Evaluating SLAM Performance with Synthesized Datasets from Unreal-Based Emulators

Muchang Bahng *Duke University*

***Abstract*—The rapid advancement in visual-inertial simultaneous localization and mapping (SLAM) has opened up numerous applications in computer vision. However, the scarcity of high-quality, publicly accessible datasets hampers the evaluation of SLAM performance in varied and tailored environments. In this study, I employ the AirSim simulator and the Unreal game engine to generate a trajectory resembling that of the *TUM VI Room 1* ground truth dataset within the *ArchViz* indoor environment representing a well-lit, furnished room. I further modify the environment and trajectory through various expansions, addition of features, and data smoothing to ensure a more stable sequence of input frames into the SLAM architecture. I then examine the efficiency of visual ORB-SLAM3 by inputting images of resolution 256×144 and 512×288 at 30 frames per second (FPS), while also adjusting the feature threshold, which refers to the maximum number of feature points that ORB-SLAM3 tracks per frame. A thorough investigation of the camera parameters within AirSim and ORB-SLAM3 has led to the essential finding that the resolution of the input images must coincide with the dimensions of the film. The subsequent runs under these variables reveal that higher resolution images lead to considerably better tracking, with an optimal feature threshold ranging between $3000 \sim 12000$ feature points per frame. Moreover, ORB-SLAM3 demonstrates significantly enhanced robustness within dynamic environments containing moving objects when using higher resolution inputs, with an increased error of close to 0cm compared to 23.19cm for lower resolutions (averaged over three runs). Finally, I conduct qualitative testing using real-life indoor environments recorded with an iPhone Xr camera, which produces results that highlight the challenges faced by ORB-SLAM3 due to factors such as glare and motion blur.**

***Index Terms*—Visual simultaneous localization and mapping, Unreal, AirSim, ORB-SLAM3, Feature threshold**

I. Introduction

Simultaneous localization and mapping (SLAM) is an essential component in robotics and computer vision systems, which involves constructing a map of an unknown environment while simultaneously locating the robot or a camera within that environment. SLAM algorithms typically use a combination of sensor measurements and motion models to estimate the position of the robot or camera at each point in time, and to update the map of the environment based on these estimates. A typical visual SLAM workflow includes the following steps [1], [16]–[18]:

1) **Sensor data acquisition**: Acquisition and preprocessing of camera images (along with IMU sensors, etc.)
2) **Visual odometry (VO)**: Estimation of the camera movement between adjacent frames to estimate the displacement of the observer (ego-motion) and generate a rough local map, also known as the frontend. This tends to be noisy and accumulates error.
3) **Backend filtering and optimization**: Generation of a fully optimized trajectory and map from camera pose and time stamp data from the VO and loop closing modules. This is also known as the backend.
4) **Loop closing**: Determines whether the robot has returned to its previous position in order to reduce accumulated drift. If a loop is detected, it will provide information to the backend for further optimization.
5) **Reconstruction**: Constructs the map based on camera trajectory.

The most direct applications of SLAM are in the fields of autonomous systems and augmented reality. By constructing a map and localizing, SLAM enables vehicles to navigate autonomously in unknown environments and to detect objects, which is essential for identifying and avoiding pedestrians and other vehicles. SLAM also enables robots to navigate through warehouses and hospitals to perform tasks such as inventory management or patient care, or to create accurate maps of complex infrastructure to efficiently detect needed maintenance. In the field of augmented reality, the precision of localization and mapping is essential for creating an immersive experience in realistic virtual environments, which branches out into gaming, training simulations, and remote collaboration.

To reliably test the accuracy of these models, researchers use public, pre-available datasets such as TUM VI [3], KITTI [2], EuRoC [13], and SenseTime [14], which are known as the "gold standard" due to their accuracy of ground truth data of the agent's pose trajectory. These datasets provide real-world sensor data collected from cameras and more advanced sensors such as laser range finders (LIDAR) mounted on a mobile agent.

The generation of the data using simulators has become an increasingly popular approach in computer vision research due to several reasons. First, simulators provide researchers with greater flexibility to create complex datasets with specific features and characteristics that may not be available in existing datasets. This allows researchers to test their algorithms under a wide range of scenarios, which can improve their performance in real-world applications [4]. Second, simulators provide precise ground truth data, which is the exact location, orientation, and other relevant information of the objects in the scene, without the need to collect it separately through more advanced sensors. This is important for training and evaluating computer vision algorithms since ground truth data provides a reference for accuracy and helps to ensure that the algorithm is performing as intended. Third, generating data using simulators like AirSim can be a cost-effective

alternative to collecting real-world data. Collecting data in the real world can be time-consuming and expensive, and may require specialized equipment and personnel. Using a simulator can be a more affordable and accessible option for researchers.

The key contributions of this study are as follows. First, I generate a trajectory resembling the TUM VI Room 1 ground truth dataset within the ArchViz indoor environment using the AirSim simulator and Unreal game engine, and further modify the environment and trajectory to ensure a more stable sequence of input frames for the SLAM architecture. Second, I perform a comprehensive analysis of the efficiency of visual ORB-SLAM3 by adjusting the input image resolution and feature threshold, leading to essential findings on the optimal settings for improved tracking performance within both static and dynamic environments. Third, I investigate the importance of tuning the camera parameters within AirSim and ORB-SLAM3, highlighting the need for the input image resolution to match the dimensions of the film. Lastly, I conduct qualitative testing using real-life indoor environments recorded with an iPhone Xr camera, providing insights into the challenges faced by ORB-SLAM3 in real-world scenarios due to factors such as glare and motion blur. Collectively, these contributions aim to enhance the understanding of visual SLAM performance and its practical applications in various indoor environments.

II. Related Works

A. Feature Extraction

An effective visual odometry system relies not only on the quantity of feature points but also on their quality, which refers to the properties of feature points that influence the performance of feature-based image registration. These properties include repeatability, distinctiveness, efficiency, and locality [1]. Several feature detection algorithms have been developed over the years, aiming to extract distinctive points from images that can be matched across multiple frames. Some of the widely-used feature detection algorithms include scale-invariant feature transform (SIFT), speeded-up robust features (SURF), features from accelerated segment test (FAST), oriented FAST and rotated BRIEF (ORB), and accelerated-KAZE (AKAZE), which vary in their level of accuracy and computational efficiency.

The applicability of these extraction algorithms depends on their ability to run in real time at 30FPS, or 33ms per iteration. The SIFT and SURF algorithms, known for their scale and rotational invariance, suffer from high computational costs that cannot meet the 33ms threshold. The FAST algorithm sacrifices the robustness of SIFT and SURF for faster speed, which may negatively impact its performance in scenarios with significant changes in scale or orientation. ORB, which compromises between efficiency and accuracy, is a combination of the FAST keypoint detector and the BRIEF descriptor, offering an efficient and binary feature detection algorithm. It is more computationally efficient than SIFT and SURF while providing some degree of scale and rotation invariance.

Recent advancements in feature detection algorithms have focused on incorporating machine learning techniques, particularly deep learning, to improve their performance. Convolutional neural networks (CNNs) have been employed to learn feature representations that are more robust and discriminative than traditional hand-crafted features. While these learned features show promising results, they often require significant computational resources, which can limit their applicability in real-time scenarios. Recently, with the advent of edge computing and efficient parallelization, this paradigm of extraction algorithms have been shown to be effective in feature detection. The SuperPoint algorithm is a learning-based approach for keypoint detection and description, which aims to overcome the limitations of traditional hand-crafted feature detection algorithms such as SIFT and FAST, particularly in challenging conditions like poor lighting or low-texture scenes [2]. SuperPoint is based on a lightweight neural network architecture that is trained on image pairs undergoing known homographic transformations. The algorithm comprises three main steps: MagicPoint detection, homographic adaptation, and descriptor training.

1) MagicPoint is the initial step in the SuperPoint algorithm, which focuses on keypoint detection. It is trained on synthetic data to detect corner points in images.
2) Homographic adaptation is then employed to generate image pairs with known homographic transformations, which are used to train the descriptor network.
3) The final step involves training the descriptor network, which is responsible for providing a descriptor for each keypoint detected by MagicPoint. These descriptors are used for image matching tasks, enabling the algorithm to find correspondences between keypoints in different images.

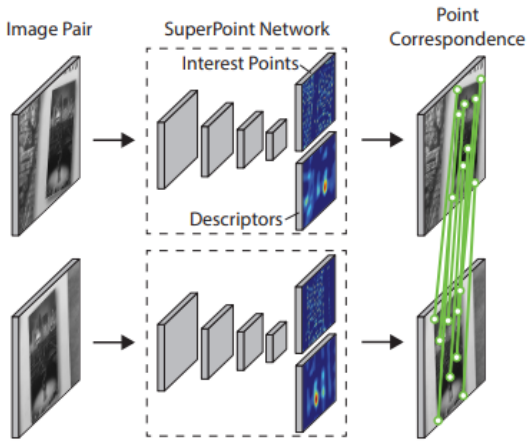


Fig. 1: SuperPoint for geometric correspondences [5].

SuperPoint has demonstrated superior performance compared to traditional hand-crafted algorithms in most benchmarks. It has been found to extract more feature points and perform more stably in low-texture scenes, as well as in poor lighting conditions [2]. The lightweight architecture of the SuperPoint network allows it to maintain relatively low computational requirements while providing improved key-

point detection and description performance, with an empirical runtime of 70FPS on a Titan X GPU.

B. Complex Dataset Generation

Simulators have become a valuable tool in robotics and computer vision research, enabling researchers to generate diverse and complex datasets for various applications. One notable example is TartanAir, a synthetic dataset created by researchers at the Robotics Institute at Carnegie Mellon University, which is specifically designed for visual localization, visual odometry, and SLAM research [8].

TartanAir is an extensive dataset comprising photorealistic RGB-D images, depth maps, and camera poses collected from a wide range of indoor and outdoor environments using a custom-designed flying robot. The creators of TartanAir have meticulously designed the dataset to be challenging and diverse, incorporating a multitude of scenes with dynamic objects, varying lighting conditions, and different types of textures and materials. This level of diversity allows researchers to develop and evaluate algorithms in a more comprehensive and realistic manner, accounting for many real-world challenges that simpler datasets might overlook.

The TartanAir dataset has been instrumental in advancing the state of the art in visual localization, visual odometry, and SLAM research. Its comprehensive nature enables researchers to identify limitations in existing algorithms and develop novel solutions to overcome these challenges. Furthermore, by providing a standardized dataset, TartanAir facilitates fair and consistent comparison between different approaches, which is crucial for understanding the strengths and weaknesses of various techniques.

The use of simulators to generate datasets such as TartanAir not only offers a cost-effective and efficient alternative to collecting real-world data but also provides researchers with the flexibility to easily modify and tailor the dataset to specific research needs. This adaptability allows for rapid iteration and experimentation, ultimately accelerating the development and refinement of algorithms in the field of computer vision.

C. Edge-Assisted SLAM

The integration of SLAM with edge computing has gained significant interest in recent years due to the need for real-time processing of localization and mapping data for various applications. Edge computing refers to the processing of data on devices located close to the source of the data, instead of sending it to a centralized cloud server for processing. This approach has several advantages, such as reduced latency, improved data privacy, and reduced network bandwidth consumption [9]. One of the main needs for using SLAM with edge computing is to enable real-time processing (of 30FPS) of data from sensors such as cameras, LIDAR, or IMUs. These sensors generate a large amount of data, which must be processed in real-time and may cause computational bottlenecks in a lightweight mobile agent. To overcome this limitation, outsourcing computation to an edge server can provide additional computing resources to improve accuracy and performance [10].

One prominent example of integrating edge computing with SLAM is the Edge-SLAM system, which adapts visual SLAM into an edge computing architecture to enable prolonged operation on mobile devices [9]. Edge-SLAM offloads computation-intensive modules of visual SLAM, such as global map optimization and loop closing, to the edge, reducing resource usage on the mobile device without sacrificing accuracy. That is, Edge-SLAM maintains the tracking computation on the mobile device while moving the rest of the computation to the edge [10], allowing the system to overcome the computational bottlenecks often faced by lightweight mobile agents that need real-time processing. Additionally, edge computing allows the agent to offload tasks without the large latencies seen when offloading to the cloud, saving a considerable portion of time during transmission. This approach may be particularly applicable when using machine learning-based feature extraction algorithms such as the previously mentioned SuperPoint algorithm, which may require significant computational power to process data in real-time. By offloading these tasks to the edge, the mobile device can focus on other critical tasks, and the overall system performance can be improved.

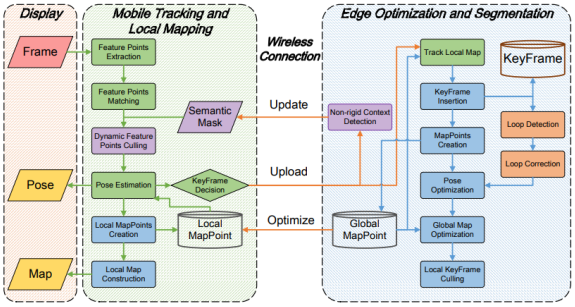


Fig. 2: EdgeSLAM architecture [9].

III. Methodology

A. Environment and Trajectory Initialization and Modification

1) Building Emulators with Unreal and AirSim Plugin: Unreal Engine 4 is a game engine developed by Epic Games that provides developers with a highly customizable editor, a wide range of pre-built templates and advanced rendering capabilities [15].

AirSim is an open-source simulator developed by Microsoft that provides a platform for researchers to simulate and test the performance of autonomous systems, such as drones or cars, which can be controlled through its API supported by C++, Python, and MATLAB. It supports a variety of sensors, such as RGB cameras, depth cameras, LIDAR, and GPS, which allows for efficient data collection that can be streamlined directly into SLAM algorithms. It essentially takes in a sequence of pose vectors and assigns this pose to the agent within the Unreal environment per frame. After the sequence is complete, a directory containing the stream of frames for each pose is generated.

In this study, I will use ORB-SLAM3, an open-source simultaneous localization and mapping system developed by

the Robotics and Perception Group at the University of Zurich [9]. ORB-SLAM3 uses a combination of feature-based and direct monocular, stereo, and RGB-D visual odometry methods and incorporates advanced features such as semantic segmentation, loop closure detection, and multi-map fusion. I will specifically focus on performance testing of visual ORB-SLAM3 with a non-distorted monocular camera at 30FPS.

2) Environment and Trajectory Initialization: In order to investigate the performance of ORB-SLAM3 in a typical indoor setting, I selected a representative environment that closely resembled a standard household interior. The chosen environment, *ArchViz Interior*, is publicly accessible through the Unreal marketplace. This specific environment offers a unique advantage as the distribution of furniture and objects is non-uniform, and consequently, certain frames within the room will exhibit numerous feature points, while others, such as specific wall sections and the ceiling, will have a sparse feature distribution. This non-uniformity presents a substantial challenge for ORB-SLAM3 during tracking, unlike the TUM VI Room 1 environment, in which I noticed contains checkerboards, densely packed shelves, and hanging nets that create a grid-like structure, simplifying the tracking process for ORB-SLAM3.

Fig. 3: Corner view of Unreal ArchViz Interior environment.

In regard to the trajectory, my objective was to identify a trajectory that accurately reflects the movement patterns within an indoor setting, while simultaneously presenting a considerable challenge. The TUM VI Room 1 trajectory proved to be an ideal choice, as it encompasses multiple pans, tilts, and rotations, which together create a complex trajectory for evaluating the performance of ORB-SLAM3.

3) Coordinate Conversion in AirSim: The ground truth data of the TUM VI consists of $16,541$ rows in a CSV file, with each row formatted as

$$\text{row } r = (T^r, p_x^r, p_y^r, p_z^r, q_w^r, q_x^r, q_y^r, q_z^r)$$

where T^r represents the number of nanoseconds past Unix Epoch, (p_x^r, p_y^r, p_z^r) is the translation component in $\mathbb{R}^3$, and $(q_w^r, q_x^r, q_y^r, q_z^r)$ is the unit quaternion representation of the $\mathrm{SO}(3)$ orthogonal group.

When comparing side-by-side the trajectories of the TUM VI Room 1 image frames and those generated by AirSim in the ArchViz environment, I clearly noticed that the coordinates did not align. It is common for different softwares to use different coordinate systems, so a conversion factor between Unreal and

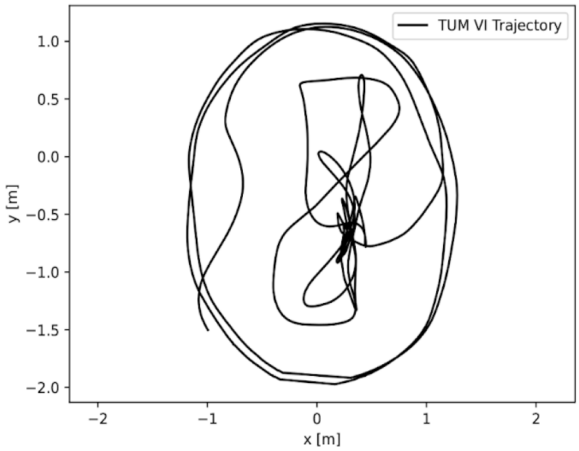


Fig. 4: Time-parameterized plot of the translational component of the TUM VI Room 1 ground truth trajectory.

AirSim was needed. For conciseness of notation, let $(\mathbf{p}, \mathbf{q})$ represent the timestamp plus the 7D pose in Unreal coordinates, and $(\mathbf{p}', \mathbf{q}')$ represent the same in AirSim coordinates. The proper conversion has been empirically tested to be

$$\begin{aligned} p_x &\mapsto -p'_y \\ p_y &\mapsto -p'_x \\ p_z &\mapsto -p'_z \\ q_w &\mapsto q'_w \\ q_x &\mapsto q'_x \\ q_y &\mapsto -q'_y \\ q_z &\mapsto -q'_z \end{aligned}$$

Therefore, after I extracted the TUM-VI ground truth into a Python pandas DataFrame, I followed the coordinate conversion above before inputting the poses into the AirSim API.

Note that the translational conversion factor may differ depending on the orientation of the entire environment. That is, if the entire Unreal environment was rotated 90 degrees across the XY-plane without any change in the world coordinate system, the x and y components in Unreal and AirSim would align.

4) Environment Expansion: After the conversion, I noticed that the total area covered by the trajectory was greater than the original room. During multiple instances, the camera would get too close to the furniture and walls, even sometimes going through them, which is clearly unrealistic and would be problematic for tracking. My goal was for there to be approximately an "arms-length" of space between the agent's camera and the walls throughout the entire simulation, since the ground truth video in the TUM VI Room 1 dataset had kept this approximate distance throughout.

Therefore, I manually expanded the room by selecting all objects on one side of the room, translating them out to the point where I wanted them to be in the expanded room, and then manually adding more sections of the walls, floors, and ceiling where needed. It was not as straightforward as simply highlighting everything and stretching it across the axis using the stretch tool, as there were already too many dependencies

between the objects that caused errors. Additionally, stretching the furniture itself would cause noticeable deformations.

The expansion caused a significant decrease in lighting in two corners of the room, so further modifications were made:

1) The windows on the walls on the north end of the blueprint were stretched out to cover the majority of the stretched wall.
2) An extra window along with a directional light source was added on the walls of the south end of the blueprint to allow for more consistent lighting.

Fig. 5: ArchViz environment before expansion, which had problems in fully containing the coordinate-aligned TUM VI Room 1 trajectory.

Fig. 6: After the expansion of the ArchViz environment, the full TUM VI Room 1 trajectory was observed to have an appropriate distance from the walls of the room for all frames.

B. ORB-SLAM3 and AirSim Parameter Adjustment

1) Timestamp Conversion for ORB-SLAM3 Preprocessing: The frames generated by AirSim are labeled with a timestamp (in nanoseconds after Unix Epoch) representing the time at which the frame was captured in the API. The variance of the time gaps between adjacent frames may be high depending on the hardware in which the simulator is run, the latency of the API calls, the presence of additional functions within the API scripts, and the length of the sleep timer between each API call.

In order to process this into ORB-SLAM3 at 30FPS, I needed to relabel these timestamps such that the time gaps between adjacent poses are approximately 33ms. ORB-SLAM3 also processes timestamp labels in nanoseconds after Unix Epoch, so I relabeled them by choosing an arbitrary starting point $T_0 = 1.5 \cdot 10^{18}$ and setting the timestamp for frame k to be

$$T_k = T_0 + 33,000,000k$$

where $33,000,000$ns represents 33ms.

2) Camera Parameter Adjustment: It must be ensured that the parameters of the AirSim camera align with the camera parameters of ORB-SLAM3. I found that ORB-SLAM3 is extremely sensitive to these parameters and should be double checked for each run. The relevant camera parameters of ORB-SLAM3 can be found and set in a `yaml` file, each of which I explain below.

1) The type of the camera is always set to `"PinHole"`, since I am working with a monocular camera.
2) (W, H) is the width and height of the image, in pixels.
3) (c_x, c_y) is the central point of the camera.
4) (k_1, k_2, k_3) and (p_1, p_2) describes the radial and tangential distortion parameters, respectively.
5) (f_x, f_y) are the focal lengths for each axis.
6) The FPS will always be set to 30 within this study.

I could find or compute all these parameters in AirSim. The basic settings of the AirSim camera can be specified in the `settings.json` file, which should be located at the same directory of the Unreal project executable. It should contain one camera, with the `"ImageType"` set to `0` (to indicate that this is RGB) and the `CameraName` set to `"0"` by default. I have specified the resolution $W \times H$ in pixels of the AirSim camera here.

For reasons elaborated in the *Necessity of Resolution Consistency* subsection, I have noticed that the ORB-SLAM3 parameters require that the resolution ratio $W : H$ matches the ratio $W_{sens} : H_{sens}$ of the sensor width and height of the AirSim camera digital film, which can be accessed directly in the API. While the simulator is running in "AirSimGameMode" in Unreal, I can connect to the API with the following commands.

```
client = airsim.VehicleClient()
client.confirmConnection()
```

One the connection has been established, I have called the `client.simGetFilmbackSettings("0")` method to return the sensor width and height, which is by default $23.76 : 13.365$. I went back to `settings.json` and modified the resolution to match the sensor ratio (e.g. 512×288 if the ratio is $23.76 : 13.365$). Once the proper resolution $W \times H$ has been set, the central points can be set as simply half the resolution as a float value: $c_x = W/2$ and $c_y = H/2$. The distortion parameters can be found by calling `client.getDistortionParams("0")`, which in this study was set to all 0's. The focal lengths f_x, f_y are not as straightforward to retrieve since AirSim does not have built in functions that directly provide them, so I have calculated them manually. To clarify, the axes focal lengths refer to the f_x, f_y shown below when the 3D diagram in figure 7 is projected in the X and Y axes.

By calling the `client.simGetFocalLength("0")` method, I have found the focal length to be 11.9cm, and by

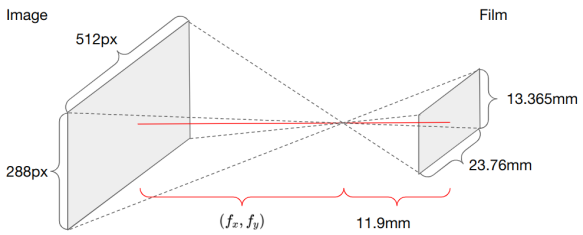


Fig. 7: The diagram representing the projection of the camera image onto the film in the pinhole camera allows us to use geometry to calculate the focal lengths f_x, f_y needed to correctly parameterize ORB-SLAM3.

simple geometry, the axes focal lengths are computed as

$$f_x = \frac{Wf}{W_{\text{sens}}} = \frac{512 \cdot 11.9}{23.76} = 256.431\text{px}$$
$$f_y = \frac{Hf}{H_{\text{sens}}} = \frac{288 \cdot 11.9}{13.365} = 256.431\text{px}$$

While not directly relevant for focal length calculation, I have found that some other methods which may be of use is `simGetCameraInfo`, which returns the pose and the projection matrix of the camera, and `simGetLensSettings`, which returns additional parameters of the AirSim camera.

C. Improving SLAM Stability through Environment Enhancement and Trajectory Smoothing

1) Environment Modification with Feature Addition: Even with the proper unobstructed trajectory and the proper camera parameters, ORB-SLAM3 consistently lost tracking during multiple timestamps, leading to a trajectory error of 87.98cm. I believed that the most likely possibility was due to the lack of features that resulted from the previous environment expansion. When I expanded the room, I did not add any new furniture to it, which subsequently led to large sections of the wall and floor with significantly less features. It seemed as when ORB-SLAM3 processed frames that turned towards these relatively featureless sections, it lost tracking.

Therefore, I added more features to the environment to reduce the possibility of these sudden feature "dead zones." As seen below in the two diagrams, several pieces of furniture were copied and added, including a new dining table set, the rug underneath, multiple ceiling fans, an extra couch, multiple paintings, and additional slight modifications. However, running ORB-SLAM3 within this modified environment led to a marginal improvement from 87.98cm to 80.75cm, which was not a significant enough for me to consider the lack of features to be the main problem, especially since there were still several instances of lost tracking.

I have read that the performance of ORB-SLAM3 drops within environments that have transparent objects such as windows. I have tested this hypothesis by taking the same trajectory but stopping localization right before the frame turns to look at the windows. Creating this new subtrajectory had required significant cutoff, reducing the number of timesteps from 16,541 to less than 2000, but this led to a great improvement in the error, from 80.75cm to 13.17cm.

Fig. 8: Before manual feature addition, the expanded ArchViz environment had an uneven distribution of features throughout the room, with several empty zones where ORB-SLAM3 cannot extract features from.

Fig. 9: After manual feature addition within the environment, ORB-SLAM3 performed marginally better, improving the trajectory error from 87.98cm to 80.75cm.

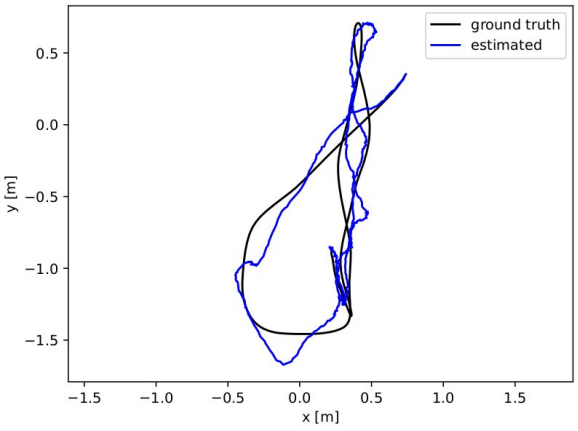


Fig. 10: Upon stopping localization right before the frame turns to look at the first window, the estimated subtrajectory was much closer to the ground truth subtrajectory, reducing the error from 80.75cm to 13.17cm.

However, two pieces of evidence from further testing had

shown me that the presence of windows was not the main problem behind lost tracking. First, when I slightly extended the subtrajectory to include the windows for a few hundred more frames, the error of ORB-SLAM3 still remained at approximately 13cm and did not spike as I expected. Second, when I initialized the trajectory at a pose that initially included the windows in its frame, this also had no considerable effect on the error.

2) Trajectory Jumps and Data Smoothing: From further experimentation with different initial poses and subtrajectories, I have concluded that it was not at a certain frame that ORB-SLAM3 lost tracking, but it was at a certain *timestamp*. This indicated that the problem might have been the trajectory itself. Upon inspection of the TUM VI Room 1 ground truth data, I noticed that there were several large "jumps" between adjacent poses, where the pose in row R was significantly different from the pose in row $R+1$. To quantify the magnitude of these differences, I have constructed a custom pseudo-metric. Computing the distances between the translational components is quite easy, since I can simply take the Euclidean distance between them. The quaternion component is slightly less straightforward.

Let $\mathbb{P} = \mathbb{R}^3 \times \mathrm{SO}(3)$ be the space of all poses, constructed as the Cartesian product of translations $\mathbb{R}^3$ and rotations $\mathrm{SO}(3)$. Let $(\mathbb{H}_1, \cdot)$ denote the quotient group of unit quaternions representing rotations. It is well known that the natural mapping

$$\rho : \mathbb{H}_1 \longrightarrow \mathrm{SO}(3)$$

is 2-to-1. That is, given $\mathbf{q} \in \mathbb{H}_1$ both $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation. Therefore, the distance between $\mathbf{q}$ and itself should be 0, but also $\mathbf{q}$ and $-\mathbf{q}$ should also be 0. In fact, Ravani and Roth recognizes this ambiguity in quaternion representation and constructs the metric for quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ as $\min\{||\mathbf{q}_1 - \mathbf{q}_2||, ||\mathbf{q}_1 - \mathbf{q}_2||\}$, where $||\cdot||$ is the Euclidean norm [11]. Therefore, in this study, I will define the *pose metric* of two poses $\mathbf{P}_1 = (\mathbf{p}_1, \mathbf{q}_1), \mathbf{P}_2 = (\mathbf{p}_2, \mathbf{q}_2)$ to be

$$d(\mathbf{P}_1, \mathbf{P}_2) = ||\mathbf{p}_1 - \mathbf{p}_2||_2 + \min\{||\mathbf{q}_1 - \mathbf{q}_2||_2, ||\mathbf{q}_1 - \mathbf{q}_2||_2\}$$

From graphing the pose metric of each consecutive pair throughout the ground truth trajectory in figure 11, it was evident that there were huge distances between several pairs. While the vast majority of the pose metrics remained small (< 0.02), I have observed in several instances that some translational and rotational components vastly exceeded this difference. For example, in rows 11588 and 11589, the difference of just the q_z components were as large as 0.79. In fact, the majority of the pose metric values came from the difference between quaternion components, rather than the translational ones.

Closer inspection revealed that the the time gap between these large pose jumps were also very large, and so the ground truth data contained several "time gaps" where the pose was not collected. Some data smoothing was required, but before I could begin, I also noticed that the ground truth inconveniently had several instances of where the rotational pose $\mathbf{q}_r$ at row r suddenly jumps to $\mathbf{q}_{r+1} \approx -\mathbf{q}_r$ in the next row. This flip in orientation is not a problem when simulating the trajectory

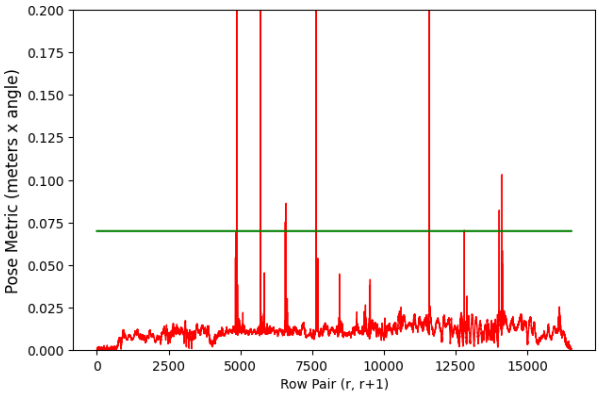


Fig. 11: The graph of the pose metric between adjacent poses in TUM VI Room 1 ground truth trajectory shows several spikes that indicate that there are huge jumps between adjacent frames in our estimated trajectory.

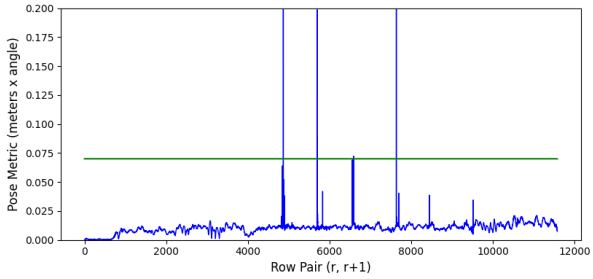
in AirSim since given a rotation quaternion $\mathbf{q}$, both $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation in $\mathrm{SO}(3)$. But this is a problem for data smoothing, and so I created a script that parsed through the ground truth file and converted all the quaternions to the correct orientation. Once all quaternions have been oriented, the data smoothing was done in 3 steps:

1) *smooth.py*: I ran a simple moving average of 10 rows (SMA-10) on only the quaternion components of the oriented data, to ensure that the differences in each component of $\mathbf{q}_r$ and $\mathbf{q}_{r+1}$ was < 0.07, which was a threshold that I've empirically determined to be an acceptable pose metric that did not cause ORB-SLAM3 to lose tracking. This smoothed out most of the data.
2) The greatest pose metric after smoothing had still remained at a value of ~ 0.5, and increasing the timesteps of the moving average further may have distorted the rotational data too much, so I cut off the trajectory right before this massive jump.
3) I have manually smoothed out any other remaining spikes by inserting rows in between them and linearly interpolating until all pose metrics met the < 0.07 threshold.

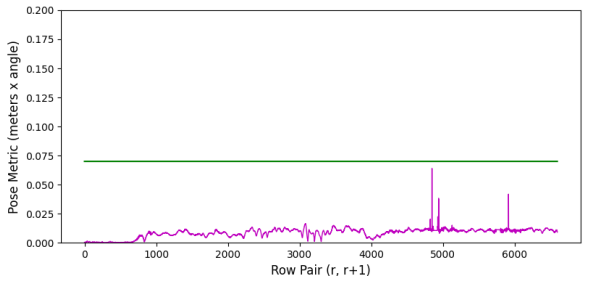
D. Resolution and Feature Threshold Adjustment

The two main factors of performance of ORB-SLAM3 is the resolution of the input images, which dictate how much data is being fed into the algorithm, and the number of features that ORB-SLAM3 is allowed to detect. There is another parameter in the settings `yaml` file, `nFeatures`, that sets the maximum number of feature points M that ORB-SLAM3 would detect in a given frame. If L feature points are detected in a frame, then $\min\{M, L\}$ features would be extracted for visual odometry. Therefore, I have performance tested ORB-SLAM3 with the following datasets and parameters:

1) I work with two different resolutions: 256×144 and 512×288 pixels.



(a) Pose metric between adjacent poses in TUM VI Room 1 ground truth trajectory after SMA-10 smoothing and cutoff.



(b) Pose metric between adjacent poses in TUM VI Room 1 ground truth trajectory after linear interpolation and cutoff.

Fig. 12: After performing both smoothings and cutoffs, there were no more instances of ORB-SLAM3 losing tracking.

2) For each resolution, I set the feature threshold to $750, 1500, 3000, 6000$, and 12000 points.
3) Due to the stochastic nature of ORB-SLAM3, I run ORB-SLAM3 three times for each resolution and for each feature threshold.

1) Error Evaluation: I have computed the error using the *evaluate_ate_scale.py* module, which computes the absolute trajectory error of the estimated trajectory. It first reads in the the ground truth trajectory $\{\mathbf{P}_k\}_{k=1}^N = \{(\mathbf{p}_k, \mathbf{q}_k)\}_{k=1}^N$ and estimated trajectories $\{\mathbf{P}'_k\}_{k=1}^N = \{(\mathbf{p}'_k, \mathbf{q}'_k)\}_{k=1}^N$ as two lists of 7D poses, each pose consisting of a timestamp, 3D translation, and 4D quaternion representing the rotation. Then, it aligns the estimated trajectory to the ground truth trajectory using Horn's method, which computes a rotation matrix and a scale factor that best aligns the two trajectories, which I will denote $\{\mathbf{P}^*_k\}$ for the ground truth and $\{\mathbf{P}'^*_k\}$ for the estimated. After alignment, it computes the translational error between the two trajectories by finding the Euclidean distance between the corresponding translation components of each pose. The script then reports the Root Mean Squared Error (RMSE) of the translational error over all poses as the absolute trajectory error.

$$\text{Error} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} ||\mathbf{p}^*_k - \mathbf{p}'^*_k||_2^2}$$

In figure 13, we can observe Horn's method aligning the ground truth trajectory within the TUM VI Room 1 environment and ORB-SLAM3's estimated trajectory within the ArchViz Interior environment by seeing that the translational components roughly align if we perform the coordinate conversion $p_x \mapsto p_z, p_y \mapsto p_x, p_z \mapsto p_y$, which is represented by the rotation matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Similarly in figure 14, for the rotational components, we expect Horn's method to align the basis elements of the quaternions with the coordinate conversion $q_w \mapsto q_z, q_x \mapsto q_y, q_y \mapsto q_w, q_z \mapsto q_x$, represented by the rotation matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

2) Robustness in Dynamic Environments: I also tested the robustness of ORB-SLAM3 within a dynamic environment containing moving objects. After identifying a range optimal feature thresholds within the static environment, I had analyzed the change in tracking performance of ORB-SLAM3 for both the 256×144 and 512×288 resolutions. The moving objects consisted of three ceiling fans, rotating at fast speed, and a decorative hanger, rotating relatively slowly. While this may not seem like a significant difference, the directional lights coming in from the windows had cast large moving shadows as well. A quick snapshot of the fans (on the left) and the hanger (next to the window on the right) is provided in figure 15.

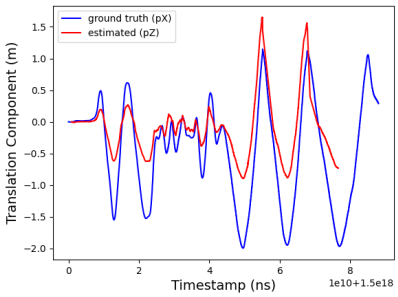
E. Performance Testing in Real-Life Environments

Furthermore, I have tested ORB-SLAM3 within two real-world environments:

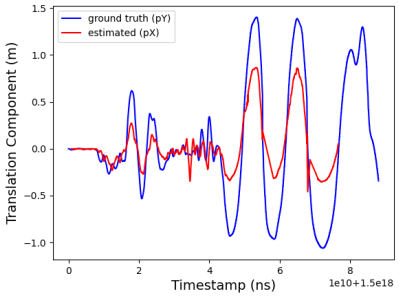
1) the Wilkinson 4th floor graduate lounge, which consisted of glass walls, with multiple tables, chairs, a fridge, a sink, under bright lighting.
2) the Kilgo Quad N common room, consisting of various couches, tables, chairs, a TV, a foosball table, and framed walls.

Both environments represent typical indoor locations with consistent lighting. The main difference between the two is that the graduate lounge consists of glass walls, while the common room does not have many windows nor transparent objects. I believed that having at least a qualitative analysis of how the performance of ORB-SLAM3 is affected by the presence of transparent windows and walls would prove useful in developing assumptions in future works. While the ground truth data was not available, I have qualitatively observed the general accuracy of ORB-SLAM3 by manually comparing the trajectories taken in the video with the ego-motion computed by the localization module.

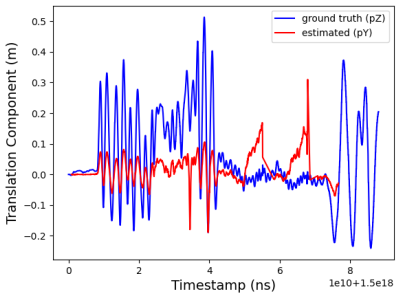
The default resolution for an iPhone Xr camera is 1920×1080px, but I have compressed and cropped the image frames to be 960×720px. The camera type was also sent to pinhole, the central point was set as $(480, 360)$, and the distortion parameters were all set to 0. To compute the axes focal lengths f_x, f_y, I've collected measurements on the focal length and the film dimensions online.



(a) Ground truth pX aligned with estimated pZ trajectory

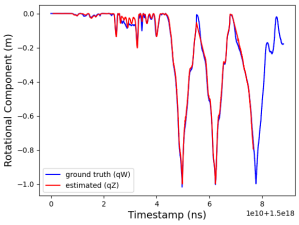


(b) Ground truth pY aligned with estimated pX trajectory

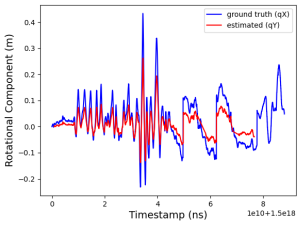


(c) Ground truth pZ aligned with estimated pY trajectory

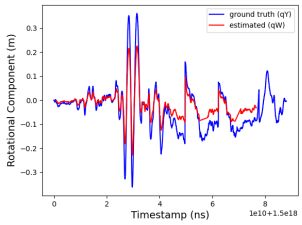
Fig. 13: Rough estimate of how Horn's method would align the translational components of the ground truth and estimated trajectories with a rotational factor.



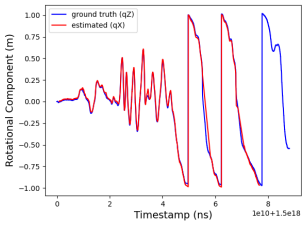
(a) Ground truth qW aligned with estimated qZ trajectory



(b) Ground truth qX aligned with estimated qY trajectory



(c) Ground truth qY aligned with estimated qW trajectory



(d) Ground truth qZ aligned with estimated qX trajectory

Fig. 14: Rough estimate of how Horn's method would align the rotational components of the ground truth and estimated trajectories with a rotational factor.

Fig. 15: The moving objects in the dynamic environment consisted of 3 fast-rotating fans and slow-rotating decorative hanger, along with their projected shadows.

Fig. 16: Corner view of Wilkinson 4th floor graduate lounge.

1) The sensor number is known to be $1/2.55''$, which corresponds to a film dimension of 5.33×4mm
2) The focal length is 2.45mm (26mm equivalent for digital camera lens)

Therefore, the axes focal lengths were calculated to be

$$f_x = \frac{960 \times 4.25}{5.33} = 765.478\text{px}, \; f_y = \frac{720 \times 4.25}{4} = 765.0\text{px}$$

Once the video was taken, each frame was saved as an image using OpenCV and labeled with the appropriate timestamp as specified previously before inputting the data into ORB-SLAM3.

IV. Results

The AirSim dataset of images and ground truth were collected in Unreal Engine 4.27, run on a Dell XPS 9570 with 16GB RAM and 16 CPU cores. ORB-SLAM3 was run on the Ubuntu 20.04 operating system, with 8GB RAM and 8 CPU cores. Within both the static and dynamic environments, the performance of ORB-SLAM3 with 512×288 resolution images had significantly outperformed that of 256×144 resolution images. Furthermore, ORB-SLAM3 with the higher resolution inputs had been much more robust in dynamic

Fig. 17: Corner view of Kilgo N common room.

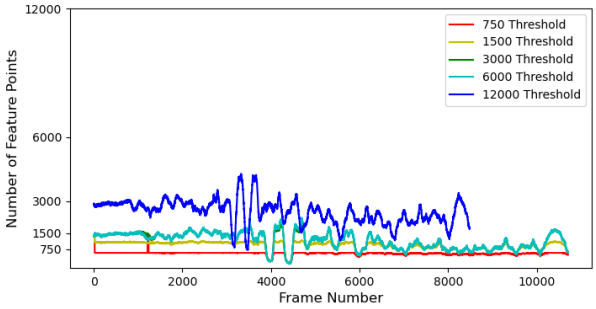


Fig. 18: Number of features detected per frame in ORB-SLAM3 with images of resolution 256×144 in static environment. For the thresholds of 750 and 1500, ORB-SLAM3 consistently detects the maximum number of feature points throughout the entire trajectory.

environments, with no noticeable loss in accuracy, while the accuracy had significantly dropped with lower resolution inputs.

A. *Impacts of Resolution and Threshold in Static Environment*

In the static environment, I have observed that the absolute trajectory error was significantly smaller when utilizing higher resolution inputs compared to lower resolution ones under all threshold levels. Although this may seem obvious, as higher image resolution intuitively leads to improved tracking, it is important to consider the role of feature points in visual odometry for localization. In this case, both the 256×144 and 512×288 resolution images had their feature threshold limited at the same level, which indicates that ORB-SLAM3 is extracting the same number of features per frame, particularly for lower thresholds.

TABLE I: Absolute trajectory error of estimated trajectory in stationary environment (cm)

		Feature Threshold				
		750	1500	3000	6000	12000
Res.	256×144	65.825	64.089	42.787	31.556	77.704
		96.676	86.998	78.844	68.121	40.119
		97.385	96.858	66.803	64.351	100.312
	512×288	12.735	10.025	9.431	8.246	5.815
		6.000	9.963	9.315	6.657	4.942
		9.963	9.384	11.959	6.652	7.763

I have slightly modified ORB-SLAM3 to have it output the number of features detected in each frame, and it was evident that for the runs with the lower thresholds (750 and 1500), the number of features extracted from each frame remained consistently at the maximum threshold for both the higher and lower resolution inputs. This suggests that the "quality" of the features is a critical factor, as demonstrated by the 750 or 1500 threshold runs, where the quality of the 1500 features detected in the 512×288 images was significantly better than that of the 1500 features detected in the 256×144 images.

A visual graph in figure 20 is shown below to better compare the performance between the two resolutions, where the mean errors of each three runs are plotted, along with their sample standard deviation in the error bars. The graph shows that ORB-SLAM3 not only has significantly less error with higher resolution inputs, but the trajectory error remains very consistent throughout multiple runs. On the lower resolution inputs, however, ORB-SLAM3 was quite unstable in localization,

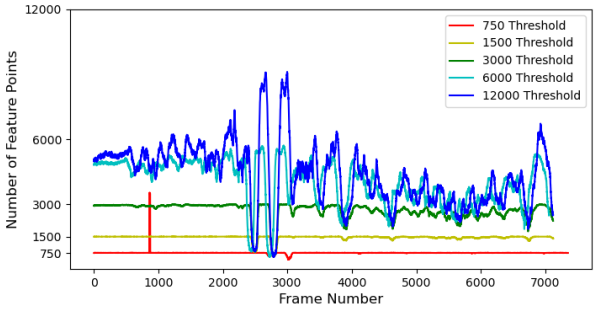


Fig. 19: Number of features detected per frame in ORB-SLAM3 with images of resolution 512×288 in static environment. For the thresholds of 750 and 1500, ORB-SLAM3 consistently detects the maximum number of feature points throughout the entire trajectory. For higher thresholds, ORB-SLAM3 detects significantly more feature points on the 512×288 images than on the 256×144 images.

generating significantly different trajectories per run, leading to a high standard deviation in the sample data.

It seems that ORB-SLAM3 with extremely high feature points was able to construct a perceivable point-cloud map of the room. In figure 21, certain objects, like the walls, the lights, the table, and the rug, are slightly discernible, though they are easier to identify directly in the interactive software outputting the point-cloud map.

B. *Impacts of Resolution and Threshold in Dynamic Environment*

The optimal performance of both resolutions seemed to lie within the $3000, 6000$, and 12000 feature thresholds, and so under this range, I ran ORB-SLAM3 again within the dynamic environment. The results show that ORB-SLAM3 is significantly more robust to moving entities with the higher resolution images.

Since I was working with higher feature thresholds in the dynamic environment, there was a significant difference in the number of feature points detected between the 256×144 and

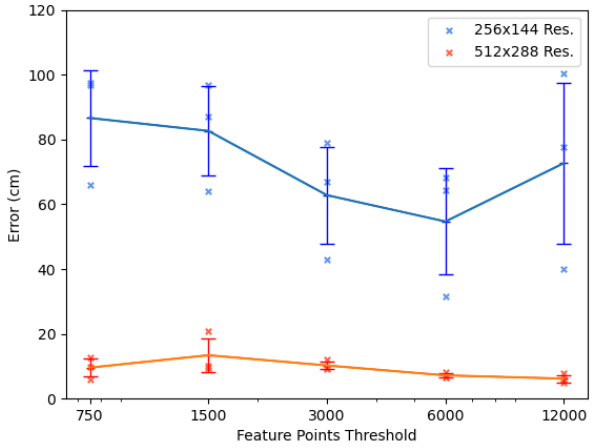


Fig. 20: Effects of resolution on ORB-SLAM3 performance under varying feature thresholds in static environment. These results indicate that the feature points detected by ORB-SLAM3 are of higher quality (for lower thresholds) and more numerous (for higher thresholds), both of which contribute to higher performance.

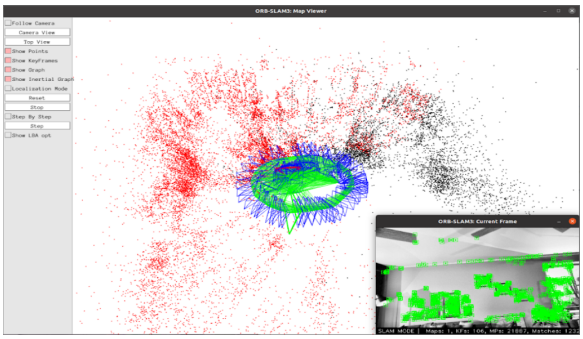


Fig. 21: Distinguishable objects are visible in the point-cloud map generated by ORB-SLAM3 with 512×288 resolution inputs and threshold of 12000 feature points.

TABLE II: Absolute trajectory error of estimated trajectory in dynamic environment (cm)

		Feature Threshold		
		3000	6000	12000
Res.	256×144	104.524	63.135	99.287
		87.443	60.367	104.449
		58.399	101.922	99.836
	512×288	7.987	7.849	4.812
		8.731	7.212	6.778
		13.314	9.014	5.082

512×288 resolution images. Even though the figures below indicate that even at the 3000 threshold level, ORB-SLAM3 is detecting much more features in the 512×288 images, the difference in its performance is most likely due to the difference in quality of the feature points.

By superimposing the trajectory errors within the dynamic environment onto the those computed within the static, figure 24 shows that the performance of ORB-SLAM3 is barely affected when inputting 512×288 resolution images, while the absolute trajectory error increases by about $20 \sim 30$cm with 256×144 resolution inputs.

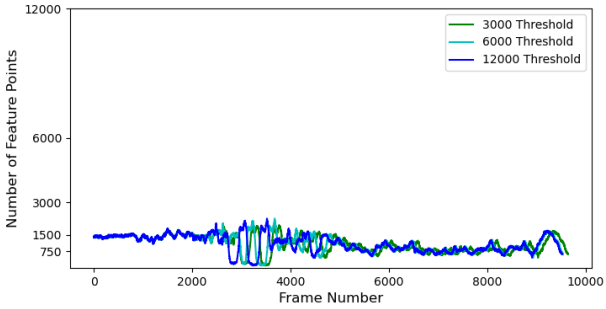


Fig. 22: Number of features detected per frame in ORB-SLAM3 with images of resolution 256×144 in dynamic environment.

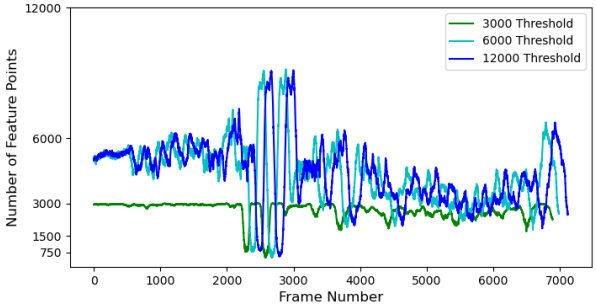


Fig. 23: Number of features detected per frame in ORB-SLAM3 with images of resolution 512×288 in dynamic environment. For these higher thresholds, ORB-SLAM3 detects significantly more feature points on the 512×288 images than on the 288×144 images.

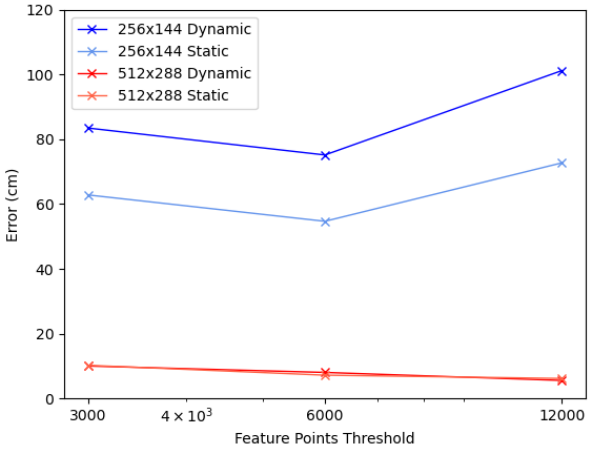


Fig. 24: Effects of resolution on ORB-SLAM3 performance under varying feature thresholds in both static and dynamic environments. This figure indicates that ORB-SLAM3 is much more robust in dynamic environments with 512×288 resolution inputs than with 256×144 resolution inputs.

I have organized the error difference below. Even though the presence of negative values indicate that there is a marginal

improvement in tracking in the dynamic environment, this is most likely due to the stochasticity of ORB-SLAM3, and with subsequent runs, the average difference in the absolute trajectory error will converge to a positive value.

TABLE III: Increase in absolute trajectory error upon shift from static to dynamic environment (cm)

		Feature Threshold		
		3000	6000	12000
Res.	256 × 144	20.64	20.46	28.48
	512 × 288	-0.22	0.84	-0.62

C. Necessity of Resolution Consistency

I want to emphasize a certain restriction on the resolution of the input images when working with AirSim and ORB-SLAM3. It is extremely important that the resolution of the image and the resolution of the film have the same width-to-height ratio.

During the experiment, I have initially worked with 256×256 and 512×512 resolution images. The respective focal lengths was computed to be $(f_x, f_y) = (128.215, 227.939)$ for 256×256 and $(f_x, f_y) = (256.431, 455.877)$ for 512×512. However, running ORB-SLAM3 with these parameters had resulted in the 256×256 resolution inputs consistently outperforming the 512×512 resolution inputs.

TABLE IV: Absolute trajectory error of estimated trajectory in stationary environment with 1:1 ratio resolutions (cm)

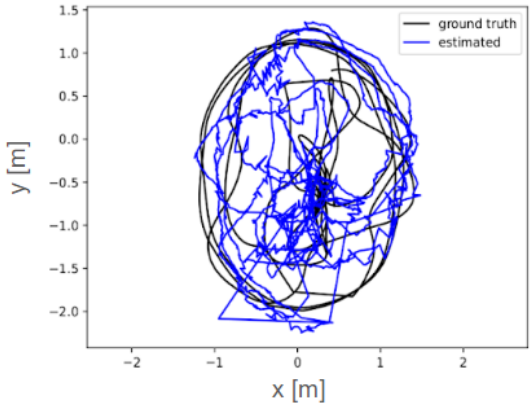
		Feature Threshold				
		750	1500	3000	6000	12000
Res.	256 × 256	22.702	36.570	26.764	20.625	23.856
		25.030	31.278	33.358	21.965	23.846
		20.705	30.555	27.503	25.265	22.253
	512 × 512	83.934	56.951	35.948	31.921	26.400
		50.444	39.745	33.029	26.522	30.590
		52.048	79.724	30.383	38.893	77.566

This was a very counter-intuitive result, as I would have expected that having more information of an image through higher resolutions would result in better tracking overall. I had hypothesized that perhaps the 512×512 too fine of a resolution, containing extremely large amounts of data that led ORB-SLAM3 to overfit the features in some way, but this contradicted the results of multiple other studies in the literature, where researchers would work with resolutions that went as high as 1920×1080 in very complex environments and still generated very accurate localization [12]. Even the TUM VI dataset consisted of images with a relatively high resolution of 1024×1024 pixels, and ORB-SLAM3 had still produced a consistently small $\sim$ 7cm error.

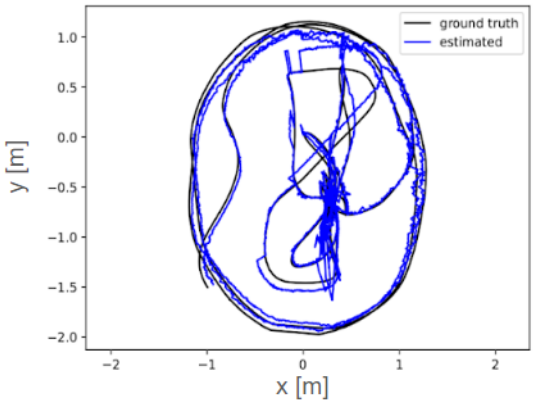
I noticed that the parameters for ORB-SLAM3 tuned for the TUM VI Room 1 dataset had set the axes focal lengths to be approximately equal: $f_x \approx f_y$. I attempted to mimic this setting by modifying the resolution of my input images. If I keep the dimensions of the resolution $W : H$ the same as the dimensions of the film $W_{\text{sens}} : H_{\text{sens}}$, then the resulting calculations would lead to $f_x = f_y$. Since the sensor had a $16 : 9$ ratio, I had modified my input resolutions to 256×144 and 512×288, which led to the axes focal lengths to be $(128.215, 128.215)$ and $(256.431, 256.431)$, respectively. This led to an extremely large improvement in error for the higher resolution images.

TABLE V: ORB-SLAM3 performance comparison between 1:1 ratio resolutions and 16:9 ratio resolutions in static environment, averaged over 3 runs (cm)

		Feature Threshold				
		750	1500	3000	6000	12000
Res.	256 × 256	22.813	32.801	29.208	22.618	23.318
	256 × 144	86.629	82.648	62.811	54,676	72.712
	512 × 512	62.142	58.807	33.120	32.225	44.852
	512 × 288	9.566	9.791	10.235	7.185	6.173



(a) Trajectories of 512×512 image frames with 1500 feature threshold.



(b) Trajectories of 512×288 image frames with 1500 feature threshold.

Fig. 25: Upon modifying the resolutions to match the film dimensions, the ORB-SLAM3's tracking performance had greatly improved, from an error of 58.8cm to 9.8cm.

While I haven't found out exactly the reason why the image resolution must have the same dimensions as the film, this may simply be an intrinsic property of ORB-SLAM3. Regardless, this observation in its own right is very useful for future studies.

D. SLAM with iPhone Xr Camera in Physical Indoor Environment

Qualitatively, the localization of ORB-SLAM3 in a real-world environment appears to correspond reasonably well

with the actual trajectory, as observed by comparing the video and localization module side-by-side. Moreover, the system generates a relatively dense point-cloud map of the environment, allowing for the identification of distinct features such as wall nooks.

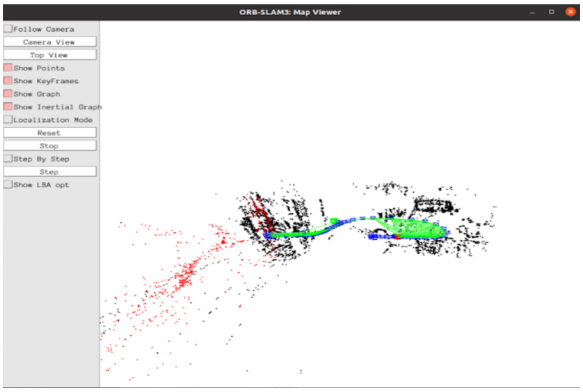


Fig. 26: Snapshot of the localization and point cloud map of Kilgo Quad N common room environment in ORB-SLAM3 map viewer. The estimated trajectory roughly aligns with my trajectory in the video, and the density of the point-cloud map allows us to identify distinct features of the environment.

Nonetheless, in both the graduate lounge and the common room, the localization exhibited considerable instability, characterized by multiple instances of lost tracking. I attributed this phenomenon to two primary factors:

1) The substantial exposure to windows and transparent objects in the graduate lounge may have caused ORB-SLAM3 to lose tracking due to the glare and reflections of both the individual operating the system and the objects within the room.
2) The videos captured by the iPhone Xr displayed significant motion blur, even during relatively slow changes in pose. This factor could have potentially impeded ORB-SLAM3's capacity to detect features within each frame, consequently resulting in abrupt declines in performance.

V. Future Works

This research has provided valuable insights and a foundation for further exploration and development in the domain of visual SLAM. Based on our findings, I have identified three key areas for future works:

A. Visual-Inertial SLAM Integration

Expanding the scope of this research to include visual-inertial SLAM is essential, as the majority of real-world SLAM applications incorporate inertial data using gyroscopes. Integrating inertial measurement units (IMUs) with the visual ORB-SLAM3 framework could significantly improve the system's accuracy and robustness under various conditions. This integration would involve fusing IMU data with visual information, allowing for enhanced tracking and mapping in situations where visual data alone may not suffice, such as in low-light environments or during rapid motion. Furthermore, exploring various sensor fusion techniques and optimization methods could lead to even more accurate and reliable visual-inertial SLAM systems.

B. Diverse and Challenging Datasets

Creating diverse and challenging datasets is crucial for advancing the evaluation and development of SLAM algorithms, and many efforts have been made to add to the diversity of public datasets. The *RobotCar* dataset, collected through a consistent route in Oxford, UK, over the span of a year, contains large scale data in changing light and weather conditions for self-driving tasks [19]. Other researchers have taken advantage of emulators to efficiently navigate virtual environments to create diverse datasets such as TartanAir [8] and DISCOMAN [20]. Following this approach, by using emulators to generate complex datasets under different environments, weather conditions, and challenging trajectories, I can provide researchers with a wide range of test scenarios and foster the development of versatile SLAM systems. This endeavor could involve creating indoor and outdoor scenes, simulating varying lighting conditions, and incorporating challenging elements such as occlusions, dynamic objects, and complex motions. Ultimately, these comprehensive datasets would help assess the performance and robustness of SLAM algorithms under a broad array of real-world conditions.

C. Streamlining Camera Calibration Process

Streamlining the camera calibration process is a critical aspect of making SLAM systems more usable and accessible, since it may not be always the case that researchers have access to the relevant camera specifications. Certain calibration modules included in the OpenCV library allows one to extract an estimate of the camera intrinsic and extrinsic matrices, along with distortion parameters [21], but the integration of this process into SLAM and AirSim is still poor.

By using SLAM to initially approximate camera parameters, I believe that is is possible to develop a hierarchical model that adaptively tunes the hyperparameters of SLAM based on the range of parameters in which it performs optimally (i.e., exhibits the least trajectory error). This approach not only eases the calibration process but also allows for continuous adaptation and improvement of SLAM performance as the environment or operating conditions change. By refining this hierarchical model and exploring novel techniques for adaptive hyperparameter tuning, I can ultimately enhance the accuracy and adaptability of SLAM systems, making them more suitable for a wide range of applications. This improvement in the calibration process could significantly enhance the usability and accessibility of SLAM systems for various researchers and developers.

VI. Conclusion

In conclusion, this study utilized the AirSim emulator to generate a challenging trajectory within the ArchViz Interior environment built within Unreal. Various modifications such as

environment expansion and trajectory smoothing, paired with a careful calculation of the camera parameters, were applied to ensure stability of ORB-SLAM3's tracking module.

The results demonstrate that employing higher resolution images (256×144 and 512×288) at 30FPS significantly improves tracking performance in visual SLAM, specifically ORB-SLAM3. It also highlights the importance of environment selection, trajectory stability, and camera parameter tuning for maintaining consistent localization, allowing for a comprehensive evaluation of the visual ORB-SLAM3 performance. My results indicate that the optimal feature threshold is found to be between $3000 \sim 12000$ feature points per frame, and ORB-SLAM3 shows enhanced robustness in dynamic environments with moving objects when using higher resolution inputs. These findings provide valuable insights for future development and optimization of visual SLAM systems.

Furthermore, qualitative testing in real-world indoor environments highlighted challenges faced by ORB-SLAM3, such as glare and motion blur, which offers a direction for future improvements in the SLAM system. Overall, the study's findings contribute to the growing body of knowledge in the field of visual SLAM and can potentially enhance the development and application of SLAM systems in computer vision.

ACKNOWLEDGMENTS

I would like to thank Ying Chen for her mentorship and Professor Maria Gorlatova for overseeing this Spring 2023 research independent study.

REFERENCES

[1] X. Gao and T. Zhang, "Introduction to Visual SLAM From Theory to Practice," 2021.

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision Meets Robotics: The KITTI Dataset," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, vol. 32, no. 11, pp. 1231-1237.

[3] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stuckler, and D. Cremers, "The TUM VI Benchmark for Evaluating Visual-Inertial Odometry," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1696-1701.

[4] S. Shah, D. Dey, C. Lovett, A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, vol. 78, pp. 1-16.

[5] D. Detone, T. Malisiewicz, and A. Roabinovich, "SuperPoint: Self-Supervised Interest Point Detection and Description," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7122-7131.

[6] P. H. Christiansen, M. F. Kragh, Y. Brodskiy, and H. Karstoft, "UnsuperPoint: End-to-end Unsupervised Interest Point Detector and Descriptor," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 118-133.

[7] J. Tang, H. Kim, V. Guizilini, S. Pillai, R. Ambrus, "Neural Outlier Rejection for Self-Supervised Keypoint Learning," in *Computing Research Repository (CoRR)*, 2019, vol. 43, no. 3, pp. 834-847.

[8] W. Wang, D. Zhu, X. Wang, Y. Hu, W. Qiu, C. Wang, Y. Hu, A. Kapoor, S. Scherer, "TartanAir: A Dataset to Push the Limits of Visual SLAM" in *IEEE Robotics and Automation Letters*, 2020, vol. 5, no. 4, pp. 6673-6680.

[9] J. Xu, H. Cao, D. Li, K. Huang, C. Quian, L. Shangguan, and Z. Yang, "Edge Assisted Mobile Semantic Visual SLAM," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1828-1837.

[10] A. J. B. Ali, Z. S. Hashemifar, and K. Dantu, "Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping," in *ACM Transactions on Embedded Computing Systems*, 2020, vol. 22, no. 1, pp. 1-31.

[11] B. Ravani, B. Roth, "Motion Synthesis Using Kinematic Mappings," in *Journal of Mechanisms, Transmissions, and Automation in Design*, 1983, vol 105, no. 3, pp. 460-467.

[12] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, J. D. Tardos, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM," in *IEEE Transactions on Robotics*, 2020, vol. 37, no. 6, pp. 1874-1890.

[13] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets" in *The International Journal of Robotics Research*, 2016, vol. 35, no. 10, pp. 1157-1163.

[14] J. Li, B. Yang, D. Chen, N. Wang, G. Zhang, and H. Bao, "Survey and Evaluation of Monocular Visual-Inertial SLAM Algorithms for Augmented Reality" in *Journal of Virtual Reality & Intelligent Hardware*, 2019, pp. 386-410.

[15] Epic Games, "Unreal Engine," Epic Games, March 2015, Available: *www.unrealengine.com*

[16] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," in *IEEE Transactions on Robotics*, 2015, vol. 31, no. 5, pp. 1147–1163.

[17] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "Avr: Augmented vehicular reality," in *MobiSys '18: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 81-95.

[18] T. Pire, T. Fischer, G. Castro, P. D. Cristóforis, J. Civera, and J. J. Berlles, "S-ptam: Stereo parallel tracking and mapping," in *Robotics and Autonomous Systems*, 2017, vol. 93, pp. 27-42.

[19] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," in *The International Journal of Robotics Research (IJRR)*, 2017, vol. 36, no. 1, pp. 3–15.

[20] P. Kirsanov, A. Gaskarov, F. Konokhov, K. Sofiiuk, A. Vorontsova, I. Slinko, D. Zhukov, S. Bykov, O. Barinova, and A. Konushin, "Discoman: Dataset of indoor scenes for odometry, mapping and navigation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2470-2477.

[21] Y. M. Wang, Y. Li and J. B. Zheng, "A camera calibration technique based on OpenCV," in *The 3rd International Conference on Information Sciences and Interaction Sciences*, 2010, pp. 403-406.