Comparison of Visual and LiDAR SLAM Algorithms using NASA Flight Test Data

Keerthana Kannan*, Anjan Chakrabarty† and Joshua Baculi‡
KBR Wyle Services, NASA Ames Research Center, Moffett Field, CA 94035

Evan Kawamura§, Wendy Holforty¶ and Corey Ippolito‖
NASA Ames Research Center, Moffett Field, CA 94035

Simultaneous Localization and Mapping (SLAM) is a promising technique that provides localization information and precise mapping of the physical environment without having much prior knowledge of the surroundings. SLAM may have a vital role in aeronautics and aerospace, where vehicles and aircraft must operate in complex environments with traditional localization services that may be degraded or unavailable. This paper compares several pre-canned 3D SLAM algorithms based on vision and LiDAR, namely ORB-SLAM, ORB-SLAM2, LOAM, A-LOAM, and F-LOAM on NASA UAS (Unmanned Aircraft System) flight test data. The NASA ARC UAS flight test demonstrates preliminary SLAM algorithm results, which serve as a stepping stone to simulated AAM (Advanced Air Mobility) concepts. Conducting AFRC UAS flight test for simulated AAM approach and landing with SLAM algorithms provides an Alternative Precision Navigation and Timing solution based on distributed landmarks and fiducials in the landing zone. These algorithms use the telemetry data as ground truth for a baseline comparison. The criteria of the performance comparison include robustness, accuracy, re-localization, response to environmental changes, and real-time effectiveness, which are currently qualitative but to be quantitative in the future.

I. Introduction

SLAM estimates sensor motion and reconstructs 3D structure in an unfamiliar environment [1, 2]. Its original purpose was to attain autonomy in robots [3]. However, soon after in the years, its applications broadened into the field of computer vision, augmented reality [4], and self-driving technology [5–7].

SLAM with visual information from cameras is visual SLAM (vSLAM), in which vSLAM estimates the camera's trajectory by visually reconstructing the environment based on keyframes and detected features. The vSLAM technique spreads vastly because of its suitability for camera pose estimation, map optimization, and relocalization using easily accessible GoPros or smartphones. vSLAM has three main stages – initialization, tracking, and mapping [8]. Each algorithm uses a different approach for each module mentioned in the later sections. In general, the first stage involves defining a global coordinate system and reconstructing a portion of the environment in the global coordinates as an initial map. Next, continuous tracking and mapping provide periodic estimations of the camera position and orientation. Then the algorithm tracks the initial map in the image to approximate the camera pose of it for the map by obtaining the 3D correlation between the image and the initial map from feature matching. The initial map then expands by calculating the 3D structures of the scenario when the camera captures unknown regions not mapped before. Finally, relocalization comes into play when SLAM loses track due to fast camera movements and other external disturbances where it computes the camera pose with the map again to continue tracking. Global map optimization suppresses or reduces the accumulated errors due to the estimation, and the loop closing is essential to estimate the error accumulated due to camera motion. The search for a closed loop in the current image compared to the previously observed images takes place, and if one of the once-obtained views gets noticed, it means a loop is detected. Bundle adjustment can minimize this error by accurately estimating the camera localization and the geometrical map [9].

*Embedded Software Developer, KBR Wyle Services, Intelligent Systems Division, Mail Stop 269-1, AIAA member.
†Research Engineer,KBR Wyle Services, Intelligent Systems Division, Mail Stop 269-1.
‡Systems Engineer,KBR Wyle Services, Intelligent Systems Division, Mail Stop 269-1.
§Computer/GNC Engineer, Intelligent Systems Division, Mail Stop 269-1, AIAA member.
¶Technical Engineer, Intelligent Systems Division, Mail Stop 269-1.
‖Aerospace Scientist, Intelligent Systems Division, Mail Stop 269-1, AIAA member.

The SLAM algorithm that uses LiDAR and optional inertial sensor data to generate a 3D map of the environment is called LiDAR-based SLAM. LiDAR Odometry and Mapping (LOAM) is a method to estimate the state and map the 3D LiDAR data in real time. LOAM generates the real-time odometry and map using range measurements for a 2-axis LiDAR moving in 6 DOF (degree of freedom) [10].

This paper reviews the initial results of the real-time open-source pre-canned SLAM algorithms using the flight test data obtained from NASA Ames Research Center (ARC) and NASA Neil A. Armstrong Flight Research Center (AFRC). This paper aims to compare the feature-based approach module in vSLAM for monocular camera inputs and LiDAR SLAM algorithms against the telemetry data as ground truth. Though these algorithms originally were not written for flight datasets, the reason to choose them is to test their ability to detect features from the sky and their capability to plot flight trajectories with precision. Unfortunately, the features on the ground do not appear the same when seen from an aircraft (due to scaling), so this will serve as a preliminary survey for the prospective SLAM algorithms for flights [11]. Future work may include processing more datasets and comparing other SLAM algorithms.

II. SLAM Algorithms

This section outlines the theory and mechanisms of the SLAM methods under consideration. Since the flight test payload has a LiDAR, IMU, and a visual sensor, this will have a mixture of LOAM and vSLAM algorithms.

A. ORB-SLAM

ORB-SLAM uses Parallel Tracking And Mapping (PTAM), where the threads for tracking, mapping, and loop closing run in parallel (as shown in Fig.1). In monitoring, the camera localizes and decides on the keyframe to insert. Feature matching of the current frame with the previous frame optimizes the pose using motion-only bundle adjustment. If the tracking gets lost in the middle of the execution, the place recognition function kicks in to perform relocalization. The embedded bag of words in DBoW2* [12] (an open-source C++ library) performs loop detection and relocalization. After the initial feature matching and camera pose estimation, the co-visibility graph of keyframes generates the visible map [13]. The mapping operates on the keyframes and performs local Bundle Adjustment (BA) to 3D reconstruct the surroundings. Only the high-quality feature points remain after sorting the information from tracking, eliminating redundant keyframes. Finally, in the loop closing, the system searches for a loop in every new keyframe. When two keyframes share a lot of bag of words, they qualify for the loop closer. If a loop gets detected, the similarity transformation aids in fusing all the identical points. A pose graph optimization in g2o† [14] (an open-source C++ framework) over similarity transformations takes place to correct the scale drift [15].

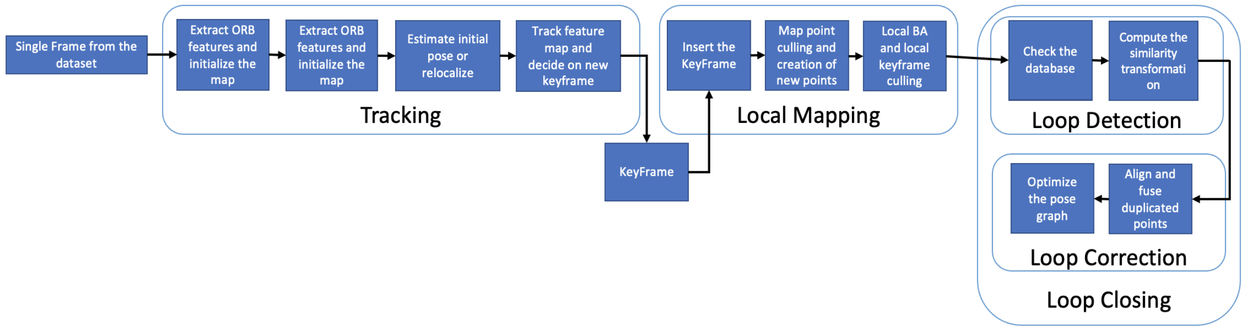

Single Frame from the dataset
Extract ORB features and initialize the map
Extract ORB features and initialize the map
Estimate initial pose or relocalize
Track feature map and decide on new keyframe
Tracking
KeyFrame
Insert the KeyFrame
Map point culling and creation of new points
Local BA and local keyframe culling
Local Mapping
Check the database
Compute the similarity transformati on
Loop Detection
Optimize the pose graph
Align and fuse duplicated points
Loop Correction
Loop Closing

Fig. 1 Block Diagram of ORB-SLAM with its three parallel threads - tracking, local mapping and loop closing

B. ORB-SLAM2

ORB-SLAM2 builds upon ORB SLAM. The apparent difference is that ORB SLAM has only monocular modes of operations while the ORB-SLAM2 has RGB-D and Stereo modes along with the monocular mode [16]. The algorithm

*https://github.com/dorian3d/DBoW2
†https://github.com/RainerKuemmerle/g2o

also has improved the bag of words for the ORB vocabulary and full bundle adjustment after loop closing. It can also run on Robot Operating System. The improved bag of features allows ORB-SLAM2 to detect more key features than ORB-SLAM, which leads to better performance. Figure 2 shows the processes similar to ORB-SLAM in a single block and the different procedures in green blocks.

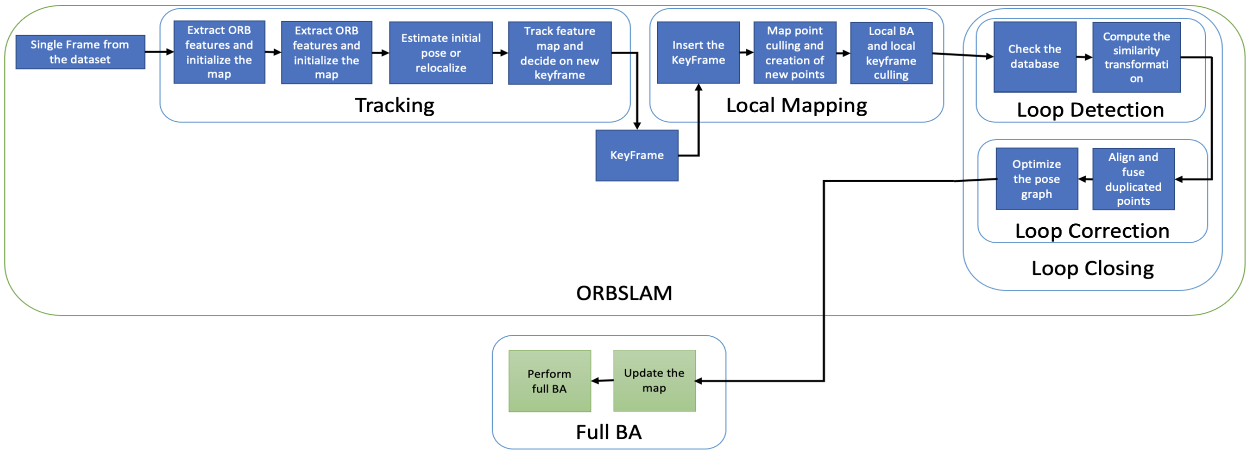

Single Frame from the dataset
Extract ORB features and initialize the map
Extract ORB features and initialize the map
Estimate initial pose or relocalize
Track feature map and decide on new keyframe
Tracking
KeyFrame
Insert the KeyFrame
Map point culling and creation of new points
Local BA and local keyframe culling
Local Mapping
Check the database
Compute the similarity transformati on
Loop Detection
Optimize the pose graph
Align and fuse duplicated points
Loop Correction
Loop Closing
ORBSLAM
Perform full BA
Update the map
Full BA

Fig. 2 Block Diagram of ORB-SLAM2 with its four parallel threads - tracking, local mapping, loop closing and full bundle adjustment

C. ORB-SLAM3

ORB-SLAM3 builds on ORB-SLAM2 with a significant difference in the concept of relocalization. This algorithm can relocalize quickly even when poor visual information is available after the tracking is lost [17]. It is helpful when the flight is taking a swift turn. It also requires IMU values along with the frame inputs. Unlike ORB-SLAM, which assumes all camera components to be pin-hole models, ORB-SLAM3 also offers the Kannala-Brandt [18] fish-eye model.

D. LOAM

After each sweep, the points received from the LiDAR scan register to the point cloud. LiDAR odometry processes the combined point cloud after a few sweeps, where the movement of the LiDAR between two consecutive sweeps computes the estimated motion to correct any distortion. Then the output is sent to LiDAR mapping for matching and registering the undistorted cloud onto the map at a frequency of 1Hz (as shown in Fig.3). Transforming and integrating the LiDAR odometry and LiDAR mapping yields an output that runs at 10 Hz. The advantage of using LiDAR is that it is insensitive to ambient lighting and optical texture in the scene [10], whereas the camera turns sightless to extreme brightness and darkness.

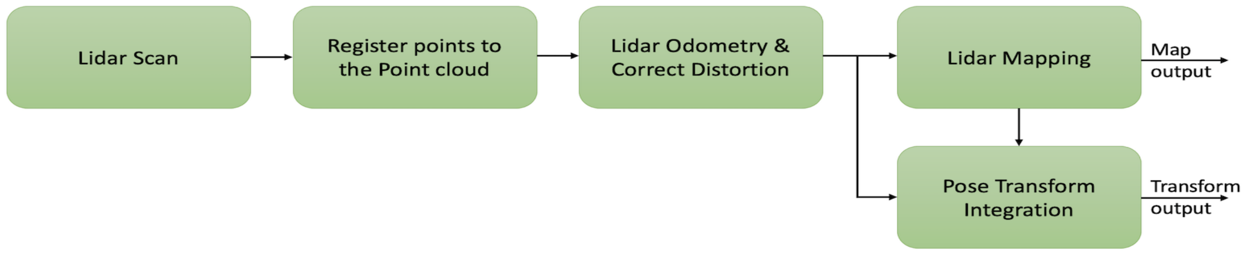

Lidar Scan
Register points to the Point cloud
Lidar Odometry & Correct Distortion
Lidar Mapping
Map output
Pose Transform Integration
Transform output

Fig. 3 Block Diagram of the LOAM software

E. A-LOAM

The Advanced LOAM (A-LOAM[‡]) is a simple and clean implementation of LOAM with less complicated mathematical derivations and operations. It uses Eigen (C++ template library for linear algebra) and Ceres Solver (C++ library for modeling and solving significant, complex optimization problems) to enhance the code.

F. F-LOAM

The Fast LOAM (F-LOAM) is an optimized version developed from LOAM and Advanced LOAM (A-LOAM), with the computational cost reduced by a factor of three. The LOAM and A-LOAM algorithms use iterative methods for scan-to-scan match and scan-to-map. Still, F-LOAM uses a non-iterative two-stage, computationally inexpensive, and accurate process. During each scan, the edge and planar features are extracted and mapped with the local edge map and local plane map, respectively [19].

III. Experiment Setup

This section describes the flight test setup and software used to run the SLAM solutions.

A. NASA ARC Flight Test Setup

The goal of the ARC flight test campaign was to obtain high-definition RGB camera images and 3D scanning LiDAR data to support multiple studies. These studies include airborne autonomy for hazard perception, safe beyond-visual-line-of-sight operations, GPS-free navigation, and real-time mapping. In this campaign conducted in November-December 2021, the dataset was collected at a low altitude of <100m, targeting different suburban/urban structures of interest, including roads, buildings, trees, people, vehicles, and vegetation. The payload had Velodyne VLP16 3D Scanning LiDAR, Xsens MTI-20 VRU IMU, Intel NUC i7 CPU, a FLIR DUO-R Thermal IR, and 4K Visual Camera, along with a GoPro Max camera as the ground sensor. Figure 4(a) shows the block diagram of how the sensors and components connect in the payload, and Fig.5(b) is the location where the payload mounts on the flight. Subsection V.A mentions the different routes taken.

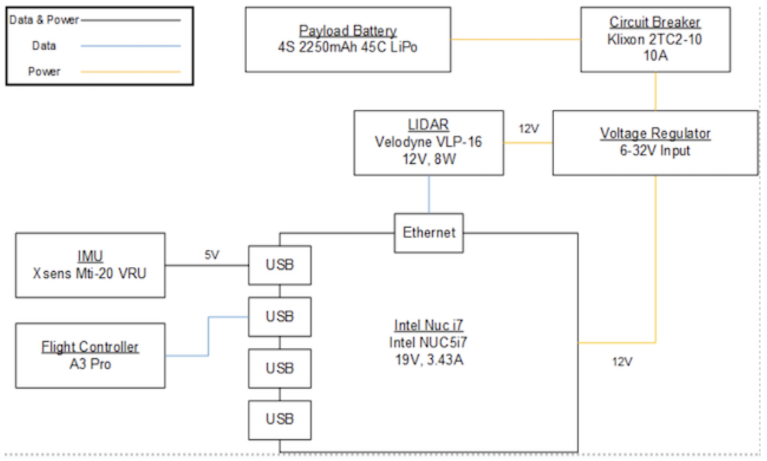
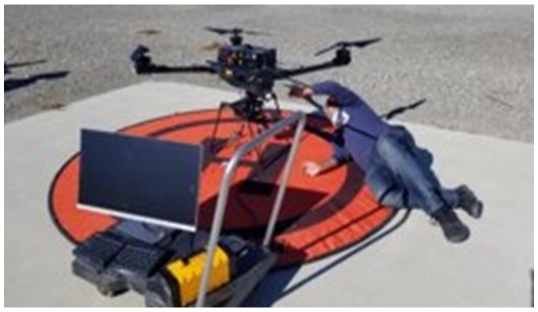


Fig. 4 ARC flight payload block diagram

B. NASA AFRC Flight Test Setup

The NASA Armstrong sUAS Lab conducted a flight test campaign using a Freefly Alta8 multicopter from March to June 2022. The aircraft collected 4K-resolution 60 FPS video following a trajectory resembling a notional eVTOL 9-degree approach and landing at the helipad. The camera is a RED DSMC2 mounted on a Movi Pro gimbal. In addition to the video, it logged the data from the Pixhawk autopilot and Movi Pro. Nearby weather stations recorded meteorological conditions, augmented by an ASC (vendor – Atmospheric Systems Corp.) 4000 (i.e., 4000 Hz frequency

[‡]https://github.com/HKUST-Aerial-Robotics/A-LOAM


(a)

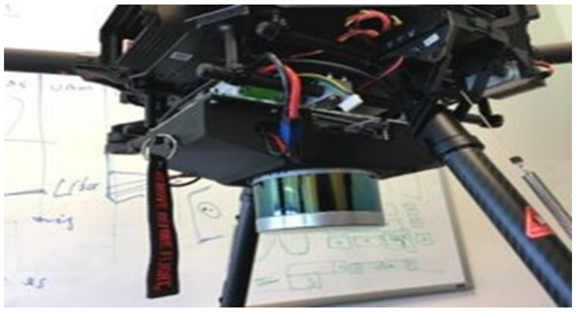
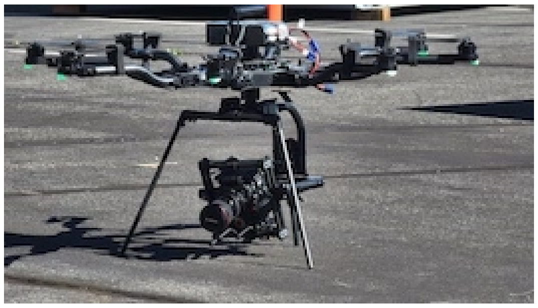

(b)

Fig. 5 ARC flight images (a) The flight in its landing/takeoff pad with the team calibrating (b) The payload with the sensors and the NUC i7 CPU

band) mini SODAR (sound detection and ranging) monitoring 3D winds in the boundary layer. As mentioned earlier, subsection V.A talks about the flight routes.


(a)

(b)

Fig. 6 AFRC flight images (a) The flight and payload with the sensors (b) The flight in its landing zone

C. Software Setup

For the software setup, a virtual machine on a Mac Book Pro and an Intel NUC with Kubuntu runs Ubuntu 18.04, 16.04, and 20.04. These machines drive the SLAM algorithms, namely ORB-SLAM2, LOAM, F-LOAM, and A-LOAM setup, based on the instructions provided in their respective official GitHub sites. In the ORB-SLAM2 pre-processing phase, the dataset videos had to be converted into images to run through the algorithms. So, it utilizes a custom python code to convert the video at 30-60fps and to snip only the exciting part of the video for the analysis. It is also necessary to auto-generate a text file with the image file name and the corresponding time using a python script during the conversion process.

IV. Understanding the SLAM Outputs

Figure 7 is the output window that pops up when the ORB-SLAM2 code executes. The window to the right, referred to as Map Viewer, displays the keyframes, key features, and even the previously mapped points. The red dots mean reference map points, the black dots denote all the identified map points, and the blue squares represent the keyframes, i.e., frames with comparatively unique data. The green line shows the mapped trajectory, and the green box or envelope implies the current position\location of the camera frame. The window on the left presents the actual video with the current frame in the picture with all the detected feature points as tiny green boxes.

Executing the LiDAR SLAM algorithm opens a ROS visualization window called RViz, as shown in Fig.8(a). The sidebar in the left column has several settings that change the display's appearance. The rainbow-colored dots are the

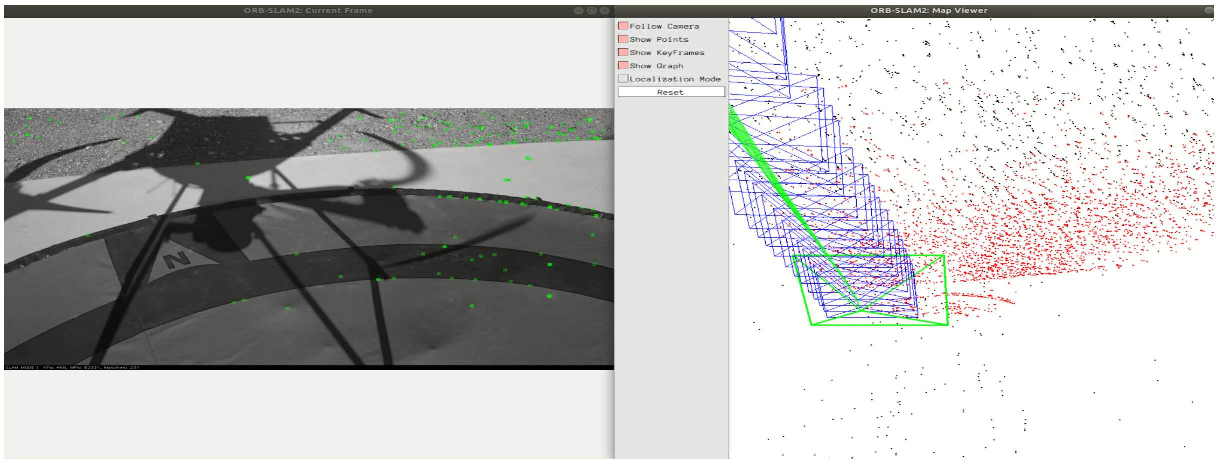


Fig. 7 Sample Output from ORB-SLAM2

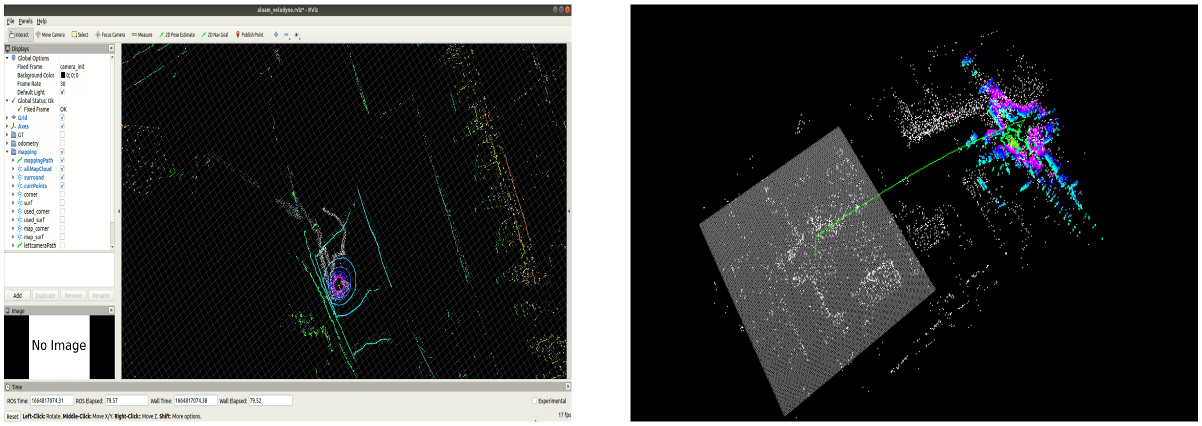


Fig. 8 Sample output from A-LOAM

points seen in the current LiDAR sweep or scan. The white dots are the dots from the point cloud, i.e., points from the old sweeps. The green line in Fig.8(b) is the trajectory marked by A-LOAM. The square plane is a grid used to mark the ground along the axis. The keyframe trajectory text file[§] that saves the trajectory information contains every single pose with its

- timestamp - A float datatype provides the time in seconds since the Unix epoch
- tx ty tz - Three float values give the position of the optical center of the color camera with respect to the world origin as per the motion capture system
- qx qy qz qw - Four float values offer the orientation of the optical center of the color camera in the form of a unit quaternion (following vector-first formalism) with respect to the world origin as per the motion capture system

Similarly, in the output text file obtained from A-LOAM's output rostopic, the timestamp is saved to its nanoseconds with the position (x,y,z) and orientation (x,y,z,w).

V. Results

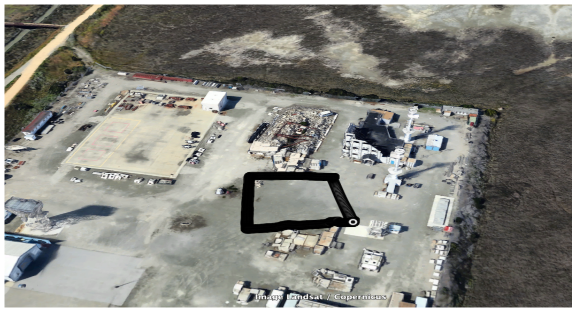
A. Dataset Description

Table 1 highlights the different datasets for the SLAM analysis, their routes, and descriptions. The trajectory gives each dataset the reference names used in the upcoming sections.

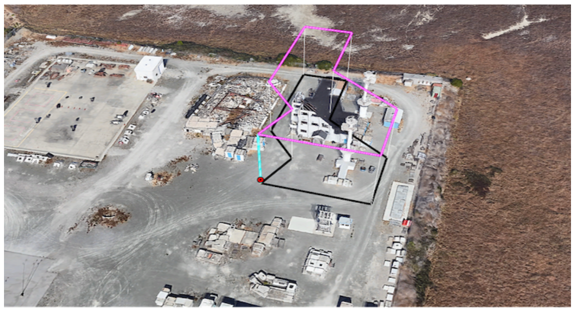
[§]https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats

Dataset Name	Trajectory\Routes	Altitude (m AGL)	Ground Speed (m/s)
DART_SQ	Takes a simple square path around the DART region in NASA Ames as shown in Fig.9(a)	30	2
DART_ACTUAL	Takes a path around a building in NASA Ames DART site Fig.9(b)	30	2
NFAC	A route near the National Full-Scale Aerodynamics Complex Fig.9(c)	30	2
DFM_1	A straight route in the Defrance Mid Road starting from point A to B like in the Fig.9(d)	30	2
DFM_2	A straight route in the Defrance Mid Road starting from point B to A (vice versa of DFM_1) Fig.9(d)	30	2
AFRC	A route in the Edwards Airforce center Fig.9(e)	460	3
AFRC_CONES	A route in the Edwards Airforce center with cones around the landing site Fig.9(e)	460	3

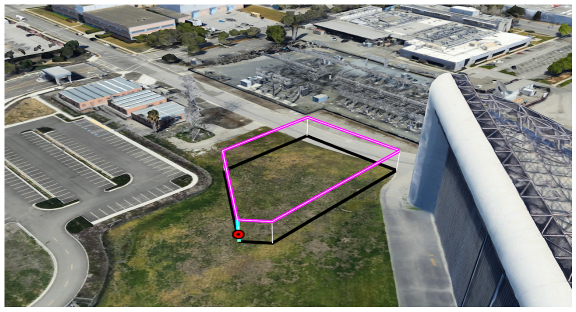
Table 1 Information of the different routes



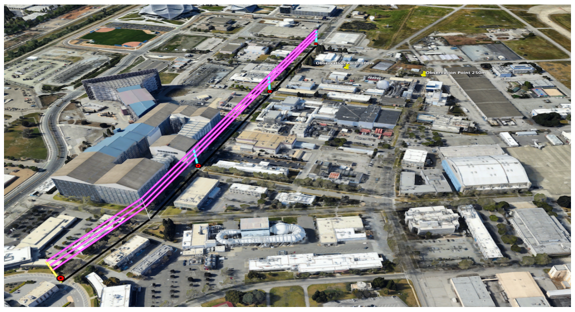
(a) DART_SQ



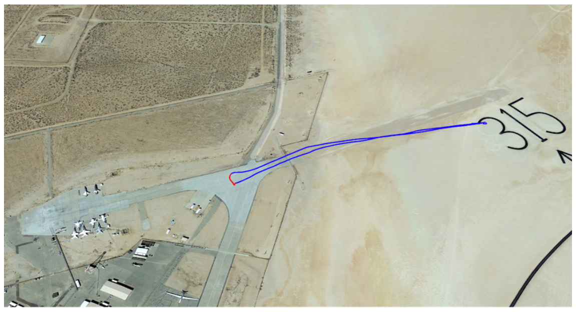
(b) DART_ACTUAL



(c) NFAC



(d) DFM_1 and DFM_2

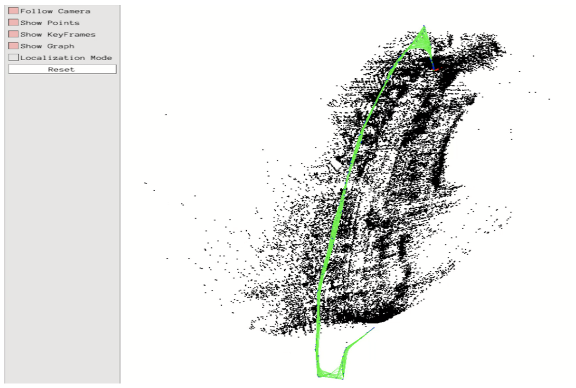


(e) AFRC and AFRC_CONES

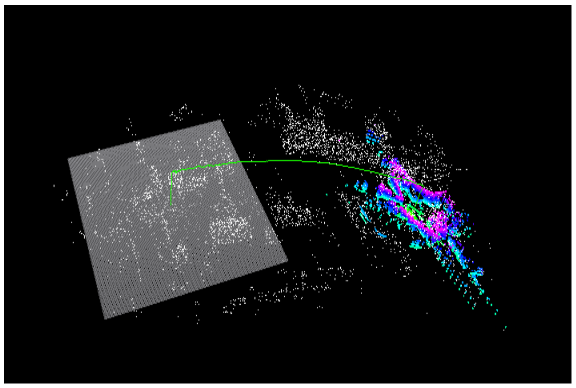
Fig. 9 Routes taken by the different flights

B. DFM_1 Dataset

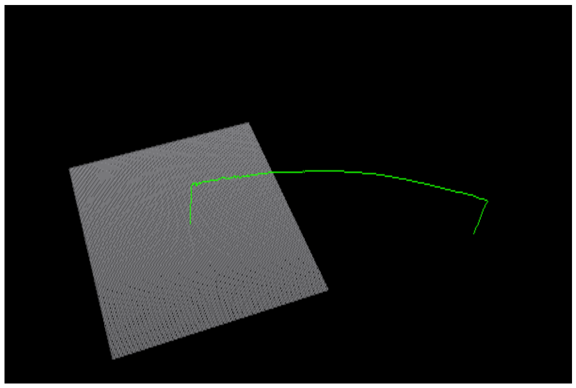
In Fig.10, (a) gives the output from the ORB-SLAM2, (b) and (c) the outcomes from A-LOAM, and (d) shows the initial result from F-LOAM. The route can be seen in these images but looks a little warped because it can be challenging to output absolute measurements with only one monocular camera without supervised calibration. One could think of initialization by doing a small translation of the camera or using IMU calibration like in ORB-SLAM3, or Lego-LOAM[¶] [20]. Using such algorithms that use inertial SLAM can help overcome warping. However, it is essential to overcome such distortions as it leads to inaccurate SLAM solutions causing higher estimation errors.



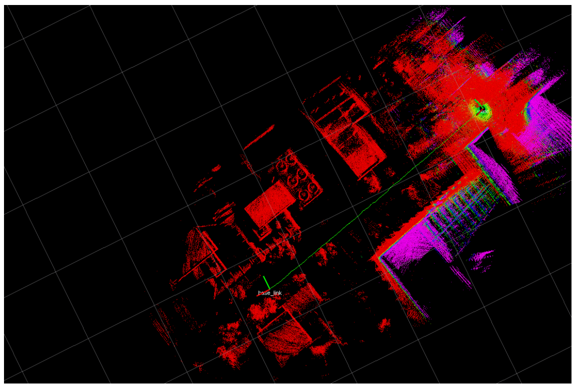
(a)



(b)



(c)



(d)

Fig. 10 SLAM results for the DFM_1 dataset

C. DFM_2 Dataset

Figure 11(a) displays the output from running the dataset with ORB-SLAM2, and Fig.11 (b),(c) shows the A-LOAM algorithm outputs. Since the route is opposite the DFM_1, the results look identical but still have minor differences.

D. NFAC Dataset

Figures 12(a) and (b) show the outputs from the A-LOAM, while (c) represent the Matlab plot of the true value from the flight's telemetry data and the A-LOAM result obtained. These images show how closely accurate the SLAM algorithms produce their results. Especially, Fig.12(b) shows how the LiDAR has captured the wind tunnel on the left of the image.

[¶]https://github.com/RobustFieldAutonomyLab/LeGO-LOAM

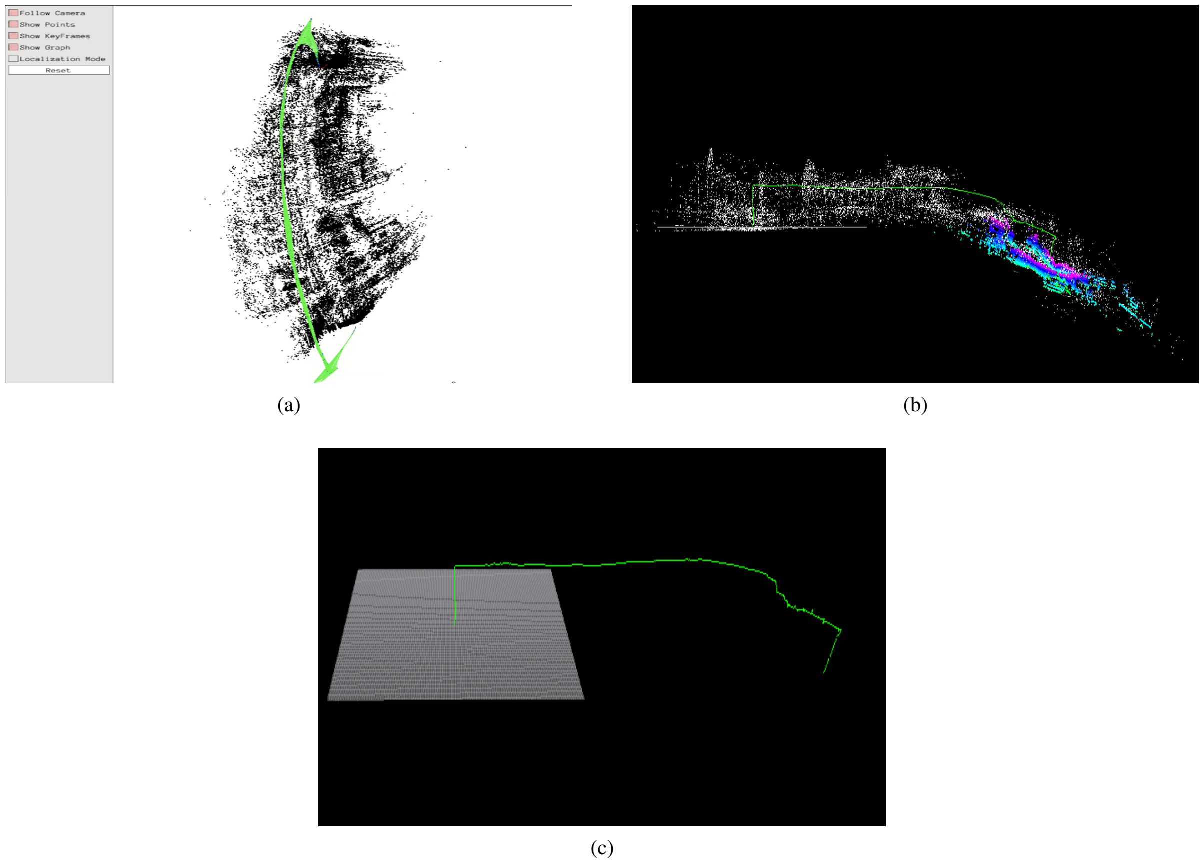


Fig. 11 SLAM results for the DFM_2 dataset

E. DART_SQ Dataset

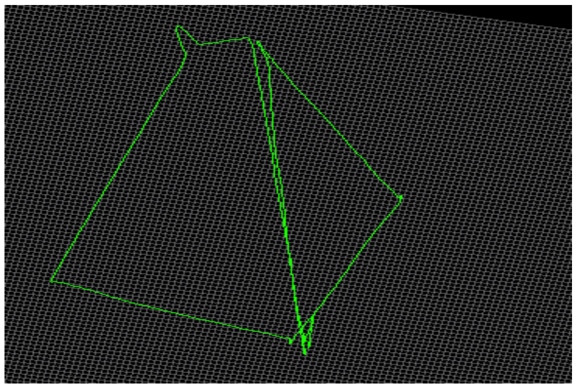
Figure 13 shows the incomplete trajectory of ORB-SLAM2. In the actual video, multiple swift turns and sudden jumps cause ORB-SLAM2 to lose track and cause discontinuities in the estimated trajectory. It usually relocalizes quickly, but in some cases, added complexity makes it challenging to get back on track. More analysis is in section VII of this paper.

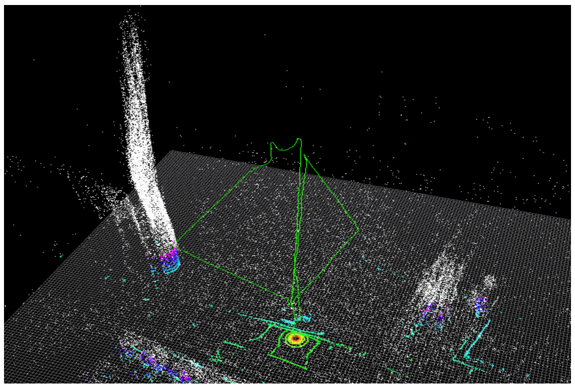
F. DART_ACTUAL Dataset

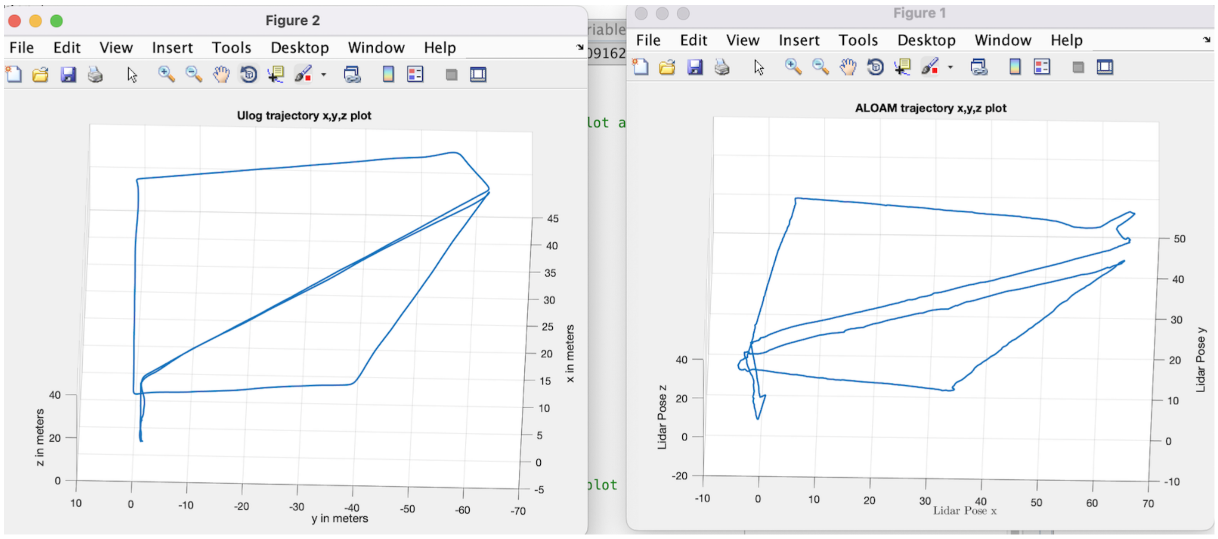
Unlike DART_SQ, DART_ACTUAL lacks visual perturbations such as jitter, sudden jumps, and swift turns, which provides smoother tracking. Figure 14(a) is how the trajectory looked from an ORB-SLAM2 display window versus Fig.14(b) shows the unscaled plot of the trajectory based on the output recorded from ORB-SLAM2. Figure 14(c) is the A-LOAM output of the rosbag recorded by the payload's Velodyne LiDAR. Due to unknown factors, part of the trajectory in the dataset does not appear appropriately near the landing phase. Future investigations will look into what caused the drop in data. Figure 14(d) is the F-LOAM output of the trimmed rosbag to avoid the unpredictable behavior of the SLAM due to the improper data.

G. AFRC Dataset

This dataset runs on both ORB-SLAM and ORB-SLAM2. One of the initial tests helped declare that ORB-SLAM2 was much more efficient than ORB-SLAM in these considered datasets. Figure 15(a) and (b) compares ORB-SLAM and ORB-SLAM2 such that ORB-SLAM2 yields better performance and more detected features due to an improved bag of features. ORB-SLAM's bag of features could not detect the helipad markings, leading to inaccurate estimation. Contrarily, ORB-SLAM2 detects numerous features associated with the helipad markings and has a more accurate estimate. Figure 15(c) Fig. 15c shows the output trajectory of ORB-SLAM, and Fig.15(d) shows the unscaled position


(a)


(b)



(c)

Fig. 12 SLAM results for the NFAC dataset

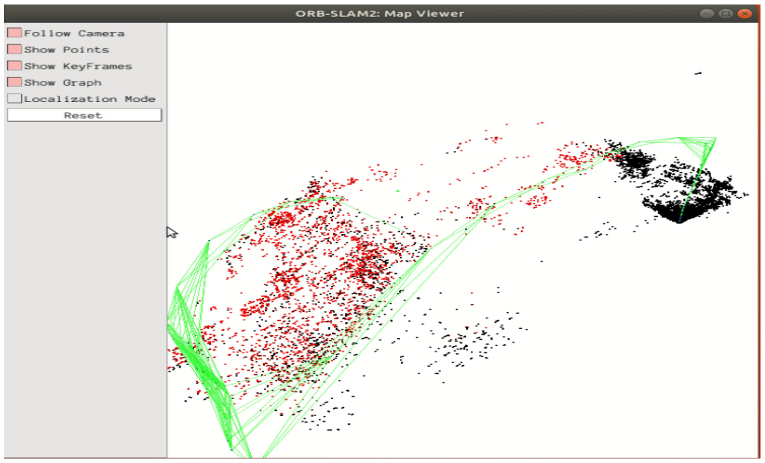
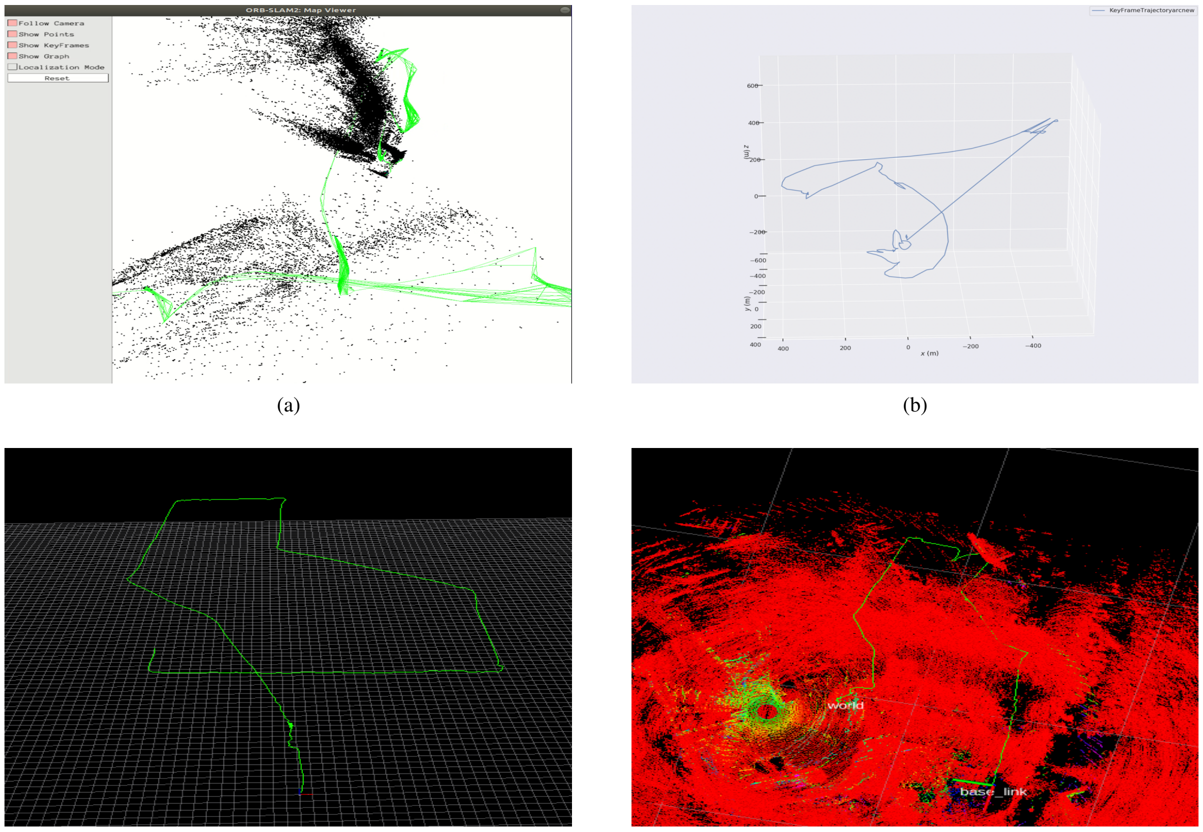


Fig. 13 ORB-SLAM2 output of the DART_SQ dataset



(a) (b) (c) (d)

Fig. 14 SLAM results for the DART_ACTUAL dataset

of ORB-SLAM2, which demonstrates a better estimation of the trajectory.

H. AFRC_CONES Dataset

Though the previous dataset yielded promising results, adding distributed cones as fiducials (mock landing lights) to the helipad region gives even better results. Based on the preliminary analysis, the SLAM solutions with distributed fiducials and higher-quality video seem to perform better than those without distributed fiducials. Figure 16(a) shows the ORB-SLAM2 output as seen in its viewer window, and Fig.16 (b) show the tracked features during landing. ORB-SLAM2 detects the cones at a higher altitude because of the higher quality video and the visual prominence of the cones compared to the background. These SLAM solutions demonstrate encouraging results, but future flight tests and datasets will provide more insight into distributed sensing for AAM concepts and operations[11].

VI. Error Analysis

This section is about the method of comparison and calculation between the SLAM and the true values, i.e., telemetry data. For example, the normalized output values obtained from ORB-SLAM2 range from -1 to 1, but the true values, i.e., the values obtained from the flight sensors like Pixhawk, are much higher. Similarly, the LiDAR SLAM outputs are on a different scale than the telemetry values. This scale difference poses a scaling problem, and the SLAM outputs must convert to actual values. The relation to scale these values is from the general equation for a straight-line, i.e., y-intercept form,

$$y = mx + b \tag{1}$$

where m is the slope, and b is the y-intercept. In this SLAM context, the scaled data is,

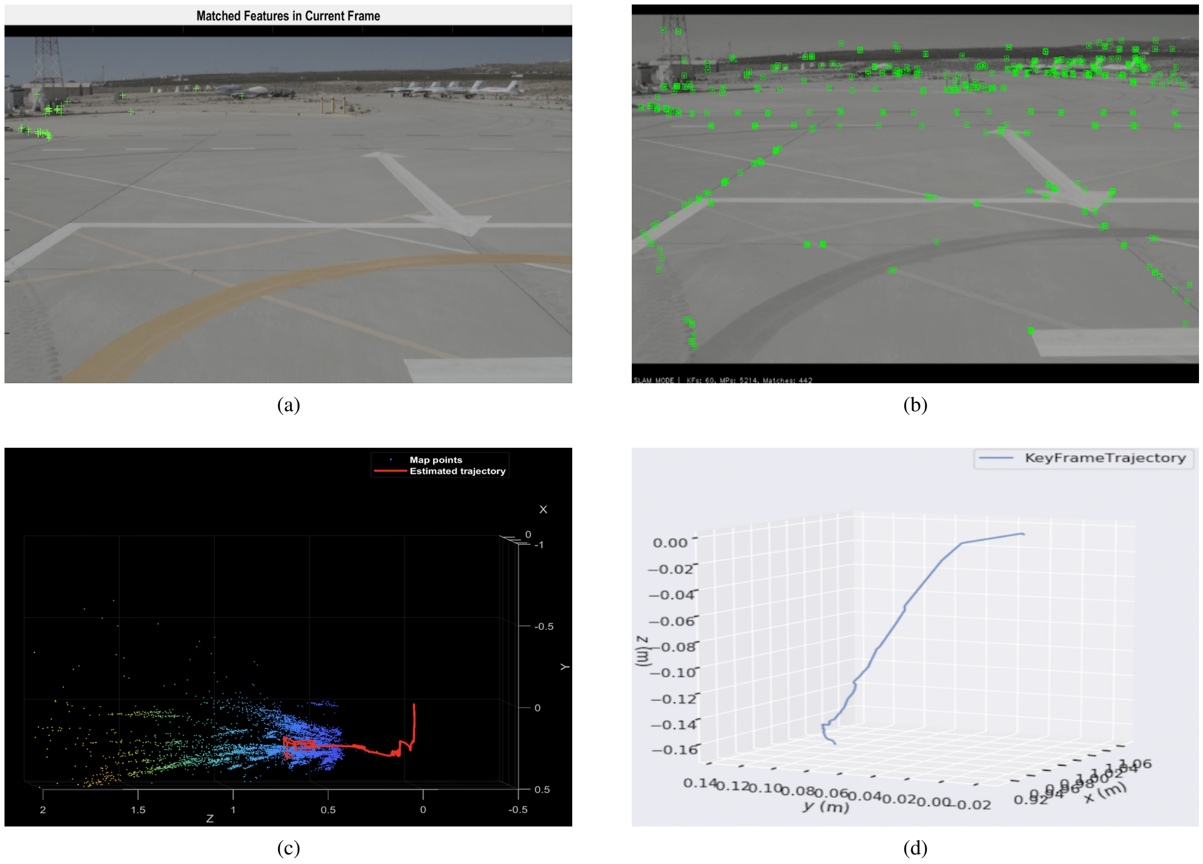


Fig. 15 SLAM results for the AFRC dataset

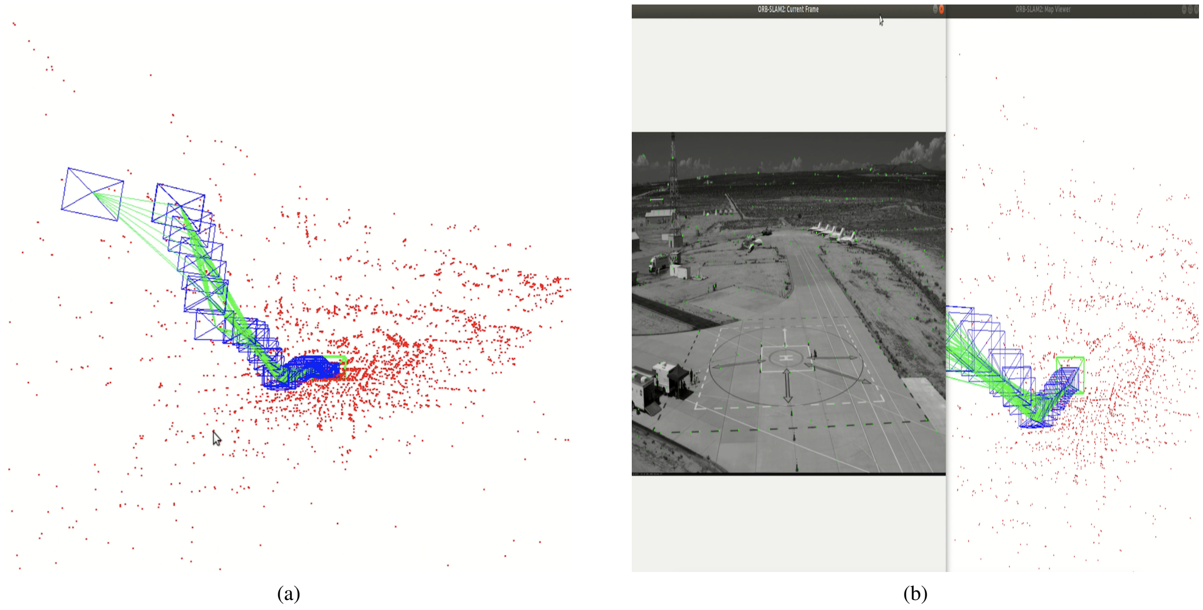


Fig. 16 SLAM results for the AFRC_CONES dataset

$$P_{slam_{scaled}} = K \cdot P_{slam_{unscaled}} + T \tag{2}$$

Again, K is the slope, and T is the y-intercept. Then LP Simplex in Excel Optimizer solves the slope 'm' and the intercept 'b' values. The three equations after rewriting this further as their x,y, and z components are,

$$\begin{aligned} P_{slam_{scaled_x}} &= (K_{x_{slam}} \cdot P_{slam_{unscaled_x}}) + T_{x_{slam}} \\ P_{slam_{scaled_y}} &= (K_{y_{slam}} \cdot P_{slam_{unscaled_y}}) + T_{y_{slam}} \\ P_{slam_{scaled_z}} &= (K_{z_{slam}} \cdot P_{slam_{unscaled_z}}) + T_{z_{slam}} \end{aligned} \tag{3}$$

where, $P_{slam_{scaled_x}}$, $P_{slam_{scaled_y}}$, $P_{slam_{scaled_z}}$ are the scaled SLAM values of x, y, z respectively. $P_{slam_{unscaled_x}}$, $P_{slam_{unscaled_y}}$, $P_{slam_{unscaled_z}}$ are the unscaled SLAM values of x, y, z respectively, and $K_{x_{slam}}$, $K_{y_{slam}}$, $K_{z_{slam}}$, $T_{x_{slam}}$, $T_{y_{slam}}$, $T_{z_{slam}}$ are the constant values obtained from the Excel Solver/Optimizer.

Since the values obtained from SLAM are discrete, the overall error formula is,

$$\varepsilon \approxeq \sum_{i=0}^{N-2} [(P_{slam_{scaled_{(i)}}} - P_{ph_{(i)}})^2 (t(i+1) - t(i))] \tag{4}$$

where N is the total number of entries in the SLAM output, and $P_{ph_{(i)}}$ are the true values from the telemetry data. Further rewriting the equation into x,y,z components with $(t(i+1) - t(i))$ as Δt gives,

$$\begin{aligned} \epsilon_x &\approxeq \sum_{i=0}^{N-2} (P_{slam_{scaled_{(i)}}} - P_{ph_{(i)}})^2 \Delta t \\ \epsilon_y &\approxeq \sum_{i=0}^{N-2} (P_{slam_{scaled_{(i)}}} - P_{ph_{(i)}})^2 \Delta t \\ \epsilon_z &\approxeq \sum_{i=0}^{N-2} (P_{slam_{scaled_{(i)}}} - P_{ph_{(i)}})^2 \Delta t \end{aligned} \tag{5}$$

As mentioned, ORB-SLAM2 does not generate keyframes constantly, but the overall solution has a regular time increment. New keyframes occur after reaching a certain threshold of detected feature points. The SLAM and telemetry datasets do not have time synchronization automatically, so manual time synchronization is required. Future work involves investigating an automated time synchronization method between ORB-SLAM2 and the telemetry data.

Following Eqn.3 and Eqn.5 gives multiple plots for each SLAM output for different datasets. In all of the plots seen in appendix A, (a), (b), and (c) are all comparisons between the true x, y, and z values with their corresponding scaled x, y, z values (both in meters) using the Eqn.3 versus elapsed time in seconds. Similarly, all the (d) plots are the error comparison between ϵ_x, ϵ_y, and ϵ_z in the Eqn.5 versus the elapsed time in seconds.

VII. SLAM Result Discussion

ORB-SLAM2 loses track/features when the flight takes a quick turn, sudden jump, and dip. It also did not perform to the fullest on the datasets with jello effect [21] (rolling shutters cause wobbly distortion effects) is in the video. One such feature-losing scenario is in the DART_SQ dataset, a video from the NASA STEReO Team on the DART site following the route shown in Fig.9(a).

The video is 132 sec long, and conversion to images at 30fps gives a total of 3940 photos in the dataset. The video has a resolution of 1920x1080 and is in mp4 format, which is then converted into jpeg images using a python code. In this scenario, the flight first takes a left turn after take-off, performs three right turns to complete the route shown, and finally, a left turn to orient itself towards the landing site.

In the processing video (which is 10 min 51 sec), the first time a delay in tracking was when the flight took a right turn around the time 2 min 46 sec (as seen in Fig.17 (a)), but only at 2 min 58 sec it thoroughly detects the rest of the prominent features (as seen in Fig.17 (b)). There is a delay of 12 sec in the processing time, which equates to 5 sec in the actual video. Similarly, in the second right turn, the tracking gets delayed/lost at around 4 min 46 sec (Fig.18(a)), and only at 4 min 51 sec (Fig.18(b)) it detects the rest of the features. This delay counts to 5 seconds in the actual video time.

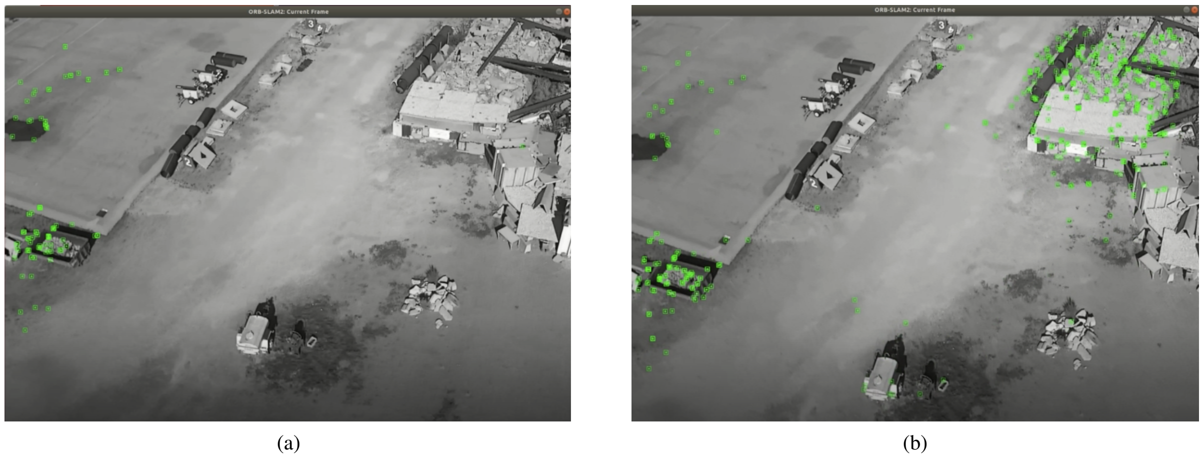


Fig. 17 Snapshots from the first right turn taken by the flight in the DART_SQ scenario

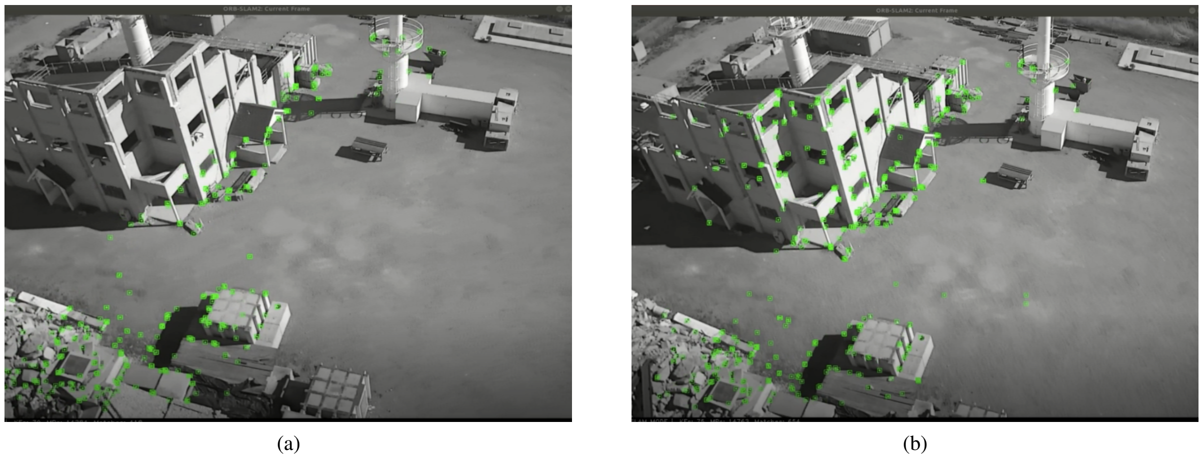


Fig. 18 Snapshots from the second right turn taken by the flight in the DART_SQ scenario

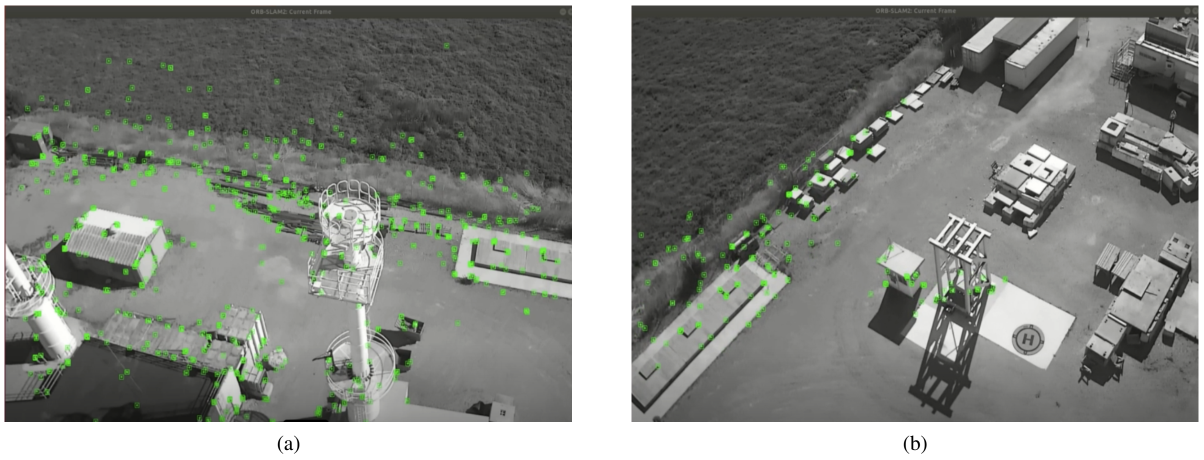
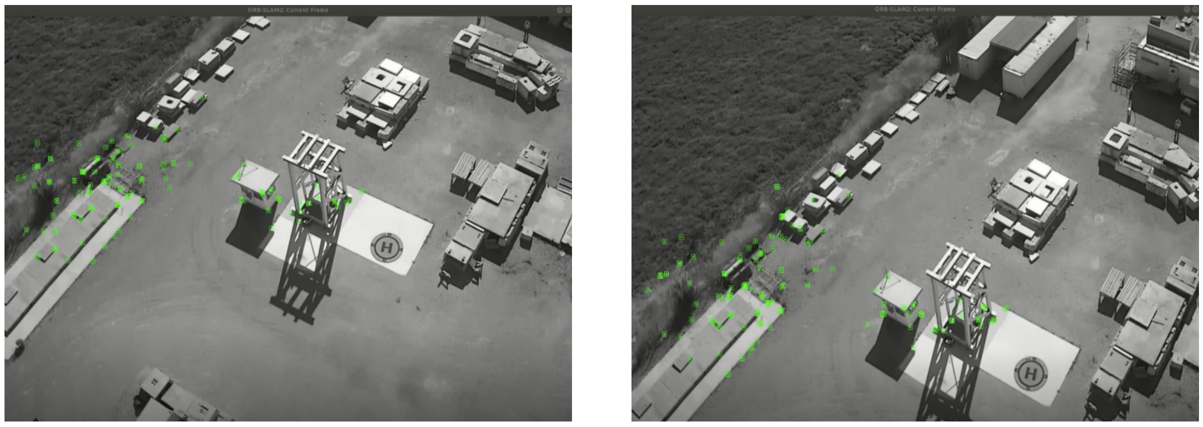


Fig. 19 Snapshots before and during the final right turn taken by the flight in the DART_SQ scenario

Next, at 7 min 23 sec just before the final right turn, it can be seen that the ORB-SLAM2 tracks most of the features in Fig.19(a). However, during the right turn at 7 min 33 sec, it can be seen (in Fig.19(b)) that most of the features are not detected, and only a few points to the left are acquired (just like in the first turn). The boxes/trailers in the top right corner are some of the distinct features which ORB-SLAM2 was supposed to identify but could not.



(a) (b)

Fig. 20 Snapshots from the final right turn taken by the flight in the DART_SQ scenario

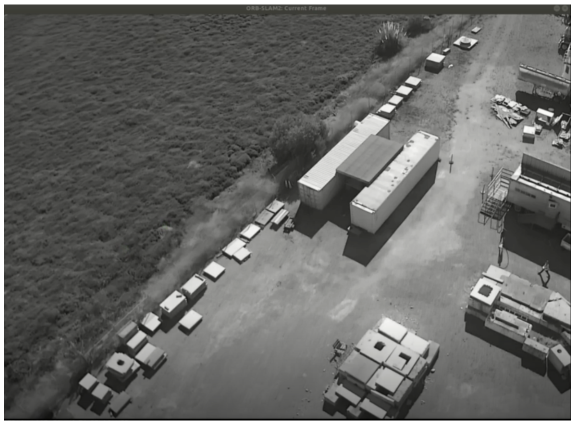
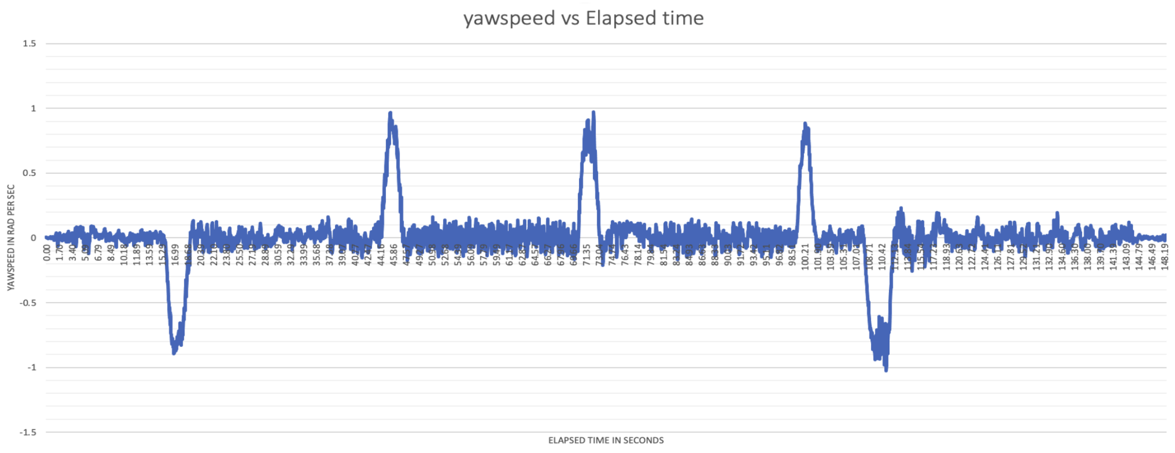


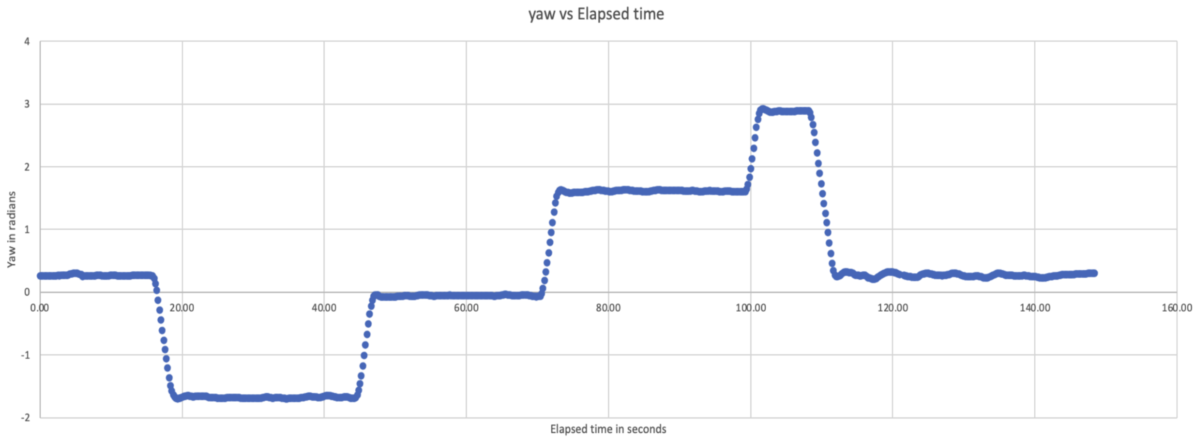
Fig. 21 Snapshot of when the sudden jump happens and ORB-SLAM2 loses all of its features

There is a dip (nose pointing to the ground) noticed at around 7 min 35 sec eliminating the boxes in the top right of the scene adds to the complexity, but it is still tracking the points from the previous frame, as seen in Fig.20(a). Right after the dip at 7 min 37 sec shown are the features still tracked in Fig.20(b), and that is when the sudden jump happens at 7 min 38 sec making ORB-SLAM2 lose all of the features (in Fig.21). An assumption is that the top right boxes needed to be in view much longer for ORB-SLAM2 to detect those as features.

The yaw and yaw rate plots from the recorded telemetry log files, as seen in Fig.22, help to understand how fast the UAS flight can turn. From these graphs and the initial dataset analysis, it seems the turn rates need to be less than 1 rad per sec for the right turns, or if the turns are faster than 1 rad per sec, then the scene needs to pause for at least 4-5 seconds to acquire all the features ultimately. From the hardware perspective, since the camera is angled 30 degrees downwards, the view at 30 m is better than at 10 m because it has more features in its field of view. Likewise, Velodyne LiDAR rotates 360 degrees and can perform better at 10-30 m rather than higher altitudes since it must maintain a visual reference to the ground plane.



(a)



(b)

Fig. 22 Yaw and yaw speed vs time plot

VIII. Conclusion & Summary

The analysis of the off-the-shelf pre-canned SLAM algorithms using the NASA flight test data and the preliminary comparison between these visual versus LiDAR SLAM algorithms gives a better understanding of SLAM algorithms for flights. In addition, the results compared with the UAS telemetry data provide a fundamental review of the deviation/accuracy in the SLAM outputs. In conclusion, based on the currently available dataset, having numerous distributed key features and landmarks in the scenery for the AAM approach and landing and other AAM concepts will help generate accurate SLAM solutions by maintaining keyframes with these distributed features and landmarks. Furthermore, in the LiDAR SLAM and vSLAM's initial error analysis, the LiDAR SLAM looks more accurate and robust to lighting and weather changes, yielding better estimation results. In this preliminary study, various challenges addressed will be learning lessons for the upcoming flights.

Acknowledgments

The authors would like to thank NASA Ames Research Center's Smart Mobility (SM) team led by Code AT, TTT/P&DS led by Code TI, the Scalable Traffic Management for Emergency Response Operations STEReO (STEReO) project team led by NASA Code TH and Code JO for their support in the flight testing. The authors also thank the NASA Armstrong Flight Research Center Dale Reed Subscale Flight Research Laboratory for their flight test data.

References

[1] Leonard, J., and Durrant-Whyte, H., "Simultaneous map building and localization for an autonomous mobile robot," *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, 1991, pp. 1442–1447.

[2] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., and et al., "Past Present and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, Vol. 32, No. 6, 2016, pp. 1309–1332.

[3] Thrun, S., "Simultaneous localization and mapping," *Robotics and cognitive approaches to spatial mapping*, 2007, pp. 13–41.

[4] Billinghurst, M., Clark, A., and Lee, G., "A survey of augmented reality," *Foundations and Trends in Human–Computer Interaction 8*, 2015, pp. 73–272.

[5] Takleh, T. T. O., Bakar, N. A., Rahman, S. A., Hamzah, R., and Aziz, Z. A., "A brief survey on SLAM methods in autonomous vehicle," *International Journal of Engineering & Technology 7*, 2018, pp. 38–43.

[6] Singandhupe, A., and La, H. M., "A review of slam techniques and security in autonomous driving," *2019 third IEEE international conference on robotic computing (IRC)*, 2019, pp. 602–607.

[7] Filipenko, M., and Afanasyev, I., "Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment," *018 International Conference on Intelligent Systems (IS)*, 2018, pp. 400–407.

[8] Taketomi, T., Uchiyama, H., and Ikeda, S., "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications 9*, 2017, pp. 1–11.

[9] Zhou, F., Zhang, L., Deng, C., and Fan, X., "Improved Point-Line Feature Based Visual SLAM Method for Complex Environments," *Sensors 21*, 2021, p. 4604.

[10] Zhang, J., and Singh, S., "LOAM: Lidar Odometry and Mapping in Real-time," *In Robotics: Science and Systems*, Vol. 2, No. 9, 2014, pp. 1–9.

[11] Kawamura, E., Dolph, C., Kannan, K., Brown, N., Lombaerts, T., and Ippolito, C. A., "VSLAM and Vision-based Approach and Landing for Advanced Air Mobility," *AIAA SciTech 2023 Forum*, 2023.

[12] Gálvez-López, D., and Tardos, J. D., "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics 28*, 2012, pp. 1188–1197.

[13] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D., "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics 31*, 2015, pp. 1147–1163.

[14] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W., "g 2 o: A general framework for graph optimization," 2011, pp. 3607–3613.

[15] Strasdat, H., Montiel, J. M. M., and Davison, A. J., "Scale drift-aware large scale monocular SLAM," Robotics: Science and Systems (RSS), Zaragoza, Spain, 2010.

[16] Mur-Artal, R., and Tardós, J. D., "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics 33*, 2017, pp. 1255–1262.

[17] Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M., and Tardós, J. D., "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics 37*, 2021, pp. 1874–1890.

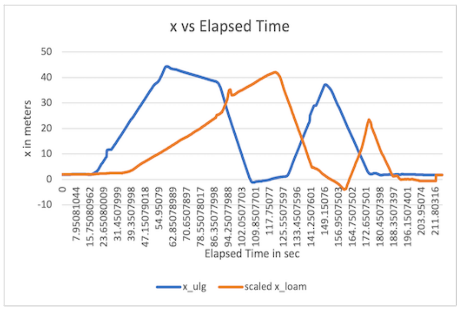
[18] Kannala, J., and Brandt, S. S., "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, No. 8, 2006, p. 1335–1340.

[19] Wang, H., Wang, C., Chen, C.-L., and Xie, L., "F-loam: Fast lidar odometry and mapping," *In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 4390–4396.

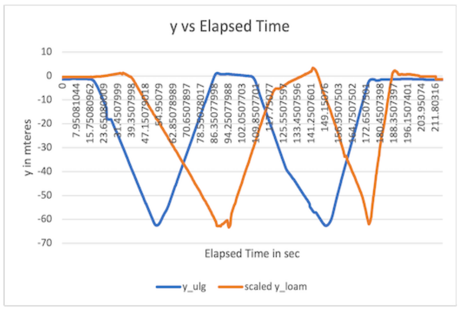
[20] Shan, T., and Englot, B., "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[21] Cai, Y., Lam, E., Howlett, T., and Cai, A., "Spatiotemporal analysis of "jello effect" in drone videos," *International Conference on Applied Human Factors and Ergonomics*, 2019, pp. 197–207.

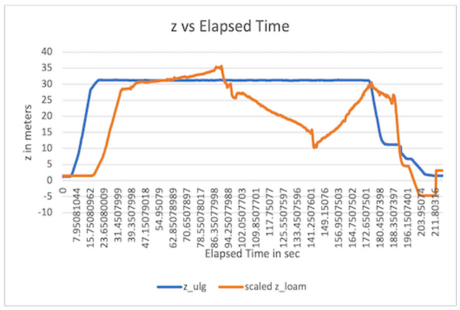
A. Error Analysis Plots



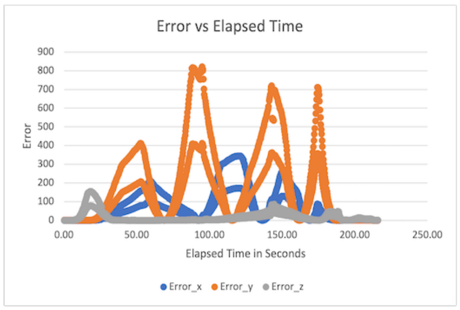
(a)



(b)

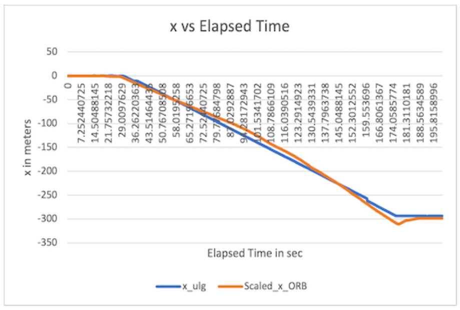


(c)

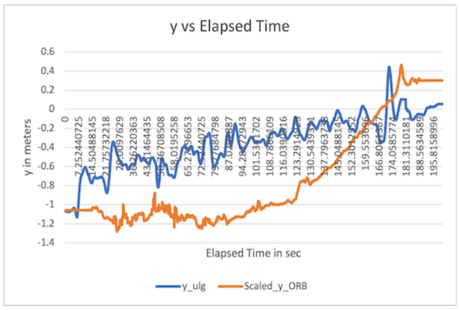


(d)

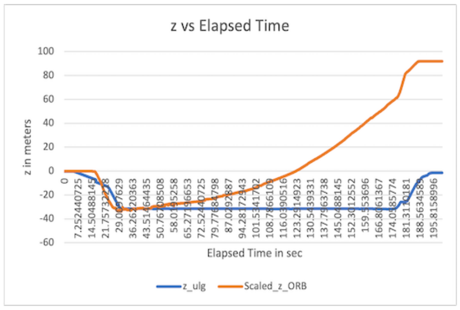
Fig. 1   x, y, z and the error versus the elapsed time plot for NFAC A-LOAM outputs



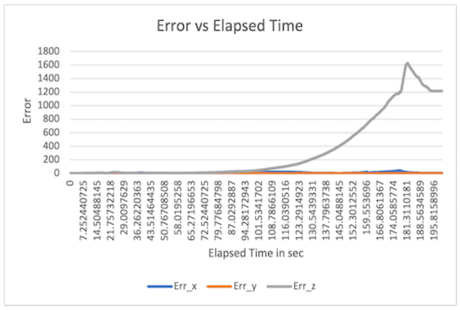
(a)



(b)

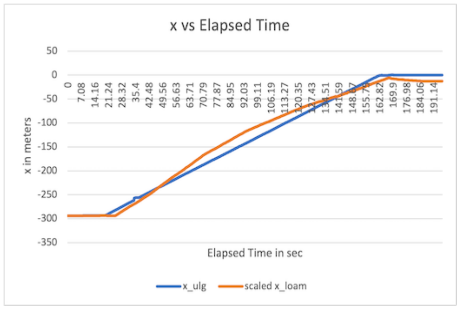


(c)

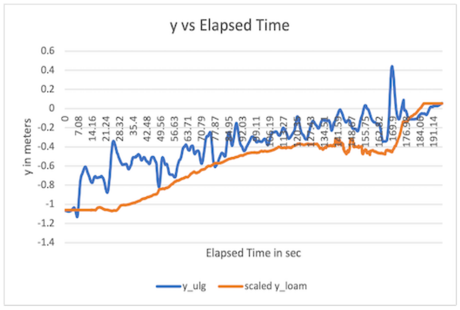


(d)

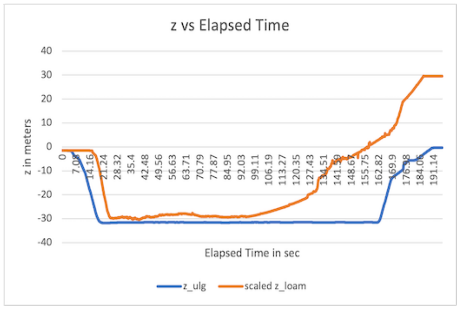
Fig. 2   x, y, z and the error versus the elapsed time plot for DFM_1 A-LOAM outputs



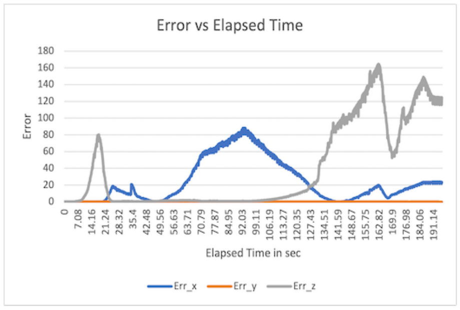
(a)



(b)

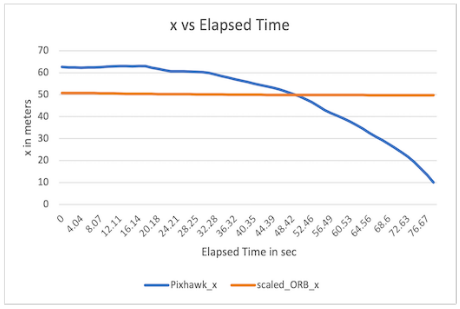


(c)

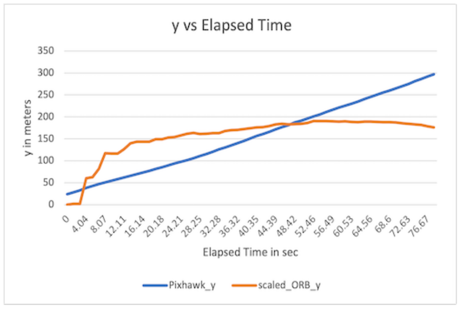


(d)

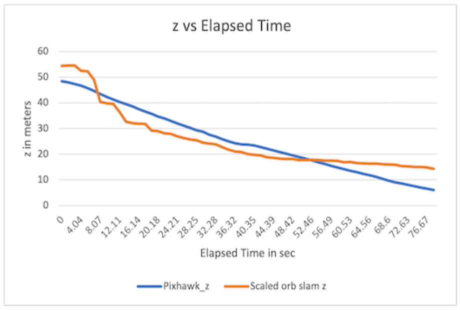
Fig. 3 x, y, z and the error versus the elapsed time plot for DFM_2 A-LOAM outputs



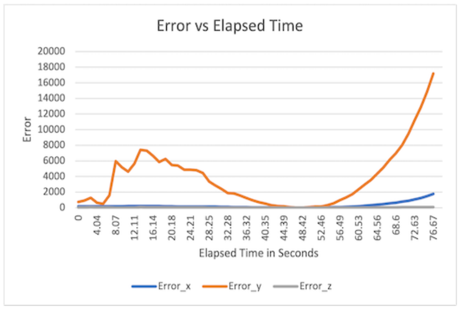
(a)



(b)



(c)



(d)

Fig. 4 x, y, z and the error versus the elapsed time plot for AFRC ORB-SLAM2 outputs

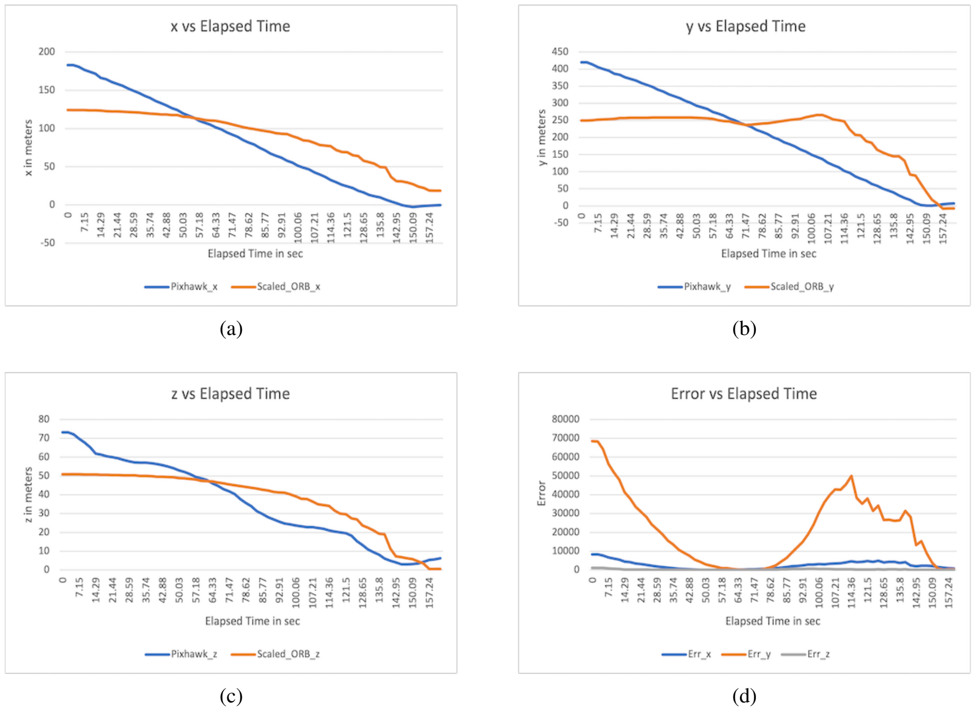


Fig. 5 x, y, z and the error versus the elapsed time plot for AFRC_CONES ORB-SLAM2 outputs

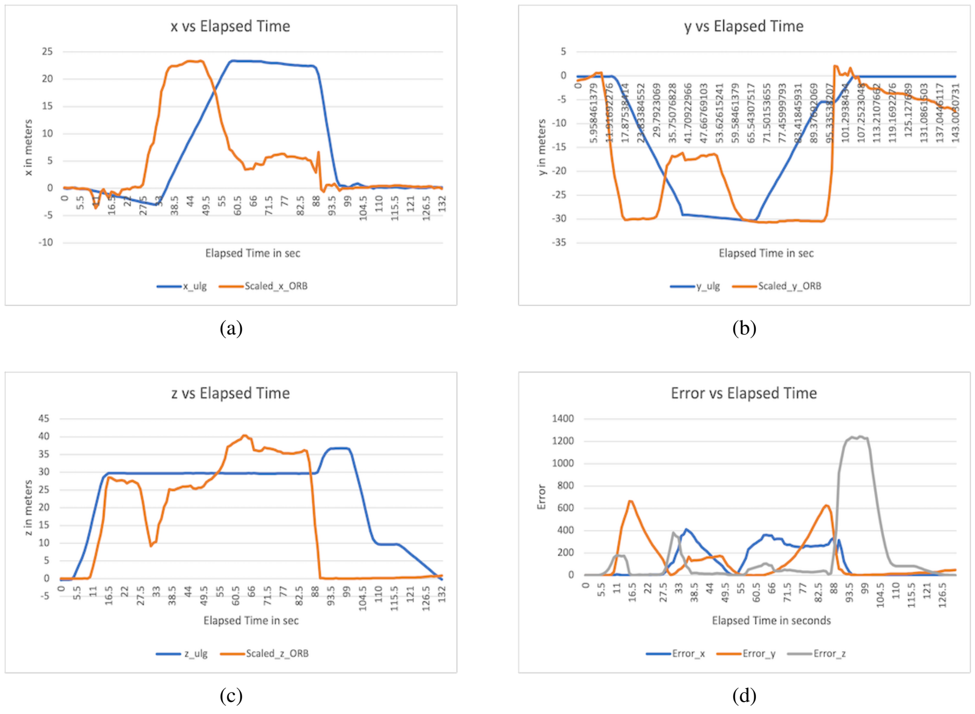
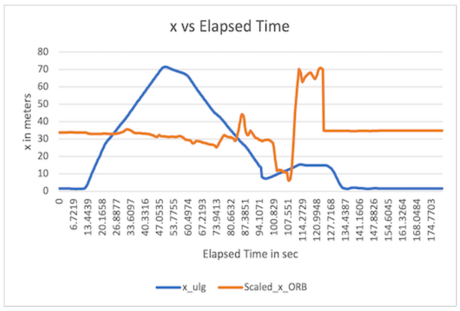
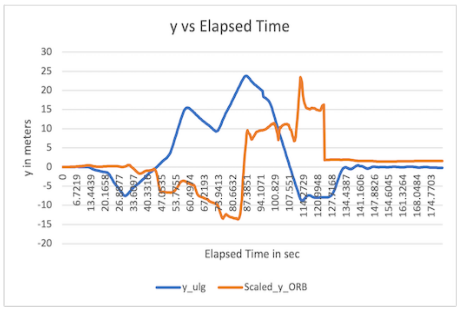


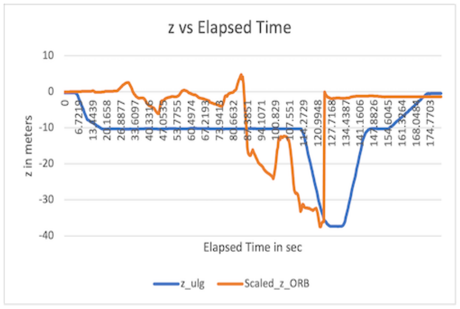
Fig. 6 x, y, z and the error versus the elapsed time plot for DART_SQ ORB-SLAM2 outputs



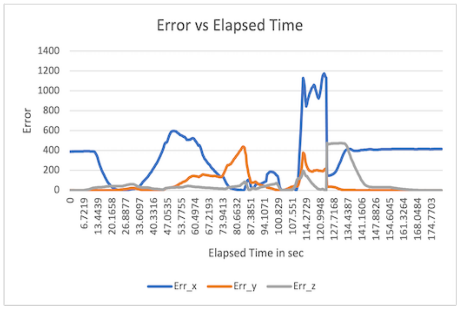
(a)



(b)

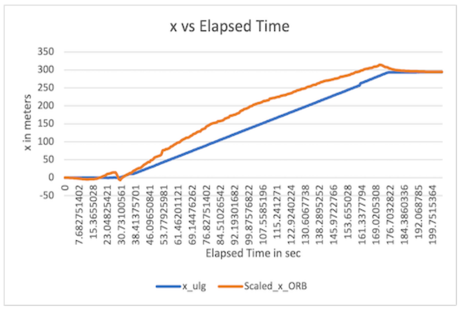


(c)

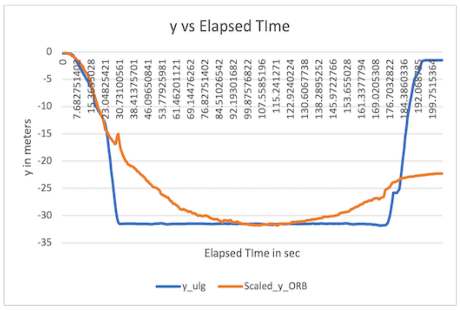


(d)

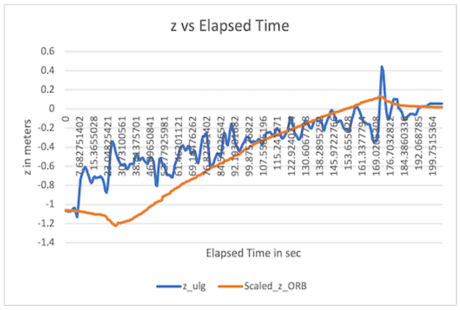
Fig. 7 x, y, z and the error versus the elapsed time plot for DART_ACTUAL ORB-SLAM2 outputs



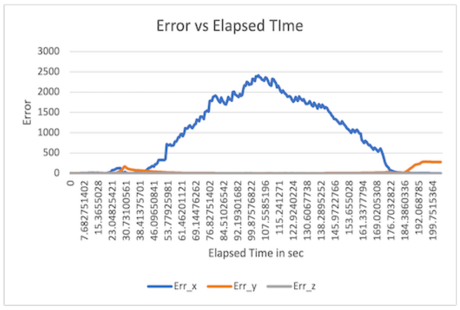
(a)



(b)

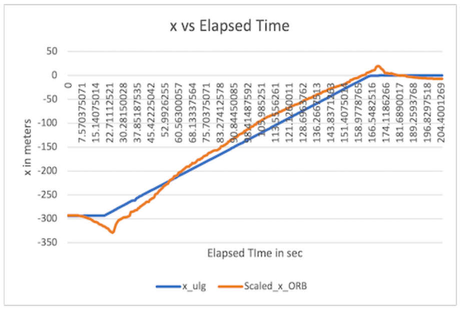


(c)

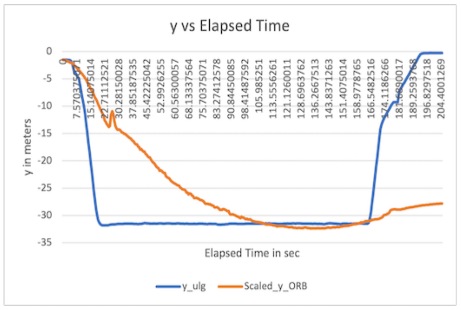


(d)

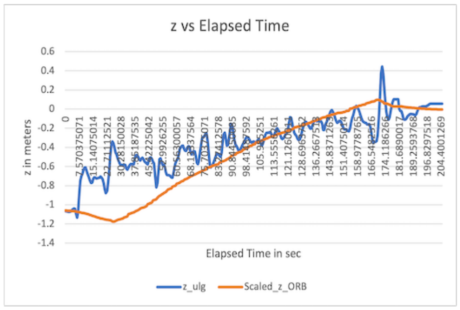
Fig. 8 x, y, z and the error versus the elapsed time plot for DFM_1 ORB-SLAM2 outputs



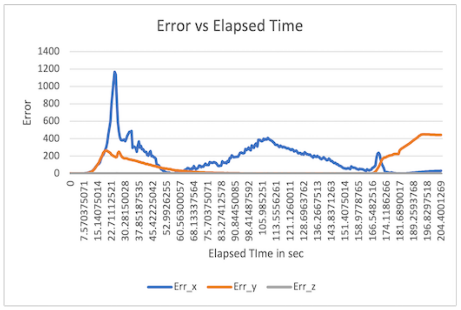
(a)



(b)



(c)



(d)

Fig. 9 x, y, z and the error versus the elapsed time plot for DFM_2 ORB-SLAM2 outputs