HIGH-FIDELITY, CLOSED-LOOP SIMULATION OF SPACECRAFT VISION-BASED RELATIVE NAVIGATION IN ROS2

Kai Matsuka[*], Leo Zhang[†], Isabelle Ragheb[‡], Christine Ohenzuwa[§] and Soon-Jo Chung[¶]

The demand for increasingly complex robotics technology for space missions has spurred interest in using the Robotic Operating System (ROS) for free-flying, spacecraft formations in orbit. In this paper, we present a new set of integrated autonomy algorithms along with high-fidelity simulation tools for formation flying, rendezvous, and proximity operations all implemented within ROS2. The autonomy algorithm includes deep-CNN-based object detection, distributed estimation, and pointing and formation control pipelines for a cooperative inspection task by multiple observers. The ROS2 simulation engine leverages an open-source astrodynamics software framework to model environmental forces and actuators in high fidelity and to propagate the ground truth dynamics. We developed a lightweight, ROS2-compatible interface that preserves flexibility and computational efficiency while leveraging existing resources in ROS2. In addition, we use a new Neural Radiance Fields (NeRF)-based rending tool that we developed to rapidly generate novel images online. These synthetic images are used as inputs to validate the vision-based navigation algorithm. All the autonomy algorithms and simulation tools are validated in a closed-loop simulation within a ROS2 framework.

INTRODUCTION

Over the past decade, the autonomy of formation-flying spacecraft has advanced significantly, as demonstrated by some previous missions.[1,2] As new mission concepts for on-orbit inspection and servicing tasks demand increasing thresholds for performance and safety, the level of autonomy required in robotics and perception tasks also must increase. Higher levels of autonomy require significant increases in both hardware and algorithmic complexity, resulting in the necessity to leverage reliable, open-source, historically terrestrial, robotics technology to enable end-to-end autonomy on orbit with a reasonable budget.

In parallel with this, there has been a recent increased interest in using the Robotics Operating System 2 (ROS2) as a middleware suit in space applications. ROS2 has advanced capabilities for real-time software and distributed systems and has been widely adopted in the robotics industry.[3] ROS/ROS2 has been implemented in a select few applications such as a lunar rover (VIPER[3]),

[*]Graduate Student, Graduate Aerospace Laboratories of the California Institute of Technology, California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, USA.

[†]Undergraduate Student, Computer Science, California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, USA.

[‡]Undergraduate Student, Mechanical and Aerospace Engineering, California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, USA.

[§]Undergraduate Student, Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544, USA.

[¶]Bren Professor of Control and Dynamical Systems, California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, USA.

flying robotic experiment platforms inside ISS (Astrobee[4]), humanoid robot at ISS (Spacenaut[5]), and other future rendezvous and docking mission.[6] There is also a discussion of forking ROS2 for space applications, called Space ROS, in the hopes of joining the benefits of ROS2 and space sectors. There are multiple advantages to using ROS2 for developing and validating autonomy FSW in space missions involving proximity operations. First, ROS2 is open-source and community-driven, allowing developers to leverage plentiful resources available to the terrestrial robotics community, enabling the rapid, low-cost development of advanced, end-to-end autonomy for space. Second, ROS2's inherent modular design, extensive features for real-time systems, and distributed communication protocols are all well-suited for the application of cooperative multiple spacecraft missions in orbit. Therefore, ROS2 has a significant potential to be useful for future robotic missions in space including on-orbit inspection and servicing.

While the use of ROS2 for space application has gained momentum, the use of ROS/ROS2 in prior autonomous operations in planetary orbit missions has been limited. The validation required for on-orbit autonomy has unique challenges. Primarily, the ability to model astrodynamic effects with high fidelity is critically important, as this is closely intertwined with the performance of the guidance and control algorithm over the course of many orbits. In addition, information-rich sensor inputs such as those from a camera, LiDAR, radar, star-tracker, etc. must be accurately modeled. Finally, wireless communication between distributed agents must be handled. In previous missions, the development and validation of Guidance, Navigation, and Control (GNC) flight software for on-orbit, close-proximity operations relied on high-fidelity astrodynamics simulation tools, such as STK or other private simulation frameworks, as well as hardware experimental platforms, such as the smallsat testbed at JPL or Caltech's Spacecraft Simulator.[7] While these traditional spacecraft simulation frameworks are good at modeling astrodynamic effects with high fidelity, they mostly lack the capabilities to model information-rich sensor inputs. On the other hand, while ROS2 does have some existing tools developed to model sensors, ROS2 lacks high-fidelity astrodynamic simulation capabilities*. In order for autonomous GNC FSW to keep up with the increased threshold for performance and safety in on-orbit inspection and servicing missions, a simulation framework capable of modeling astrodynamic effects and information-rich sensor inputs with high-fidelity is necessary.

In this paper, we propose to develop a ROS2 framework for rendezvous and proximity operation (RPO). Our contributions are three-fold. First, we developed an RPO autonomy software stack in ROS2 where the servicing spacecraft is tasked to inspect the uncooperative target and maintain a formation. Second, we developed approaches for ROS2-compatible spacecraft dynamics and sensor simulation. And third, we tested both the autonomy stack and the simulator modules in closed-loop simulations considering an RPO mission. The autonomy software stack for on-orbit inspection tasks involves vision-based navigation, pointing and formation keeping control, and formation keeping. For spacecraft dynamics simulation, we develop ROS-Basilisk, a ROS2-compatible software interface for Basilisk.[8] Basilisk is an open-source, high-fidelity astrodynamics simulation framework and we design the interface such that we preserve the flexibility and computational efficiency of Basilisk. We also demonstrate ROS-NeRF, a camera simulation module based on Neural Radiance Fields [9] for rendering high-fidelity images from novel camera views. We demonstrate the autonomy stack and spacecraft simulation framework in ROS2 in an example mission involving multi-agent,

*This is not to say physics simulation frameworks do not already exist within ROS2. Physics simulation frameworks, such as Gazebo, do exist within ROS2. However, for the purposes of validating GNC FSW, existing ROS2 physics simulation frameworks are not sufficient.

autonomous PRO. The mission scenario models a chaser spacecraft inspecting an uncooperative target using vision-based relative navigation. We validate the performance of the FSW using the ROS-Basilisk and ROS-NeRF in a closed-loop formation flying scenario using vision-based navigation and control.

The rest of the paper is organized as follows. The first section discusses the high-level architecture, spacecraft dynamics, and camera simulation modules. The next section discusses the autonomous software stack for on-orbit inspection tasks involving vision-based navigation and control. The following section discusses the experiment results in simulations. Finally, the last section makes concluding remarks.

HIGH-LEVEL ARCHITECTURE OF SIMULATION AND AUTONOMY SOFTWARE

The high-level architecture of the simulation and autonomy software modules are shown in Figure 1. The spacecraft dynamics and actuators of the spacecraft are modeled using ROS-Basilisk. The ground truth simulation states are published as ROS2 messages and they are subscribed by sensor simulation modules such as ROS-NeRF, star tracker, and GPS simulation modules. ROS-NeRF publishes RGB images as ROS messages it effectively models a visible camera. The images and other sensor outputs are subscribed by the vision-based navigation module, which applies object detection and Extended Kalman Filtering to track the 3D position of the uncooperative target. Based on the tracked 3D position of the uncooperative spacecraft, the servicing spacecraft tracks plans pointing and formation-keeping maneuvers that visually track the target and maintain the formation. The thruster and reaction wheel commands are published as ROS2 message, which is subscribed by ROS-Basilisk. Basilisk simulation applies the control based on the ROS2 message, closing the loop. Each component is a self-contained ROS module interacting with each other through the ROS2 messaging protocol. In the next sections, ROS-Basilisk, ROS-NeRF, and autonomy software stack are described in more detail.

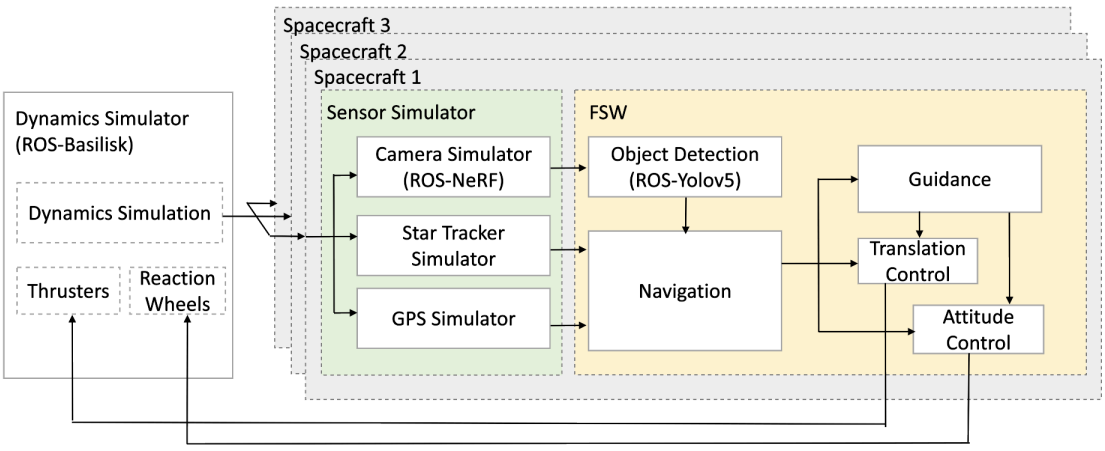

Figure 1. FSW pipeline for close-proximity operation using ROS2. Boxes with a solid outline are implemented as ROS2 modules.

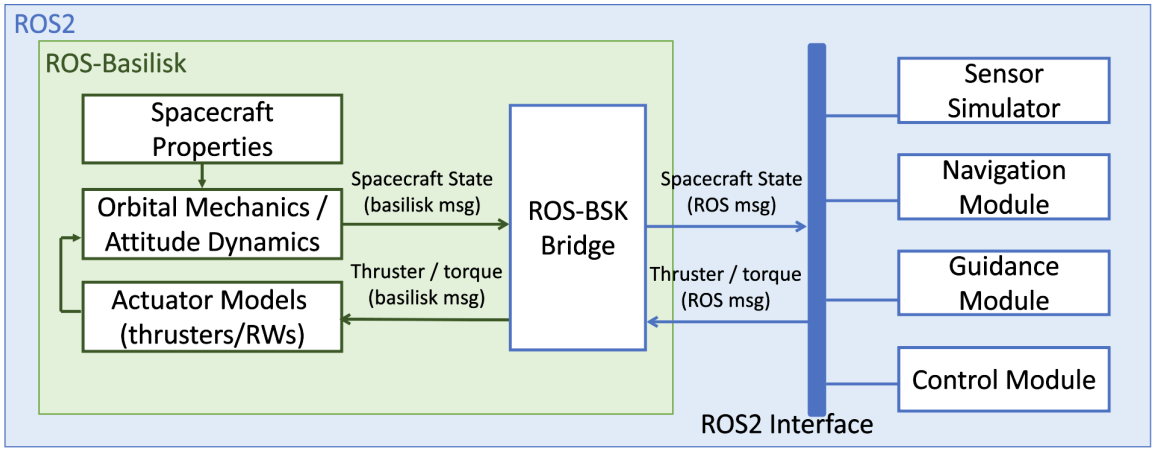

Figure 2. Diagram of ROS-Basilisk interacting with other modules via ROS2 messaging.

ROS-BASILISK: ROS2-COMPATIBLE SPACECRAFT DYNAMICS SIMULATION

This section discusses a software layer that bridges ROS2 and Basilisk,[8] the open-source astrodynamics simulation framework. Basilisk features a large library of high-fidelity force models, hardware components, visualization tools, an efficient back-end implementation in C/C++, and a user-friendly Python interface. Basilisk can be used to simulate space mission scenarios with G&C algorithms in closed-loop for missions such as formation flying.[10]

To leverage the advantages of both of these frameworks, ROS2 and Basilisk must run simultaneously and respective modules need to interact with each other. To accomplish this, we developed software that interfaces between ROS2 and Basilisk called ROS-Basilisk. At a high level, ROS-Basilisk is implemented as a timer-based ROS2 callback that propagates a discrete, fixed-time step of Basilisk simulation at each time step. From the ROS2 framework perspective, ROS-Basilisk regularly publishes the ground truth simulated states as ROS2 messages and optionally subscribes to control command messages, such as thruster burn time and reaction wheel torque commands, from other FSW algorithms implemented in ROS2 modules. Outside ROS-Basilisk, other ROS2 modules simulate sensor observations and run navigation and control algorithms. Within ROS-Basilisk, ROS2 and Basilisk messages are translated from one to another. From Basilisk's perspective, there exists a custom Basilisk module called ROS-BSK Bridge which sends control command messages and receives ground truth state messages. At each iteration, using the commands from the ROS-BSK Bridge, the rest of the Basilisk modules simulate actuators and propagate the spacecraft dynamics. The interaction between ROS2 and Basilisk can be seen in the diagram in Figure 2.

We also need to synchronize the timestamps between Basilisk and ROS2 models. Since ROS-Basilisk is a timer-based node that runs at a fixed rate, we also specify Basilisk to propagate by the same time step so that we can roughly align the two simulations. However, the exact time stamps may not align precisely due to the delays between ROS2 messages. We tackle the synchronization challenge by appending each ROS2 message with a time stamp from the Basilisk simulation time. For each Basilisk message to be translated to ROS2 message, we create a custom ROS2 message type that mostly mirrors that of the Basilisk message. But in addition, each ROS2 message also has a header with a time stamp that is associated with the Basilisk simulation time. Even when there is a small discrepancy between the Basilisk clock and ROS2 clock, sensor simulation modules

and algorithms modules in ROS2 can refer to the Basilisk simulation time by checking the ROS2 message header.

Our approach to implementing ROS2 and Basilisk is flexible; it can accommodate various mission scenarios possible by Basilisk beyond the example we consider in this paper. Because the interface only deals with translating the ROS and Basilisk messages from one to another, one can incorporate various actuator models and high-fidelity environmental forces in the Basilisk simulation scenario. In summary, ROS-Basilisk enables the use of astrodynamics simulation capabilities alongside the other modules and flight software implemented in the ROS2 framework.

ROS-NeRF: Camera Simulation and Fast Rendering

To simulate the camera sensor on the servicing spacecraft, we implemented a learning-based image rendering tool that some of the authors developed in a separate paper called ROS-NeRF.[9] ROS-NeRF uses a computer vision technique called the Neural Radiance Fields (NeRF) which is a learning-based approach to volumetric scene representation and novel view synthesis[11] and specializes in spacecraft RPO tasks. ROS-NeRF operates in two phases. In an offline phase, we train the neural network to memorize the 3D scene of the target spacecraft model. The training dataset consists of a sparse set of high-quality images of the target spacecraft with varying pose and lighting conditions. In an online phase, ROS-NeRF queries the trained neural network to render the novel views of the target spacecraft based on the relative position, orientation, and lighting at the current time. The software is implemented as ROS2 module. The implementation details of ROS-NeRF are given in the companion paper.[9] As a ROS2 node, ROS-NeRF subscribes to the ground truth spacecraft states of both target and servicing spacecraft and publishes the rendered image as its output.

ON-ORBIT INSPECTION - AUTONOMOUS ALGORITHM STACK

In this section, we discuss the autonomous algorithms for an example mission scenario. The example scenario considers one servicing spacecraft inspecting another uncooperative, free-orbiting target spacecraft in Low Earth Orbit (LEO). To make the implementation of these algorithms more concrete, we consider a servicing spacecraft equipped with a single warm gas thruster and a set of three-axis reaction wheels.

The autonomous algorithm consists of the following components. First, the vision-based spacecraft detection module detects the target spacecraft. The outputs of the spacecraft detection, GPS, and star tracker measurements are fused in an Extended Kalman Filter (EKF) to estimate the 3D positions and velocities of the servicing and target spacecraft. Then, a formation-keeping controller uses the navigation outputs to maintain the close formation. The attitude guidance module switches between two tasks: pointing a camera to the target and pointing the thruster to the delta-V axis. All of these algorithms are packaged in ROS2 modules.

Vision-based Spacecraft Detection

We implemented a vision-based spacecraft detection algorithm using object detection tools from computer vision. In an off-line phase, we apply transfer learning; that is we additionally train a pre-trained neural network weight to spacecraft detection tasks using a custom dataset. In an online phase, we used the trained neural network to detect the uncooperative target spacecraft in the real-time stream of images obtained from the (simulated) camera sensor.

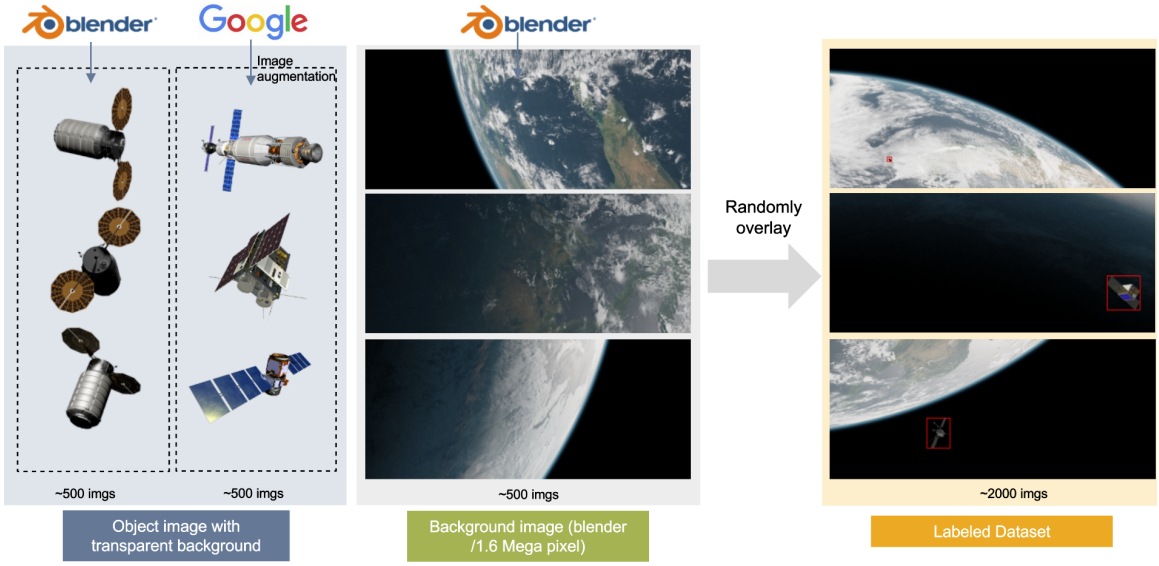

Figure 3. Generating a labeled dataset for training the spacecraft detection algorithm.

Prior to training the neural networks for the spacecraft detection task, we first generated a labeled dataset using synthetic images of spacecraft and Earth as background. A set of synthetic spacecraft images with transparent backgrounds are generated by combing a set of images generated using ray-tracing software as well as other images scraped from the internet. Another set of images consists of background images of Earth, rendered using ray-tracing software, with a variety of camera poses, lighting conditions, and cloud appearances in LEO. We overlay the spacecraft image onto the background image with a random scale range and position to generate a labeled image with a tight bounding box. By randomly combining spacecraft images and backgrounds, we generated 2000 images. Figure 3 summarizes the process of generating the labeled dataset. We consider one classification category of "spacecraft" for detection.

Using the labeled dataset, we train the neural networks to detect the spacecraft. For the results in this paper, we used the YOLOv5[12] neural network architecture with weights that are pre-trained on COCO dataset.[13] Our batch size was 32, and the number of epochs was 300. After the neural network has been trained, the model is validated on an unseen validation dataset. This validation included real space-born footage of CubeSats deployed from the ISS (Figure 4).

Navigation Module

The navigation module jointly estimates the 3D positions and velocities of both the servicing and the target spacecraft via Extended Kalman Filter (EKF). The available measurements are the noisy GPS measurement of the servicing and the size and location of bounding boxes from the target detection. The navigation uses the noisy star tracker measurement as the attitude estimate for the servicing spacecraft. Assuming that we have a priori information on the target spacecraft scale, we used the size of the detected bounding box to compute the coarse range estimate while the center of the bounding box is for bearing. Since the likelihood of error in object detection is non-trivial, we heuristically reject the outliers of bounding box measurements by thresholding at a confidence bound calculated by projecting covariance estimates onto bearing and range measurement space.

Figure 4. Space-borne image of MinXSS (bottom) and CADRE spacecraft (top) deployed from ISS (photo was taken from ISS). Our spacecraft detection algorithm successfully detects both satellites.

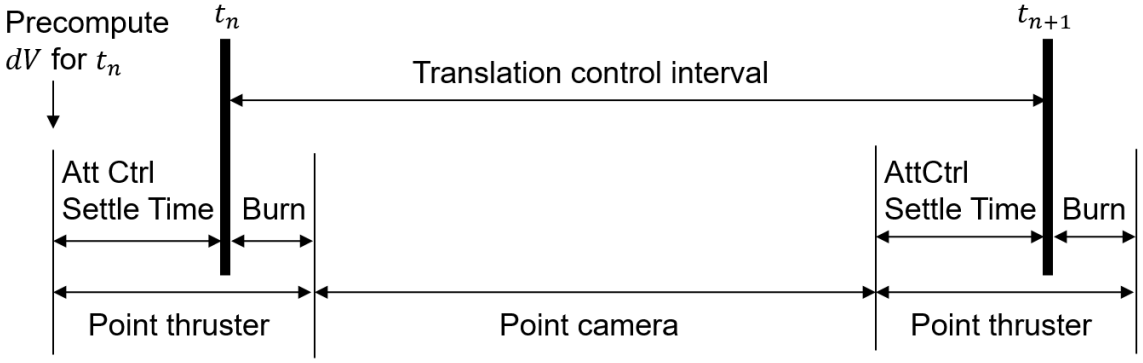

Figure 5. Attitude pointing schedule for alternating between delta-v maneuvers and visual tracking of the target.

Formation Keeping and Attitude and Control

We designed an integrated algorithm for formation keeping and attitude control. The objectives of this module are (1) to maintain an along-track formation with respect to the target spacecraft by applying periodic burns and (2) to point the camera at the target whenever possible. For the translational control, we apply a Linear Quadratic Regulator (LQR) with linearized relative orbital dynamics at a low rate of every 200 sec. Since the servicing spacecraft has a single thruster, the attitude controller must reorient the spacecraft to align the thrust vector to the desired direction during the burn. Given the desired delta-V maneuvers, the attitude control system balances between pointing the camera at the target spacecraft and pointing the thruster in the delta-V direction. The main idea is that the spacecraft briefly "looks away" from the target for the delta-V maneuver and returns to pointing the camera once the burn is complete. This attitude-pointing schedule strategy is shown in the table in Figure 5. Finally, given the attitude-pointing commands and the current attitude estimate from the navigation module, the attitude controller computes the desired torque using attitude error feedback and allocates torque to each of the three reaction wheels.

As a ROS2 node, the formation-keeping and attitude control system subscribes to the estimates of servicing and target from the Navigation module. The module publishes the thruster burn duration and the torque commands on each of the reaction wheels. These outgoing messages are sent to ROS-Basilisk which propagates dynamics based on these commands.

SIMULATION RESULTS

We validated our algorithms and simulation capabilities in an example mission scenario. All the modules are implemented in ROS2 and the simulation is run in a real-time and closed-loop fashion. For this scenario, a servicing spacecraft is tasked to visually track an uncooperative target spacecraft and maintain a close formation 30 m distance but they are initially separated by 50 m. The servicing spacecraft is equipped with a camera, a single thruster, and 3-axis reaction wheels. The simulation was carried out on a desktop computer with a CPU of Intel i7-8700 @ 3.20GHz and a GPU of GTX 1070 (8GB memory). Spacecraft dynamics and attitude control ran at a high rate of 5 Hz. Image rendering, image detection, position EKF, and attitude guidance ran at 1Hz.

Bearing and range observations computed from the object detection algorithm are shown in Figure 6. These plots show the inliers that were detected after the algorithm rejects erroneously de-

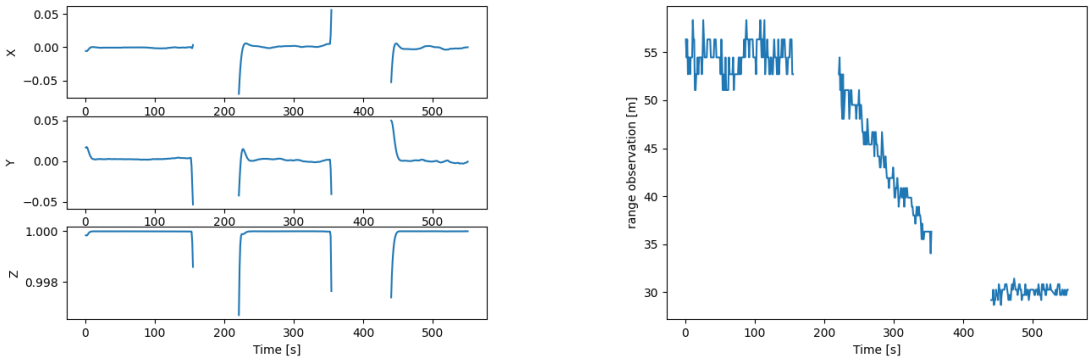

Figure 6. Bearing unit vector (left) and range (right) observations. Measurements drop out when the attitude controller reorients spacecraft for delta-V maneuvers.

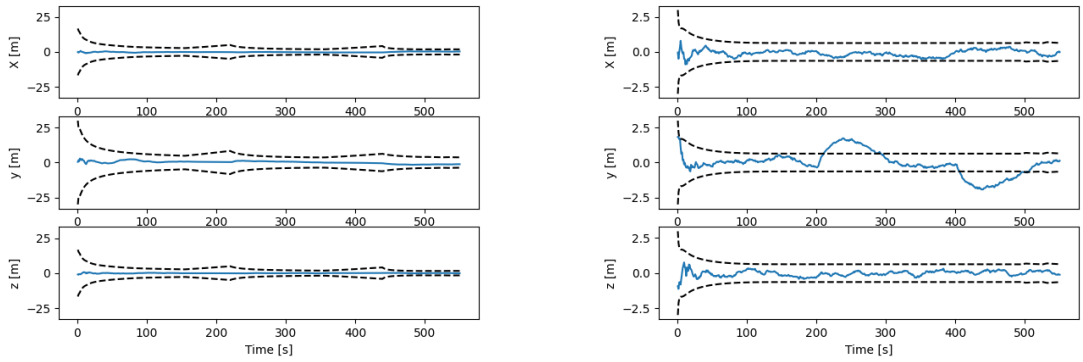

Figure 7. Position estimation error of target (left) and servicing (right) and 3-sigma bound

tected bounding boxes. The measurements drop out are associated with spacecraft reorienting for the delta-v maneuvers so the camera temporarily looks away from the target.

Figure 7 shows the 3D position estimation errors for both target and servicing spacecraft. The figures show that both servicing and target positions are estimated consistently within a few meters. The 3-sigma bounds for target position estimate show that every 200 sec, the uncertainty of the estimate grows during the thruster pointing maneuvers but as soon as the camera is pointed back to the target and the spacecraft is detected, the estimation uncertainty reduces to the nominal value. This shows that even though the servicing spacecraft loses the sight of target temporarily, the navigation module propagates the last known states with sufficient accuracy to visually acquire the target spacecraft again after some time.

Finally, Figure 8 shows the relative position control tracking error with respect to the desired formation. Initially, the spacecraft is separated by approximately 50 m but the formation-keeping control reduces the error to 30 m.

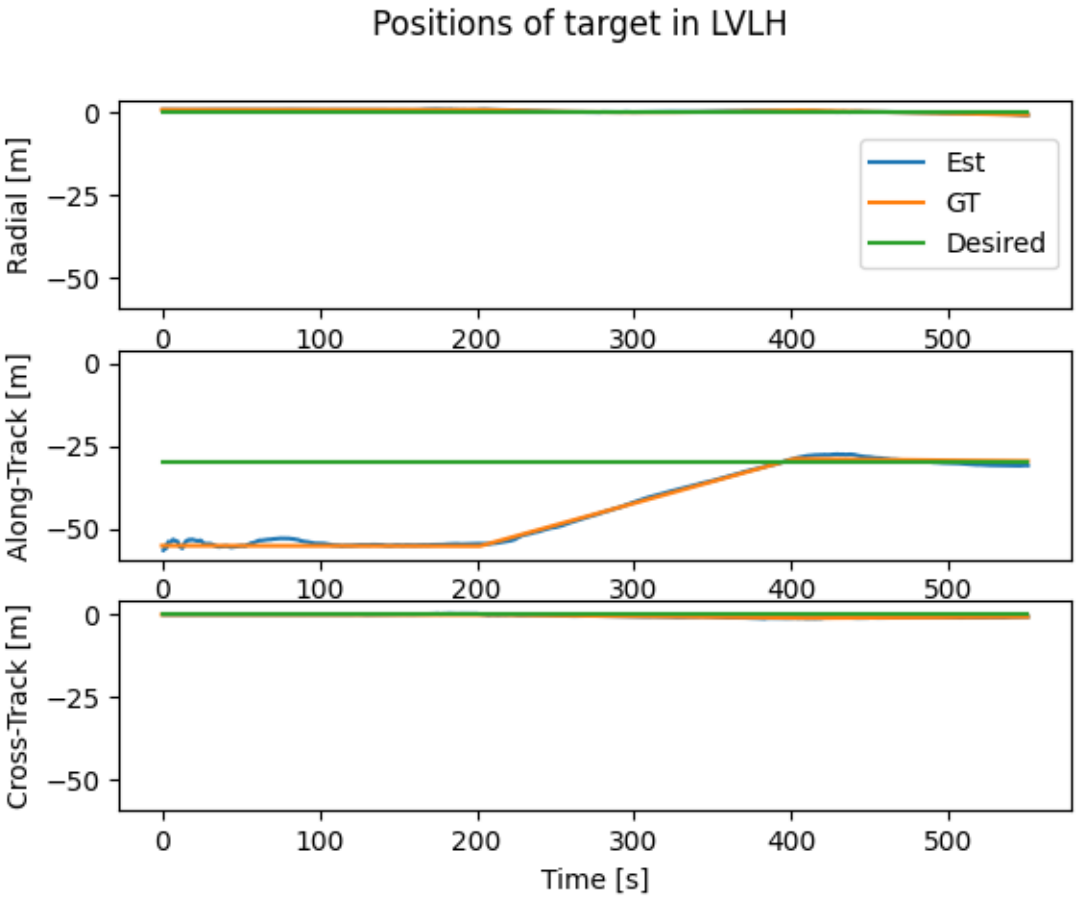

Figure 8. Relative position of the target with respect to the servicing spacecraft in LVLH frame. Blue is the estimated position, orange is the ground truth from the simulation and green is the desired formation.

CONCLUSION

In this paper, we demonstrated the autonomous on-orbit inspection algorithms as well as spacecraft dynamics and sensor simulation modules all within Robot Operating System 2 or ROS2. For the autonomous on-orbit inspection algorithm, we developed an end-to-end pipeline for vision-based spacecraft detection, camera pointing, formation-keeping control, and control allocation. For dynamics, we developed ROS-Basilisk, a ROS2 software interface that wraps Basilisk, an open-source astrodynamics simulation software. ROS-Basilisk allows flight software written in ROS2 to interact with high-fidelity spacecraft dynamics simulation in the loop. For the camera sensor, we integrated ROS-NeRF which uses the Neural Radiance Fields to rapidly render spacecraft images in real-time. Finally, we integrated all these modules in a closed-loop fashion and demonstrated successful visual tracking and formation keeping in a realistic on-orbit inspection of an uncooperative satellite by a servicing spacecraft.

ACKNOWLEDGMENT

This research was supported in part by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors were also funded by Carl and Shirley Larson Endowment (Zhang), the Aerospace Corporation (Ragheb), and Caltech's Information Science and Technology program (Ohenzuwa). The authors would like to also thank Fred Hadaegh, Charles Norton, Amir Rahmani, Sorina Lupu, Darren Rowen, and Robert Lee for their technical input.

REFERENCES

[1] J. W. Gangestad, C. C. Venturini, D. A. Hinkley, and G. Kinum, "A Sat-to-Sat Inspection Demonstration with the AeroCube-10 1.5 U CubeSats," 2021.

[2] E. Gill, S. D'Amico, and O. Montenbruck, "Autonomous formation flying for the PRISMA mission," *Journal of Spacecraft and Rockets*, Vol. 44, No. 3, 2007, pp. 671–681.

[3] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, Vol. 7, No. 66, 2022, p. eabm6074.

[4] L. Fluckiger and B. Coltin, "Astrobee robot software: Enabling mobile autonomy on the iss," tech. rep., 2019.

[5] J. Badger, D. Gooding, K. Ensley, K. Hambuchen, and A. Thackston, "ROS in space: A case study on robonaut 2," *Robot Operating System (ROS)*, pp. 343–373, Springer, 2016.

[6] K. Chow, N. Oune, J. Singh, D. Thomlinson, and T. A. Battista, "Simulating Spacecraft Docking and Berthing Using ROS and Gazebo," *AIAA SCITECH 2022 Forum*, 2022, p. 2506.

[7] Y. K. Nakka, R. C. Foust, E. S. Lupu, D. B. Elliott, I. S. Crowell, S.-J. Chung, and F. Y. Hadaegh, "A six degree-of-freedom spacecraft dynamics simulator for formation control research," 2018.

[8] P. W. Kenneally, S. Piggott, and H. Schaub, "Basilisk: A flexible, scalable and modular astrodynamics simulation framework," *Journal of aerospace information systems*, Vol. 17, No. 9, 2020, pp. 496–507.

[9] K. Matsuka, C. Ohenzuwa, and S.-J. Chung, "Rapid Synthetic Image Generation Using Neural Radiance Fields for Vision-based Formation Flying Spacecraft," *33rd AAS/AIAA Space Flight Mechanics Meeting*, 2023.

[10] S. v. Overeem and H. Schaub, "IWSCFF 19-88 SMALL SATELLITE FORMATION FLYING APPLICATION USING THE BASILISK ASTRODYNAMICS SOFTWARE ARCHITECTURE," 07 2019.

[11] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," *ECCV*, 2020.

[12] G. J. e. al., "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements," Oct. 2020, 10.5281/zenodo.4154370.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *European conference on computer vision*, Springer, 2014, pp. 740–755.