AgriNav: UAV Simulator for Vision-based Navigation in Agricultural Environments

Teaya Yang [*] **Mark W. Mueller** [*]

[*] *High Performance Robotics Lab, Department of Mechanical Engineering, University of California, Berkeley, CA 94709, USA.*

Abstract: We present AgriNav, a simulator for vision-based navigation and data collection for unmanned aerial vehicles (UAVs) in agricultural environments. Developing autonomous aerial robots for large-scale agricultural data collection requires significant effort in hardware-software integration, especially for accurate state estimation and navigation. While many existing simulators offer image generation features, they are not designed for direct use with visual-inertial odometry (VIO) packages, which depend on precise sensor calibration, data synchronization, and specific message formats. We address this challenge by generating synthetic data that mimic physical sensor outputs and providing a modular communication framework, enabling users to efficiently experiment with odometry and navigation algorithms without requiring hardware testing. Using Unity for rendering, we provide pre-built maps and plant models for rapid testing and simple scene customization. AgriNav is open source and available at: https://github.com/Teaya-Yang/AgriNav.git.

Keywords: Agricultural Robotics, Machine Vision, Sensing and Automation with UAVs, Precision Agriculture, Simulation, Robotic Navigation

1. INTRODUCTION

Unmanned aerial vehicles (UAVs) offer an effective solution for agricultural data collection due to their affordability, agility, and efficiency in large-scale sensing and monitoring. In recent years, UAVs have been used for applications such as yield prediction, plant health monitoring, and pesticide spraying (Aslan et al. 2022). However, collecting data in complex environments like dense orchards and greenhouses remains challenging due to the difficulty of developing aerial robots that can navigate safely and reliably in these settings (Velusamy et al. 2021). Navigation methods in these environments are often limited, as signals from the Global Navigation Satellite System can be unreliable, and alternatives like Ultra-Wideband or motion capture systems require costly external infrastructure (Gyagenda et al. 2022).

Vision-based navigation offers a solution to this challenge by removing the dependency on external infrastructure, making it adaptable across different environments while reducing reliance on costly sensors like LiDAR. For instance, Servières et al. (2021) highlights the advantage of using inexpensive and flexible visual sensors in combination with modern computer vision algorithms. For visual-inertial odometry (VIO), widely adopted methods include OpenVINS (Geneva et al. 2020), which uses the Multi-State Constraint Kalman Filter (MSCKF) for computationally efficient state estimation; OKVIS (Leutenegger et al. 2015), which performs sliding-window optimization over past states and landmarks; and VINS-Mono (Qin et al. 2018), which applies factor graph optimization to achieve high-accuracy pose estimation. For purely visual SLAM, ORB-SLAM (Mur-Artal et al. 2015) remains a popular choice due to its reliable feature tracking and loop closure capabilities. Although these state-of-the-art navigation packages are open-source and supported by active user communities, their setup and implementation typically require expertise in vision-based systems and state estimation. They offer excellent support for users working with commonly used hardware platforms, with community-shared calibration files and detailed usage instructions. However, these resources are primarily tailored to commercial sensors or existing datasets, making it challenging to adapt the systems to custom setups or to evaluate them in simulation without physical hardware.

On the other hand, many simulators have been introduced to accelerate the system development process. For instance, AirSim (Shah et al. 2018) emphasizes high-quality visuals to reduce the sim-to-real gap, while Flightmare (Song et al. 2021) provides a realistic multi-modal sensor suite for developing control and planning algorithms. More recently, the Pegasus Simulator (Jacinto et al. 2024) offers similar realistic visual outputs by leveraging the NVIDIA Isaac Sim as the base framework. Our previous work (Zha et al. 2024) builds upon AirSim to better support simulations for agricultural applications. While the photorealistic rendering capabilities of these simulators are well suited for dataset generation, a gap remains in their integration with state-of-the-art navigation packages. Constructing a simulator that interfaces with existing navigation algorithms faces several key challenges: i) providing accurate sensor calibration information, ii) ensuring compatibility

[⋆] This work was supported by the Agriculture and Food Research Initiative (AFRI) Competitive Grant no. 2020-67021-32855/project accession no. 1024262 from the USDA National Institute of Food and Agriculture and the William C. Webster Graduate Fellowship.

in sensor output formats, iii) enabling initialization process for proper localization, and iv) managing the rendering overhead in image generation. We build upon our previous work to propose a simulator that addresses these challenges while maintaining the focus on agricultural applications, making it easier for users to evaluate and experiment with different vision-based navigation methods for UAVs. In Section 2, we introduce the improved modular communication structure. Section 3 outlines the key features of our simulator, and example use cases are provided in Section 4.

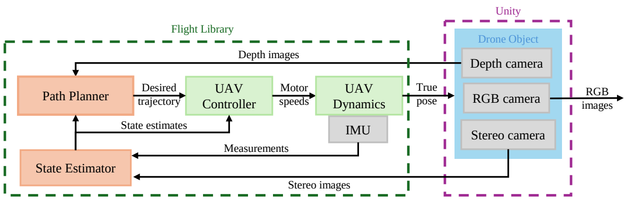


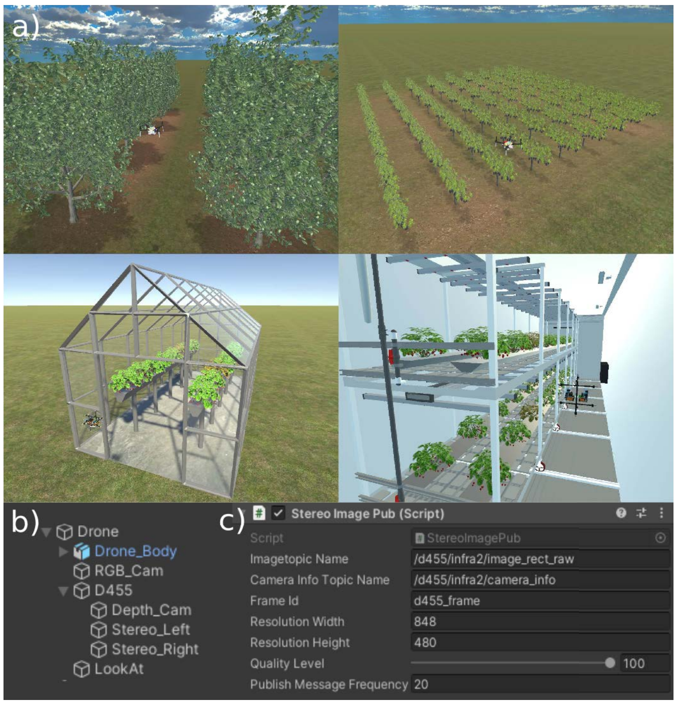
Fig. 1. Diagram illustrating the communication structure within the simulator. Communication between modules is handled through the ROS framework, supporting modular integration of autonomy components.

2. SIMULATOR STRUCTURE

A major improvement in our communication framework compared to our previous work (Zha et al. 2024) is the removal of the AirSim-based structure. Instead, we adopt a lightweight communication method using the unmodified ROS-TCP-Connector and ROS-TCP-Endpoint packages developed by the Unity Robotics Hub (Unity Technologies 2025a, Unity Technologies 2025b). These packages establish TCP-based ROS communication between Unity and external nodes. We use them as a minimal communication middleware to link Unity with our provided flight library, reducing overhead and avoiding the constraints of heavier simulators like AirSim. This ensures modularity and compatibility with a wide range of ROS-based navigation packages. A schematic illustrating the communication between different components through ROS is shown in Fig. 1.

2.1 Flight library

The core components of the provided flight library include a UAV controller and a dynamics simulator, enabling basic autonomous flight and state feedback. The simulator node also generates realistic IMU outputs, including one published in `/sensor_msgs/Imu` format, a message type commonly used for physical sensors in ROS. Users can also optionally simulate the state estimator and path planner (shown in orange in Fig. 1), with examples provided in the following sections. The flight library is designed to be modular while maintaining realistic sensor outputs, allowing users to experiment with different navigation algorithms. Finally, the simulated vehicle pose is published as a ROS message and sent to Unity, ensuring precise and up-to-date image feedback.



Stereo Image Pub (Script)

Script	StereoImagePub
Imagetopic Name	/d455/infra2/image_rect_raw
Camera Info Topic Name	/d455/infra2/camera_info
Frame Id	d455_frame
Resolution Width	848
Resolution Height	480
Quality Level	100
Publish Message Frequency	20

Fig. 2. (a) Example scenes built using provided models in Unity. (b) Drone object components including simulated Realsense D455 camera as shown in the Hierarchy window of Unity Editor. (c) Example of image publisher manager as shown in the Inspector window of Unity Editor, which allows users to specify topic name, resolution, and publishing frequency.

2.2 Unity rendering

We use Unity as the rendering engine for its accessibility and low hardware requirements. Our simulator includes several pre-built models, such as high-fidelity plant models generated using the Helios 3D framework (Bailey 2019), which can be combined to create realistic agricultural scenes (Fig. 2a). Users can easily edit the scene by dragging and placing the provided models within the Unity Editor. A drone object is included by default, equipped with a simulated camera that emulates the sensors of the Intel RealSense D455 depth camera (Fig. 2b). Depending on the application, users can enable or disable the depth, stereo, or RGB cameras and add additional cameras to capture more data. As the vehicle moves based on simulated poses, onboard images are captured from the current drone pose. Additionally, we provide image publisher scripts with a custom camera manager (Fig. 2c), allowing users to specify ROS topic names and adjust the publishing frequency by modifying parameters. In the provided example, stereo images are published in monochrome, and depth images are generated using a depth shader that replicates the D455 depth scale. AgriNav includes user-friendly Unity editing tools and detailed instructions, ensuring minimal effort is required to customize the simulator before use.

3. SUPPORTED FEATURES

The main contribution of our proposed simulator, compared to existing ones, is its ability to support closed-loop experimentation with vision-based navigation packages. While many simulators generate visual sensor data and inertial measurements, which are essential for visual-inertial odometry algorithms, they often overlook critical aspects such as camera calibration, algorithm initialization, and handling rendering-induced synchronization issues, making direct integration with existing navigation algorithms difficult. In this section, we outline the key features in AgriNav that enable VIO simulation, integrating with OpenVINS as an example. Additionally, we highlight features that make our simulator well-suited for a wide range of agricultural applications.

3.1 Sensor calibration and configuration

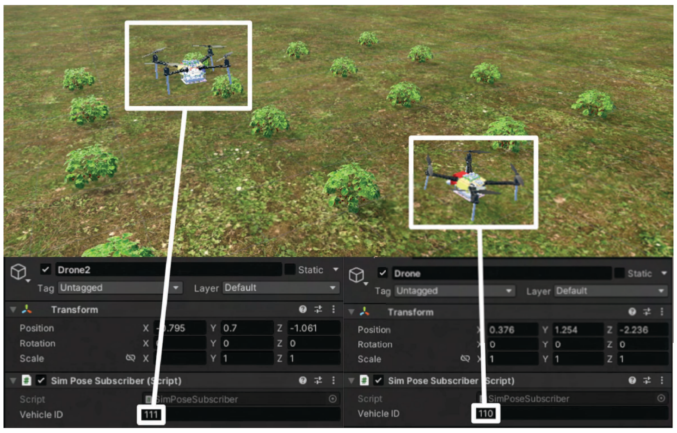
Most VIO algorithms require configuration files that specify sensor parameters. These include IMU noise densities and biases, camera intrinsics, and the transformation matrices between cameras and IMUs. For instance, OpenVINS uses the IMU profile in the high-frequency propagation steps, while camera intrinsic and extrinsic parameters are needed during measurement updates. Typically, these values are obtained through calibration tools such as Kalibr (Rehder et al. 2016), but collecting required calibration data in simulation is challenging. We address this issue by setting ground truth values for these critical parameters and configuring calibration settings based on sensor datasheets, making integration straightforward while reflecting true sensor characteristics.

3.2 Support for VIO initialization

In addition to sensor configuration, many navigation algorithms, particularly those relying on visual odometry, require an initialization process. This typically involves hand-held camera motion to initialize the algorithm, after which the state estimates can be published properly. However, most simulators start the vehicle perfectly at rest, which could result in failed initialization or dynamic initialization that leads to poor estimation performance. To address this, we introduce a short initialization sequence that lifts the vehicle slightly, allowing the system to initialize correctly. This process runs autonomously, requiring the user to simply wait for it to complete before testing control and planning algorithms. Examples in Section 4 begin with this initialization process.

3.3 Frame rate management using simulated time

Another significant challenge in vision-based simulation is the computational overhead required for image generation, which can vary depending on hardware capabilities. In some cases, rendering delays prevent images from being published at the desired frame rate, causing unexpected navigation performance. However, this is not an issue with physical cameras. To ensure consistent testing regardless of compute limitations, we use the simulation time feature in ROS. Our simulator node publishes to the `/clock` topic and subscribes to the relevant image topics, ensuring that the clock only advances when images are received at the intended frame rate. Users may choose to use either the simulated clock or the wall clock based on hardware considerations. To demonstrate this capability, all example



Drone2 — Static — Tag: Untagged — Layer: Default
Transform
Position X -0.795 Y 0.7 Z -1.061
Rotation X Y 0 Z 0
Scale X Y 1 Z 1
Sim Pose Subscriber (Script)
Script: SimPoseSubscriber
Vehicle ID: 111

Drone — Static — Tag: Untagged — Layer: Default
Transform
Position X 0.376 Y 1.254 Z -2.236
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sim Pose Subscriber (Script)
Script: SimPoseSubscriber
Vehicle ID: 110

Fig. 3. Unity setup for multiagent simulation. Users only need to make replicates of the drone object and specify different vehicle IDs in the Inspector window.

data in the following section are collected using simulated time.

3.4 Vision-based path planning

Using the provided modular framework, visual information and VIO outputs can be used to test other autonomy components, such as vision-based path planners. Users can easily publish images as ROS messages and subscribe to them in a planner node, enabling the evaluation of planning algorithms. In Section 4, we demonstrate an example using the Rectangular Pyramid Partitioning algorithm presented in Bucki et al. (2020), which requires depth images from the simulated depth camera as input. Additionally, planners that rely on perception information, such as semantic mapping-based (Ryll et al. 2020) or feature-based (Wu et al. 2022) motion planning algorithms, can also be tested within our provided framework.

3.5 Multi-agent simulation

Many agricultural applications require large-scale data collection, making multi-agent cooperative missions an important aspect of system autonomy. Our simulator fully supports multi-agent simulation while maintaining all the previously mentioned features. Users can enable multiple agents by duplicating the drone object and assigning unique vehicle IDs to each of them, as shown in Fig. 3. These vehicle IDs must also be provided as arguments to the simulator node, ensuring proper functionality without additional modifications.

4. EXAMPLE USE CASES

To showcase the key features of AgriNav, we present several example use cases that demonstrate its capabilities in vision-based navigation, planning, and data collection. These examples illustrate how the simulator can be used for evaluating VIO performance, analyzing path planning results, and assessing additional vision-based algorithms using onboard data.

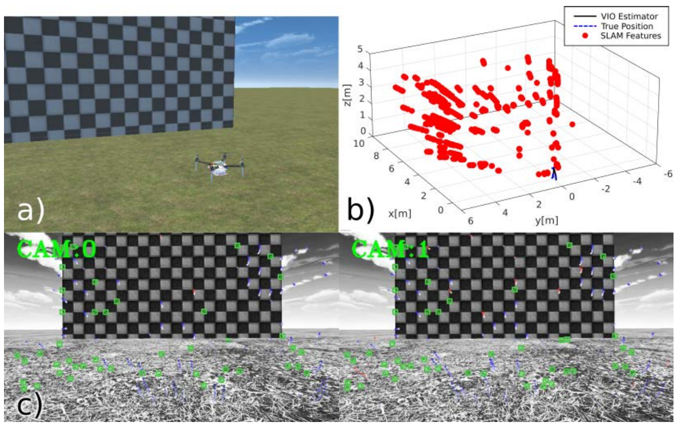


Fig. 4. a) Example scene in which hovering test is performed with VIO. b) Visualization of VIO estimated trajectory and SLAM features used to generate state estimates. c) Onboard feature tracking results using stereo images with actively tracked features marked in green.

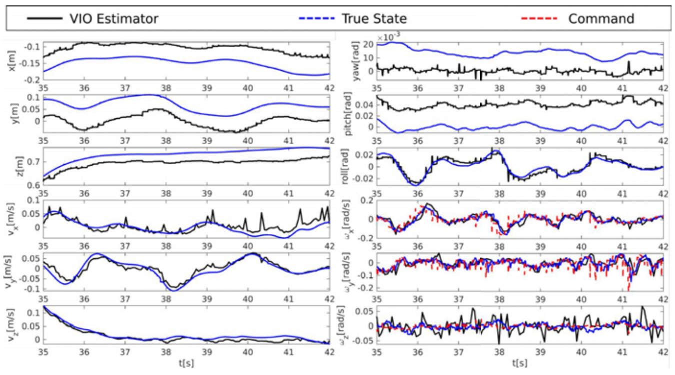


Fig. 5. Comparison of the full vehicle state estimated by VIO with the simulated ground truth during the hover test. Control commands issued as desired angular velocities are compared against both the ground-truth and estimated vehicle states.

4.1 Closed-loop hover test with VIO

We begin with a hover test with VIO in the loop, as shown in Fig. 4a. The vehicle is commanded to hover in front of a checkerboard to assess state estimation performance. The initialization sequence detailed in Section 3.2 is executed first, followed by a controlled hover for a few seconds before landing. Stereo images and IMU data serve as inputs to the VIO algorithm, with the resulting odometry used for state feedback in vehicle control. The executed trajectory and the features contributing to odometry estimation are plotted in Fig. 4b. Sample onboard stereo images are shown in Fig. 4c, with actively tracked 2D features highlighted in green. Additionally, we demonstrate the ability to analyze VIO flight performance in closed-loop using our proposed framework. Fig. 5 presents the full vehicle state estimated by the VIO algorithm compared to the simulated ground truth, as well as the control commands generated which can be used to evaluate the angular velocity tracking performance.

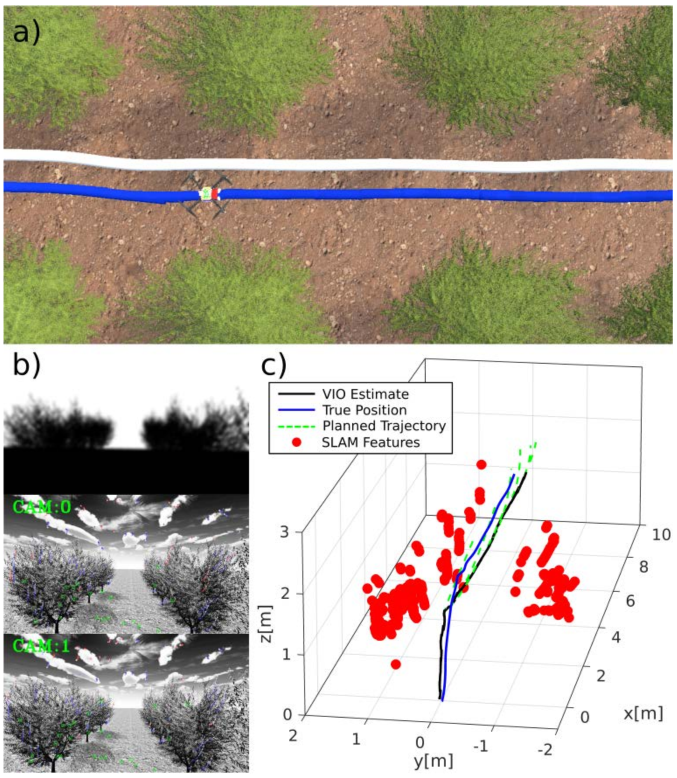


Fig. 6. a) Example scene with executed trajectories. The true trajectory is marked in white and the estimated trajectory is marked in blue. b) Samples of onboard images used as inputs for planning and estimation algorithms. c) Visualization of the trajectories and features used for localization. The planner provides trajectories in a receding-horizon fashion, marked in green.

4.2 Through-the-canopy flight with VIO and depth-based planning

We provide an additional example demonstrating how our simulation framework can be used to test vision-based planning algorithms alongside vision-based odometry. In this scenario, the drone is commanded to fly through two rows of trees. We use a vehicle controller similar to that in the previous section, with the addition of a depth-image-based planner outlined in Bucki et al. (2020). The planned trajectories are generated through collision checking using the depth image, and the optimal trajectory, shown in green in Fig. 6c, is executed in a receding horizon fashion until the goal is reached. To visualize system performance, we also provide samples of the collected onboard images with the estimated and ground-truth vehicle trajectories in Fig. 6.

4.3 Simulated image capture for detection evaluation

Since additional cameras can be added to the drone object and published as image topics, our simulator also supports image data collection to evaluate detection algorithms in postprocessing. In Fig. 7, we present an example in which ground truth annotations for visible fruits are generated using tree model information. Detection algorithms can

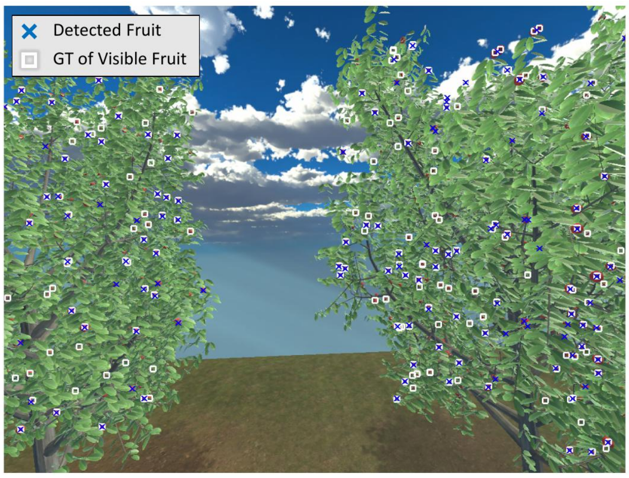


Fig. 7. Example for fruit detection. The white markers are generated using the ground-truth fruit positions, and the blue markers indicate the fruits detected by the chosen algorithm.

be evaluated against these annotations in the captured images.

5. CONCLUSION

In this work, we introduce AgriNav, a simulation framework that addresses the integration gap between vision-based navigation algorithms and existing tools. We achieve this by implementing a simple yet modular communication framework, offering customizable maps and object managers, and ensuring that simulated sensor signals reflect real hardware behavior while managing synchronization constraints. Although our current examples support only OpenVINS and the RealSense D455 camera, our aim is to expand compatibility with a wider range of sensors and provide instructions for additional navigation suites, further enhancing the simulator's flexibility for testing autonomy algorithms across diverse agricultural applications. Another important future direction is to improve the realism of simulated camera outputs by incorporating effects such as motion blur and lens distortion, as these factors can also influence vision-based navigation performance.

REFERENCES

Aslan, M.F., Durdu, A., Sabanci, K., Ropelewska, E., and Gültekin, S.S. (2022). A comprehensive survey of the recent studies with uav for precision agriculture in open fields and greenhouses. *Applied Sciences*, 12(3), 1047.

Bailey, B.N. (2019). Helios: A scalable 3d plant and environmental biophysical modeling framework. *Frontiers in Plant Science*, 10, 1185.

Bucki, N., Lee, J., and Mueller, M.W. (2020). Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation. *IEEE Robotics and Automation Letters*, 5(3), 4626–4633.

Geneva, P., Eckenhoff, K., Lee, W., Yang, Y., and Huang, G. (2020). Openvins: A research platform for visual-inertial estimation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4666–4672. IEEE.

Gyagenda, N., Hatilima, J.V., Roth, H., and Zhmud, V. (2022). A review of gnss-independent uav navigation techniques. *Robotics and Autonomous Systems*, 152, 104069.

Jacinto, M., Pinto, J., Patrikar, J., Keller, J., Cunha, R., Scherer, S., and Pascoal, A. (2024). Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation. In *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, 917–922. IEEE.

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2015). Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3), 314–334.

Mur-Artal, R., Montiel, J.M.M., and Tardos, J.D. (2015). Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5), 1147–1163.

Qin, T., Li, P., and Shen, S. (2018). Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE transactions on robotics*, 34(4), 1004–1020.

Rehder, J., Nikolic, J., Schneider, T., Hinzmann, T., and Siegwart, R. (2016). Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 4304–4311. IEEE.

Ryll, M., Ware, J., Carter, J., and Roy, N. (2020). Semantic trajectory planning for long-distant unmanned aerial vehicle navigation in urban environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1551–1558. IEEE.

Servières, M., Renaudin, V., Dupuis, A., and Antigny, N. (2021). Visual and visual-inertial slam: State of the art, classification, and experimental benchmarking. *Journal of Sensors*, 2021(1), 2054828.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, 621–635. Springer.

Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D. (2021). Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, 1147–1157. PMLR.

Unity Technologies (2025a). Ros-tcp-connector. URL `https://github.com/Unity-Technologies/ROS-TCP-Connector`. Accessed: Feb. 23, 2025.

Unity Technologies (2025b). Ros-tcp-endpoint. URL `https://github.com/Unity-Technologies/ROS-TCP-Endpoint`. Accessed: Feb. 23, 2025.

Velusamy, P., Rajendran, S., Mahendran, R.K., Naseer, S., Shafiq, M., and Choi, J.G. (2021). Unmanned aerial vehicles (uav) in precision agriculture: Applications and challenges. *Energies*, 15(1), 217.

Wu, X., Chen, S., Sreenath, K., and Mueller, M.W. (2022). Perception-aware receding horizon trajectory planning for multicopters with visual-inertial odometry. *IEEE Access*, 10, 87911–87922.

Zha, J., Yang, T., and Mueller, M.W. (2024). Agri-fly: simulator for uncrewed aerial vehicle flight in agricultural environments. *IEEE Access*.