches close to the handrail and the two handrail bar endpoints leave the depth camera field of view, the handrail detection algorithm cannot determine the relative pose of the robot along the axis of the handrail bar. When this occurs, the global direction of the handrail bar is passed as an input to the EKF, the residuals are rotated into a coordinate frame where the z axis is parallel to the handrail bar, and then the z coordinates are dropped from the residuals.

Outlier Removal: We experimented with further outlier removal within the EKF, in addition to RANSAC for the sparse map landmarks and the backwards optical flow check. Specifically, we checked the Mahalanobis distance of the

feature inputs from the expected distribution represented by x and P. While this check occasionally removes outliers successfully, it would also remove correct observations when the filter had a significant position error, so we removed it.

EKF Initialization: One key advantage of using visual landmark localization (as opposed to a LIDAR or depth sensor) is that a 3D pose can be directly computed from the observed landmarks. We do so using RANSAC, as discussed previously, and this pose is used to initialize the EKF.

Failure Recovery: The same approach is used to reinitialize the EKF when it inevitably (albeit rarely) fails and the robot becomes lost. The robot is determined to be lost when the sum of the covariance diagonal exceeds a threshold value. An intermediate "Uncertain" state is entered when the filter does not update successfully with a sparse map landmark update in a given time.

IMU Bias Initialization: The IMU bias drifts slowly through a process that can be modeled with a random walk. While the robot is running, the EKF does a good job at tracking this bias drift. However, the bias drifts when the robot is not running, and over weeks or months the drift is large enough that the EKF has difficulty recovering. Hence, we institute a bias initialization procedure, where the robot averages its IMU measurements over five seconds of remaining stationary in the dock to measure the bias. This only needs to run occasionally, but greatly aids in EKF initialization.

Centrifugal and Euler Accelerations: The original EKF formulation assumes that the IMU is located at the robot's center of rotation. However, for Astrobee this is not the case, and the IMU detects Euler and centrifugal accelerations. The total estimated linear acceleration $\hat{a}$ is computed as

$$\hat{a} = a_{imu} - b_a - \frac{d\omega_{imu}}{dt} \times r - \omega_{imu} \times (\omega_{imu} \times r)$$

where b_a is the estimated bias, r is the vector from the center of rotation to the IMU and ω_{imu} is the angular velocity vector in radians. The Euler acceleration $\frac{d\omega_{imu}}{dt} \times r$ is a reaction to the angular acceleration of the vehicle, and the term $\omega \times (\omega \times r)$ is the centrifugal force. Due to these additional forces, properly measuring the vector r is critical. The robot's position begins to drift away when rotating if r is not correct to millimeter precision.

Constant Selection: The filter is highly dependent on the choice of numerous constants, particularly the σ noise values and the noise matrix Q_{IMU} from the predict step. Unfortunately we have no better way of tuning these values than trial and error.

IV. Experimental Results

We present a number of experiments showcasing the effectiveness of our localization approach.

First, we conducted studies on the 2D granite table in our lab (see Figure 1). While the robot is physically constrained to three degrees of freedom, we run the full six degree of freedom EKF. We recorded test runs from manually pushing the robot around, recording specific movements: forward and backwards translations, sideways translations, spinning in place in both directions, and moving in a circle around the table facing outwards in both directions. For all these basic tests, we first moved the robot at a normal operating speed, and then moved it at a much higher speed, the maximum speed at which the robot could be safely controlled. Finally, we did two tests combining all the primitive motions, one at the slow speed and one at the fast speed.

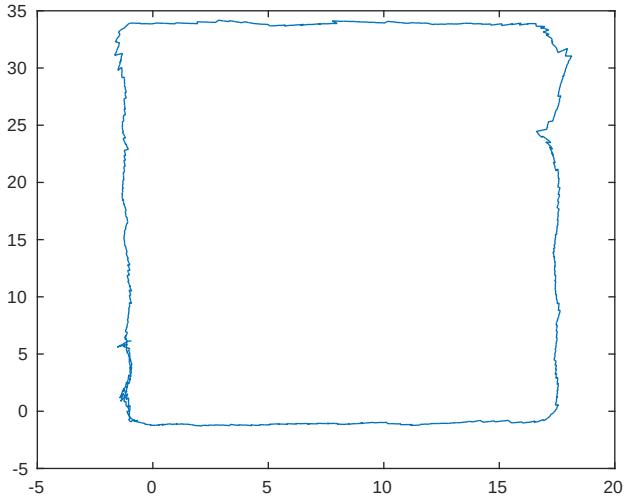

Fig. 3. The estimated path output by the EKF from Astrobee moving in a rectangle around the hallways of a building (units are in meters). The map Astrobee localized on is shown in Figure 2.

For all the tests, we recorded the ground truth position and orientation with an overhead camera and an AR tag on the robot. We ran the EKF and computed the root mean squared error of position and orientation. We tested with and without our modifications to the R matrix, for both σ_{reg} and σ_{map}, on the same logged data. The results are shown in Table II.

The changes show a modest improvement in positional accuracy for most of the test cases. For the spin in place test, there is a large improvement of 4 cm. This is due to σ_{reg}, which is critical when performing high speed rotations. The angular errors for both approaches are similar.

To test our system over longer distances, we also built a map of the hallways of our entire building (see Figure 2), then carried Astrobee through. The trace of its path as estimated by the EKF is shown in Figure 3. It was able to maintain its position and knew it had returned to its starting position. This test was particularly challenging due to the variations in lighting throughout the building— some areas are quite dark, while others have bright, direct natural light.

We have also built confidence through months of testing in our ability to handle unexpected landmarks or landmarks that have moved. We have tested both map building and localization successfully with humans in the field of view. Furthermore, while testing in a bustling, busy lab, with objects constantly moving, we were able to localize successfully using the exact same map for three months without issue. This speaks well to Astrobee's ability to handle the busy, constantly changing environment of the ISS.

Algorithm	Measurement	Forward	Sideways	Spin	Circle	All Slow	All Fast
$\sigma = \sigma_{const}$	Position RMSE (cm)	7.87	11.99	11.42	6.46	7.18	18.04
	Angular RMSE (°)	2.76	1.55	4.82	3.79	5.66	14.10
$\sigma = \sigma_{const} + \sigma_{map} + \sigma_{reg}$	Position RMSE (cm)	7.93	11.15	7.33	4.94	6.02	16.53
	Angular RMSE (°)	1.33	1.56	4.83	3.81	5.90	14.07

TABLE II

EXPERIMENTAL RESULTS COMPARING R MODIFIED TO INCLUDE MAP AND REGISTRATION TIMING ERRORS.

The video submission associated with this paper shows a variety of other testing conditions, including localization and docking while the robot is self-propelled.[1]

V. CONCLUSIONS AND FUTURE WORK

We have presented a complete localization system for Astrobee, including mapping, feature detection, and position tracking, and demonstrated its initial success in the lab and a larger environment the size of a building. Astrobee's localization system features a number of novel features, such as improved error modeling and a map rebuilding process that changes feature descriptors. However, a great deal more work needs to be done before Astrobee is ready to serve on the ISS. In particular, we need to test mapping and localization when moving freely in three dimensions. A gantry system is currently being constructed which will allow Astrobee to move with six degrees of freedom on Earth. We plan to incorporate further sensors, such as a 3D depth sensor and specialized optical flow camera, that will further improve Astrobee's localization and allow Astrobee to operate, or at least maintain its position, even when landmark-based localization temporarily fails.

ACKNOWLEDGEMENTS

We would like to thank the Astrobee engineering team and the NASA Human Exploration Telerobotics 2 project for supporting this work. The NASA Game Changing Development Program (Space Technology Mission Directorate) and ISS SPHERES Facility (Human Exploration and Operations Mission Directorate) provided funding for this work.

REFERENCES

[1] J. Enright, M. Hilstad, A. Saenz-Otero, and D. Miller, "The SPHERES guest scientist program: Collaborative science on the ISS," in *Proc. of IEEE Aerospace Conference*, 2004.

[2] S. Nolet, "The SPHERES navigation system: from early development to on-orbit testing," in *Proc. of AIAA Guidance, Navigation and Control Conference*, 2007.

[3] B. E. Tweddle, T. P. Setterfield, A. Saenz-Otero, and D. W. Miller, "An open research facility for vision-based navigation onboard the international space station," *Journal of Field Robotics*, 2015.

[4] J. Barlow, E. Smith, T. Smith, M. Bualat, T. Fong, C. Provencher, and H. Sanchez, "Astrobee: A new platform for free-flying robotics on the international space station," in *Proc. of Int. Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 2016.

[5] 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.

[6] J. Yoo, T. Kim, C. Provencher, and T. Fong, "WiFi localization on the international space station," in *IEEE Symposium on Intelligent Embedded Systems*. IEEE, 2014.

[7] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *Proc. of ICRA*. IEEE, 2012.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[9] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *Proc. of ICCV*. IEEE, 2011.

[10] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building Rome in a day," *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[11] C. Sweeney, *Theia Multiview Geometry Library: Tutorial & Reference*, University of California Santa Barbara.

[12] P. Moulon, P. Monasse, and R. Marlet, "Global fusion of relative motions for robust, accurate and scalable structure from motion," in *Proc. of ICCV*. IEEE, 2013.

[13] W. Zhang and J. Kosecka, "Image based localization in urban environments," in *Int. Symposium on 3D Data Processing, Visualization, and Transmission*, 2006.

[14] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Proc. of ICCV*. IEEE, 2011.

[15] D. Galvez-Lopez 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, October 2012.

[16] B. K. Horn and B. G. Schunck, "Determining optical flow," in *1981 Technical Symposium East*. International Society for Optics and Photonics, 1981, pp. 319–331.

[17] L. Jetto, S. Longhi, and G. Venturini, "Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 219–229, 1999.

[18] M. St-Pierre and D. Gingras, "Comparison between the unscented Kalman filter and the extended Kalman filter for the position estimation module of an integrated navigation information system," in *Proc. of IEEE Intelligent Vehicles Symposium*, 2004.

[19] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proc. of ICRA*. IEEE, 1999.

[20] B. Coltin and M. Veloso, "Multi-observation sensor resetting localization with ambiguous landmarks," *Autonomous Robots*, vol. 35, no. 2-3, pp. 221–237, 2013.

[21] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. of European Conf. on Computer Vision*. Springer, 2014.

[22] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." in *Proc. of VISAPP*, 2009.

[23] L. Moisan, P. Moulon, and P. Monasse, "Automatic homographic registration of a pair of images, with a contrario elimination of outliers," *Image Processing On Line*, vol. 2, pp. 56–73, 2012.

[24] P. Moulon and P. Monasse, "Unordered feature tracking made fast and easy," in *Proc. of CVMP*, 2012.

[25] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[26] 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.

[27] VTT Technical Research Centre of Finland, "Alvar," 2016. [Online]. Available: http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html

[28] J. Shi and C. Tomasi, "Good features to track," in *Proc. of CVPR*. IEEE, 1994.

[29] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, 2001.

[1] Video available at `https://youtu.be/xakebgUMobo`.