Simultaneous Tracking and Rendering: Real-time Monocular Localization for MAVs

Kyel Ok, W. Nicholas Greene, and Nicholas Roy[1]

***Abstract*— We propose a method of real-time monocular camera-based localization in known environments. With the goal of controlling high-speed micro air vehicles (MAVs), we localize with respect to a mesh map of the environment that can support both pose estimation and trajectory planning. Using only limited hardware that can be carried on a MAV, we achieve accurate pose estimation at rates above 50 Hz, an order of magnitude faster than the current state-of-the-art mesh-based localization algorithms. In our simultaneous tracking and rendering (STAR) approach, we render virtual images of the environment and track camera images with respect to them using a robust semi-direct image alignment technique. Our main contribution is the decoupling of camera tracking from virtual image rendering, which drastically reduces the number of rendered images and enables accurate full camera-rate tracking without needing a high-end GPU. We demonstrate our approach in GPS-denied indoor environments.**

I. Introduction

We are interested in fast monocular vision-based camera pose estimation that can track MAVs in known GPS-denied indoor environments. We would like to achieve localization performance of laser-based [1] or RGBD-based [2] algorithms using a single camera that is both lightweight and power-efficient.

State-of-the-art monocular vision-based localization algorithms that track cameras in real-time are insufficient to support autonomous MAV navigation due to difficulties in managing viewpoint changes and inferring free space. Most single camera localization algorithms [3]–[6] utilize image-space descriptors [7], [8] where the descriptors in a camera frame are matched against a database of descriptors with known 3D locations. While image-space descriptors are discriminative, they are fundamentally based on a 2D projection in the camera image and are not repeatable under large changes in the viewpoint [9]. Such viewpoint dependency is often not a significant issue on ground vehicles restricted by roads and constrained in elevation [10]–[12] or human-taken photos sharing similar viewpoints [5] but the problem worsens for MAVs unconstrained in space. More importantly, a sparse point-cloud map is not informative in inferring free space for planning a collision-free trajectory.

A few monocular vision-based localization approaches utilize colored mesh maps [13], [14] which suffer less from viewpoint changes and also contain spatial occupancy, but are computationally intractable on MAVs. These previous approaches track the camera pose by searching for the virtual camera, guided by a differentiable image-space similarity metric, that renders the most similar virtual image. The search process requires repeatedly re-rendering virtual images for each camera image until the most similar image is found. As a result, the state-of-the-art mesh-based algorithm [13] tracks at 2 Hz using a high-performance GPU difficult to carry on a MAV.

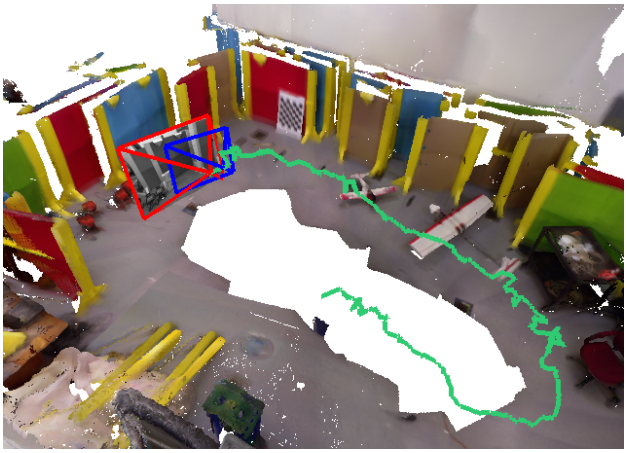


Fig. 1: STAR algorithm tracking the camera pose (red) with respect to a synthetic keyframe (blue) rendered using a colored mesh of an indoor environment.

We propose simultaneous tracking and rendering (STAR) for high frame-rate localization that drastically reduces the number of rendered images. We do not require rendering synthetic images multiple times per frame, but in fact, only render a new synthetic image after sufficient motion. This reduction comes from decoupling camera tracking from synthetic image rendering where instead of rendering the most similar image, we render any image that overlaps with the current view and use it as a *keyframe* to track future camera images with. This notion of keyframe-based localization also allows relocalization, i.e., recovering from failed localization, by searching for a candidate keyframe in proximity. This kind of relocalization has not been previously demonstrated by the mesh-based localization algorithms [13], [14] that rely on a good initial estimate on the pose.

Given a rendered keyframe, we track camera images against it using a robust semi-direct image alignment technique. This real-to-virtual image tracking is similar to the trackers in RGBD SLAM systems [15] that align RGB images with the projection of an incrementally updated 3D representation of the environment. The main difference in STAR is that a RGBD sensor is only required for the offline map building, allowing the online tracking to perform with a

[1]Authors are from the Robust Robotics Group, CSAIL, MIT (kyelok, wngreene, nickroy)@csail.mit.edu

single camera. Additionally, by building a map incrementally from a temporally coherent sequence of images, the SLAM problem minimizes the need to align images with dramatically different brightnesses or the need to align partially incomplete depth maps, problems that only occur at loop closures, if they ever occur. In contrast, localization may require dealing with images taken under illumination conditions that were never encountered during map construction, and can require localizing in an incomplete part of the mesh. In STAR, we overcome such challenges by correcting for a global illumination difference and re-weighting local flaws caused by rendering with an incomplete mesh.

The main contributions of our STAR algorithm are: 1) achieving an order of magnitude faster tracking rate than the current state-of-the-art monocular mesh-based localization, 2) removing the need for a high-end GPU difficult to carry on a MAV, 3) being able to recover from localization failure in a mesh map. We demonstrate STAR in indoor environments, such as the room shown in Figure 1.

In the following sections, we describe STAR in more detail and provide experimental results on tracking fast-moving cameras in indoor environments.

II. Simultaneous Tracking and Rendering

Given an undistorted monocular camera image I_t at each time step t, we are interested in finding the current camera pose $\mathbf{T}_t^W \in SE(3)$ with respect to the world frame of reference W. We leverage a geometrically accurate prior map M of the environment and occasionally render a synthetic camera image I_s and a corresponding depth map I_d at a desired keyframe pose $\mathbf{T}_k^W$ in the world.

Then, we can reduce the camera tracking problem to two parallel processes: the first process compares the current camera image against a synthetic keyframe image and finds the camera pose $\mathbf{T}_t^k$ with respect to the keyframe camera

$$\mathbf{T}_t^k = f(I_s, I_d, I_t) \tag{1}$$

and the second process simultaneously renders synthetic keyframe images given a prior map and a candidate pose

$$I_s, I_d = f(M, \mathbf{T}_k^W). \tag{2}$$

The first image alignment process runs at full camera-rate while the second keyframe rendering process only generates new synthetic images as the current camera leaves the view of the previous reference frame. Thus, each iteration we update the camera pose $\mathbf{T}_t^W$, via a pose composition $\mathbf{T}_t^W = \mathbf{T}_k^W \mathbf{T}_t^k$, and only change the keyframe camera as needed.

In this section, we discuss the details of the two parallel processes and describe how we choose the keyframe pose $\mathbf{T}_k^W$ to realize a robust localization system.

A. Semi-Dense Tracking

Given our ability to easily re-generate nearby keyframes, we choose semi-dense direct alignment as our tracking method for its speed and robustness in tracking with a small baseline between images. Our semi-dense tracker directly aligns a camera image I_t with a synthetically rendered keyframe image I_s by minimizing the photometric error between the two. Using the geometry imposed by the depth image I_d of the keyframe camera, we can back-project each pixel $\mathbf{u} = (u, v)$ in the keyframe image to a 3D point $\mathbf{p}^k$ in the keyframe coordinate frame by $\mathbf{p}^k = \pi^{-1}(\mathbf{u}, d) = d\mathbf{K}^{-1}\bar{\mathbf{u}}$ where $\mathbf{K}$ is a pre-calibrated pinhole camera matrix, $d = I_d(\mathbf{u})$ is the depth of the pixel, and $\bar{\mathbf{u}}$ is a homogeneous vector $\bar{\mathbf{u}} = (u, v, 1)$. To compute the photometric error, each of the 3D points must be transformed to the current camera coordinate frame by the relative transformation $\mathbf{T}_c^k$, i.e., $\mathbf{p}^c = \mathbf{T}_k^c \mathbf{p}^k$, then uncalibrated and projected into the camera image I_t by $\mathbf{u}_c = \pi(\mathbf{K}\mathbf{p}^c)$ where the projection function $\pi(\mathbf{p}) = (x/z, y/z)$ also includes dehomogenization. Then, the per-pixel photometric error between the current image and the keyframe image is

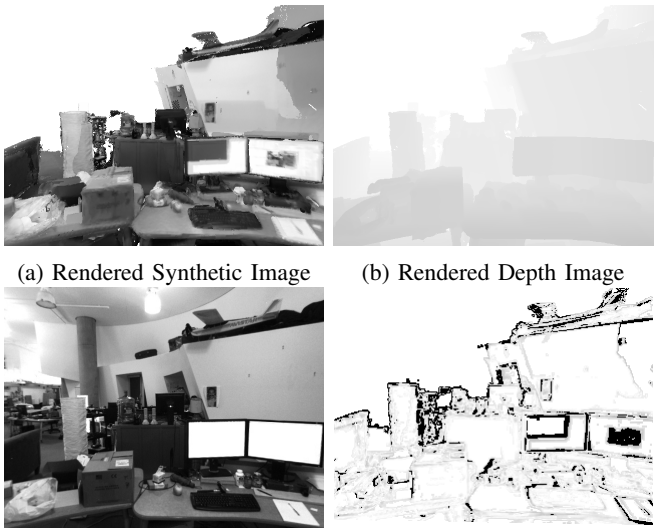
$$e(I_t, I_s, I_d, \mathbf{u}) = I_t(\mathbf{u}) - I_s(\pi(\mathbf{K}\mathbf{T}_k^c \pi^{-1}(\mathbf{u}, I_d(\mathbf{u})))). \tag{3}$$

We parametrize an update to $\mathbf{T}_k^c \in SE(3)$ as $\xi \in \mathbb{R}^6$ in the Lie Algebra $\mathfrak{se}(3)$. We use the exp operator, $\exp : \mathfrak{se}(3) \to SE(3)$ to map an incremental twist ξ in $\mathfrak{se}(3)$ to its corresponding pose in the Lie group $SE(3)$. We minimize the squared sum of the per-pixel error function

$$E(\xi) = \sum_i (I_t(\mathbf{u}_i) - I_s(\pi(\mathbf{K}\exp(\xi)\pi^{-1}(\mathbf{u}_i, I_d(\mathbf{u}_i)))))^2 \tag{4}$$

with Gauss-Newton iterations. We start with an initial estimate of ξ^0 and incrementally update with $\delta\xi^t$ by $\xi^{t+1} = \delta\xi^t \circ \xi^t$, where the $\circ$ operator denotes pose composition in $\mathfrak{se}(3)$. The incremental update is then

$$\delta\xi^t = -(J^T J)^{-1} J^T \mathbf{e}(\xi) \tag{5}$$



(a) Rendered Synthetic Image (b) Rendered Depth Image

(c) Undistorted Camera Frame (d) Residual Image

Fig. 2: Synthetic grayscale image (a) and depth image (b) rendered at the estimated camera pose are shown. Comparing to the input camera image (c), the images are visually aligned indicating successful tracking. Despite the large difference in the appearance of the images, our semi-dense alignment can correct for a global illumination and weigh down noisy regions shown as dark pixels in the residual image (d).

where J is the Jacobian $J = \frac{\partial \mathbf{e}(\xi)}{\partial \xi}|_{\xi=0}$ and $\mathbf{e}(\xi)$ is the error vector $\mathbf{e} = (e_1, e_2, ..., e_t)^T$. The derivation of this Jacobian is well-known, and more details can be found in [16].

While Eq. 4 is valid for images taken by the same camera under the brightness constancy assumption, synthetic images and real camera images are generated by different hardware under different lighting conditions. Moreover, rendered images have unmapped regions with no information, e.g. the ceiling, that are displayed with white pixels as shown in Fig. 2. Due to such differences, the appearance between rendered images and synthetic images vary both globally and locally.

To correct for the difference in the images, we apply the illumination adjustment technique introduced in [17] and refine the photometric error function as

$$E(\xi) = \sum_i (\alpha(e_i(\xi)) - \beta)^2 \tag{6}$$

where α is a robust error metric that defines the weight for the contribution of each pixel and β is the global illumination factor approximated as the median of per-pixel error e. The α term can help weigh down locally inconsistent regions while the β term can subtract out the global illumination difference between the two images. We use the Huber robust error metric to re-compute the α term each iteration and modify the update Eq. 5 to also include the weight term.

Finally, to speed-up the tracking process we selectively track only the pixels that are strong in gradient as done in [18] and also utilize a Gaussian image pyramid to track from the coarsest level to the finest level for better convergence.

B. Keyframe Rendering

We use OpenGL to generate synthetic keyframe image I_s and associated depth image I_d using a colored triangle mesh. We do not require a high-performance GPU but can render images using a software-only implementation of OpenGL, such as MESA Gallium llvmpipe, or with an embedded graphics unit such as Intel HD 4400 available on compact PCs used on modern MAVs [19], [20].

We render a keyframe grayscale image I_s by projecting a colored mesh of the environment into a desired keyframe camera pose $\mathbf{T}_k^W$. The 3D vertices $\mathbf{p}_i^W$ of the mesh are first transformed into the keyframe camera coordinates $\mathbf{p}_i^k = (\mathbf{T}_k^W)^{-1}\mathbf{p}_i^W$ by setting the model-view matrix. Then the vertices are uncalibrated and projected, similar to Eq. 3, and converted into normalized device coordinates (NDC). This additional conversion to NDC is necessary for OpenGL operations that expect each coordinate axis mapped to $[-1, 1]$.

An important detail to note is that the projection and uncalibration in OpenGL are provided by a single perspective matrix, i.e., glFrustum, which does not allow the use of a pinhole camera model. Thus, we instead uncalibrate and orthogonally project in two steps, i.e., $\mathbf{u}^{ndc} = P\tilde{\mathbf{K}}\mathbf{p}^k$ by first applying a modified pinhole camera model

$$\tilde{\mathbf{K}} = \begin{bmatrix} f_x & 0 & -c_x & 0 \\ 0 & f_y & -c_y & 0 \\ 0 & 0 & z_n + z_f & z_n z_f \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{7}$$

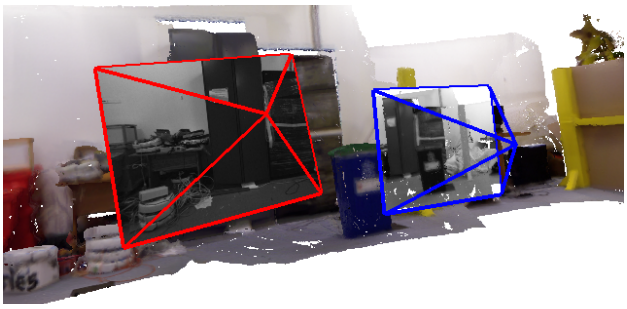


Fig. 3: A synthetic keyframe image (blue) can have as little as one third of its view overlapped with camera images (red) without losing the ability to track. The tracked camera image and the rendered keyframe images are shown in the imaging plane of each camera wire-frame.

where the third column is negated to correct for OpenGL's camera looking down the negative z-axis and the third row is added to correct for the difference [21] in depth mapping (clipping depth values z to $[z_n, z_f]$ then mapping to $[-1, 1]$) between the orthogonal projection and the perspective projection. Then applying the orthogonal projection matrix

$$P = \begin{bmatrix} 2/w & 0 & 0 & -1 \\ 0 & 2/h & 0 & -1 \\ 0 & 0 & \frac{2}{z_n - z_f} & \frac{z_n + z_f}{z_n - z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where w and h are width and height of the image, we can render a synthetic keyframe image.

When rendering an image, OpenGL implementations use a Z-buffer (depth buffer) to handle occlusions. Thus, without any additional computation, we can obtain a depth image by taking the normalized Z-buffer in $[0, 1]$ and converting it back to the metric space via inverse orthogonal projection:

$$z_{image} = \frac{2 z_f z_n}{(z_f + z_n - (z_f - z_n)(2 z_{buffer} - 1))} \tag{9}$$

After rendering is finished, we can asynchronously copy images in the GPU to local memory using the pixel buffer object (PBO). For performance reasons, we store the mesh on the GPU using the vertex buffer object (VBO) and render the frames using the frame buffer object (FBO).

C. Keyframe Selection

We render a new keyframe image as needed, i.e., when viewpoints are different, the tracking quality degrades, or a fixed interval has passed. While image alignment can work with as little as one third of overlapped regions, as illustrated in Figure 3, we would like the new keyframe image to have a significant overlap with upcoming camera images for reliable convergence. While we could render a keyframe at an expected future pose $\mathbf{T}_{t+m}^W$, we conservatively choose the current pose estimate $\mathbf{T}_t^W$ as a new keyframe pose.

When choosing the current pose estimate $\mathbf{T}_t^W$ as a keyframe pose, the accuracy of the estimate does not affect future tracking performance or accumulate error over time.

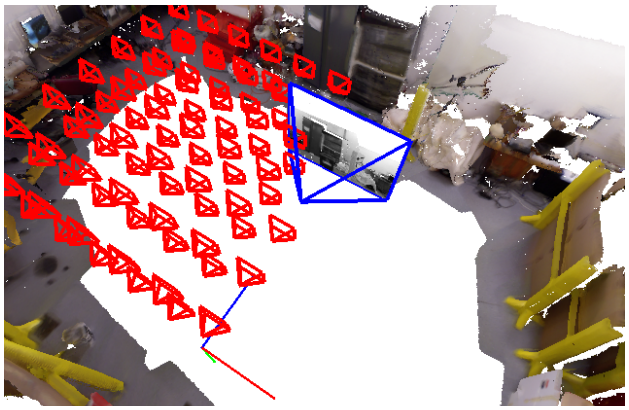


Fig. 4: Keyframe cameras (red) are uniformly placed in the mesh, varying in 6 DoF (only translation shown in this figure). For each keyframe location, a synthetic image is generated (shown inside the blue camera wireframe) and compressed down to a single ORB descriptor. The descriptors generated in this offline pre-processing stage are later compared online for relocalization.

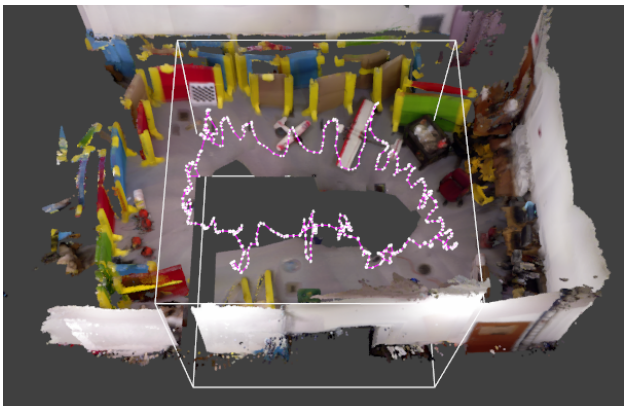


Fig. 5: A mesh of the Vicon room is constructed using a RGBD sensor and Kintinuous. The ground-truth odometry (purple) is provided to reduce errors in the mesh map.

An image and a depth map is essentially a dense 3D point-cloud and the role of the keyframe pose is only to segment out a small portion of the mesh, bounded by the keyframe camera frustum, to use for tracking. Therefore, errors in the pose estimate when initializing a new keyframe only result in suboptimal keyframe location selection and do not largely affect the accuracy of future estimates.

However, in the case of tracking failure, our estimate $\mathbf{T}_t^W$ is significantly different from the true camera position so that a keyframe generated at the estimated pose is not likely to have any overlap with incoming camera images. In this case, we initiate a relocalization procedure to choose a different keyframe pose, as described in the next section.

D. Relocalization

To recover from tracking failure, we need to find a keyframe camera capable of producing a synthetic image that overlaps with the current camera image. We adopt the approach of SLAM systems [18], [22] that save and compare past keyframes to the latest camera image for recovery. While relocalization in such SLAM systems is only possible if the camera is near a saved keyframe, in the case of relocalizing in a known map, we need to recover from anywhere in the map at its full scale.

To improve the scalability of relocalization, we must bound the number of keyframe locations, bound the online search region, and compress the information in each keyframe image. First, we can bound the number of potential keyframe locations by uniformly placing virtual cameras in the map, as shown in Figure 4, and limiting relocalization attempts to the discrete set. Then to reduce the search region, we define a conservative search radius using the latest velocity estimate and the time since last failure. Lastly, to compress the information in each keyframe image, we down-scale the keyframe image to a small patch of size 80 by 64 and extract a single ORB [8] descriptor that covers the whole image. This technique, similar to the GIST-BRIEF loop-closure method [23], compresses a single keyframe down to 256 bits, significantly reducing the memory requirement and making online recovery extremely fast; we extract an ORB descriptor once, and test each keyframe for overlap by computing the hamming distance of two 256 bit vectors. The high scalability of our similarity metric makes it more suitable for the large-scale relocalization problem compared to other image-space similarity metrics, such as mutual information or normalized information distance (NID), that require the whole image for each keyframe as well as substantial online computation, e.g., joint entropy calculation, that cannot be pre-computed offline.

III. Experimental Results

We tested STAR algorithm both offline on the publicly available TUM RGBD dataset [24] and online in GPS-denied indoor environments. We used a laptop with 2.7 GHz i7 processor and an Intel HD 4400 embedded GPU to process the data. While we could use a software-only implementation of OpenGL for slower sequences, we utilized the low-cost embedded GPU available on modern MAVs for higher efficiency. For the monocular camera, we used a PointGrey Flea3 camera at 640x512 resolution running at 50 FPS. Details of each experiment are shown in this section.

A. Meshing

We generated a colored mesh of each test environment using a Kinect RGBD sensor. We used the open-source FastFusion algorithm [25] for the public RGBD dataset and Kintinuous [15] for other environments. The drift in the odometry during mesh building often caused the mesh to be deformed, or created flaws in the texture when closing the loop. While STAR could track camera poses using a deformed mesh map, for quantitative analysis, we used ground-truth Vicon poses to isolate tracking errors from errors in the map. A mesh map of the Vicon room, built with the ground-truth, is shown in Figure 5.

TABLE I: Quantitative analysis of STAR

Experiment	Distance Travelled (m)	Position Error (cm)	Rotation Error (deg)
TUM Office	21.57	2.13 ± 1.29	0.81 ± 0.36
Single Loop	15.44	14.13 ± 8.44	2.92 ± 2.37
Aggressive 1	19.73	13.79 ± 16.79	3.92 ± 3.98
Triple Loop	36.38	13.09 ± 6.77	3.32 ± 1.48
Aggressive 2	13.98	13.66 ± 8.60	3.73 ± 2.05

B. TUM RGBD Benchmark

We tested STAR on a TUM RGBD sequence [24] captured in an office environment. We first generated a mesh map of the environment using both the RGB and depth sequences, then tracked the trajectory using only the RGB images as the input to our algorithm. Note that tracking the same images used to build the mesh simplifies the tracking problem since there are no photometric differences between the rendered images and the camera images. However, unmapped areas in the mesh still create large artifacts in the rendered images.

The simplified tracking problem demonstrates STAR algorithm under an ideal condition with fixed illumination, identical sensors, and similar viewpoints. These conditions are possible in a real environment when tracking in a closed room with controlled lighting and tracking with the same camera used to build the mesh.

Localization results are shown in Table I, where the mean translational error is 2.13 cm, and the mean rotational error is 0.81 degrees. While these results are difficult to achieve in a more realistic scenario, they demonstrate the ability of STAR to accurately localize the camera in an ideal environment.

C. Vicon Experiment

We evaluated STAR in a Vicon room shown in Figure 5. Using a mesh built with ground-truth odometry, we tracked a slow camera sequence (Single Loop) and a faster sequence with rapid rotations (Aggressive 1). After altering the appearance of the room, we re-mapped and tested two other sequences (Triple Loop and Aggressive 2).

The localization error of STAR on the sequences is shown in Table I, where the mean translational error across the different datasets was approximately 14 cm, and the rotational error was between 3 to 4 degrees. The translational error was greater along the z-axis (shown in Figure 10), for example, being 9.43 cm for the Triple Loop sequence, while the error along the xy-axes were 4.05 cm and 4.09 cm. A similar pattern was observed for the pitch angle where the error was approximately double of the other axes. Higher errors along the z-axis and in the pitch angle are suspected to be the result of a lack of horizontal edges in the test environment, observable in Figure 7.

The average time taken to estimate the camera pose for all the sequences was under 20 ms, being able to track the 50 FPS camera in real-time. Shown in Figure 6, the average time for tracking a frame in the Aggressive 1 sequence was 14.42 ms, and the Single Loop sequence was 18.17 ms. The variations in the tracking time were dependent on the amount of texture in the sequences, e.g., Aggressive 1 had fewer gradient pixels to track on, where the camera was often looking at scenes that were not completely mapped in the mesh. The average rendering time for the Aggressive 1 sequence was 21.55 ms using the embedded GPU, while the rendering time for the Single Loop sequence was 190.36 ms using the MESA Gallium llvmpipe software renderer.

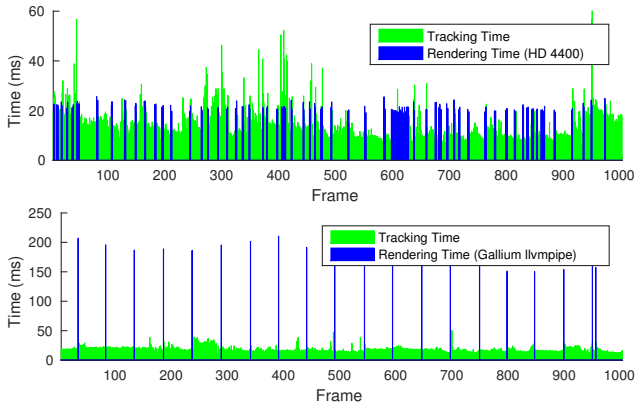


Fig. 6: Timing results for Aggressive 1 using an embedded GPU and Single Loop using Gallium llvmpipe software renderer. Software rendering is sufficient for slow moving sequences, but has difficulties dealing with fast rotations. It can be observed that some areas in the Aggressive 1 sequence contains more keyframe generation due to fast rotations.

D. Comparison to the state-of-the-art

Despite using an incomplete and imperfect mesh, STAR accurately estimated camera trajectories, with similar performance to the state-of-the-art localization methods [4], [6], [13]. In this section, we discuss how a direct comparison to these methods is difficult due to differences in the datasets and the validation method.

Lynen et. al [6] combined sparse features with an IMU [6], achieving a mean translational error of 17 cm and a mean rotational error of 0.32 degree in a large scale environment. While the rotational error is much smaller than the error of STAR, it is difficult to directly compare the results as the dataset used by Lynen et al. were restricted to roads, e.g. captured on a bicycle or along the sidewalk, without exercising the full range of 6 degrees of freedom (DoF) motion. The dataset used to evaluate the algorithms were also deliberately within moderate view-point changes of the original dataset used to build the map.

Moreover, the feature-based localization of Lim et. al [4] achieved varying results on different datasets, with 10.8 cm translational error and 1.6 degree rotational error being the run with the highest success rate of 95%. While Lim et. al exercised the full range of all 6 DoF, a direct comparison to STAR is difficult due to error metrics basing on only the successful attempts. In our STAR implementation, no estimates were rejected, with the worst translation error being 53.19 cm. This design choice, which increased the mean error, was to later do the rejection (reduction) in an filtering framework fusing in an IMU.

Fig. 7: The Aggressive 1 sequence, where the first row shows the camera images and the second row shows rendered images at the estimated poses. Despite the imperfections in the mesh, shown as white holes in the rendered views, renderings at our pose estimates accurately match the camera images, indicating successful camera tracking. Refer to `https://groups.csail.mit.edu/rrg/star` for the full sequence.

Lastly, the most recent mesh-based localization work [13] reported a RMS translational error of 3.73 to 7.42 cm, and a RMS rotational error of 0.35 to 0.91 degrees with a 93% success rate. Comparing to this method is also difficult due to the dataset being captured on a ground-based vehicle (3 DoF only), the trajectory for map-building being the same trajectory for tracking, and the dataset being processed frame by frame offline[1].

In terms of timing, STAR performed over 25 times faster than the 2 Hz tracking rate reported by the state-of-the-art mesh-based localization method [13], and did similar to the 23 to 37 ms [4] and 24.5 ms [6] of feature-based methods.

E. Relocalization Experiment

We tested our relocalizer on the Aggressive 1 sequence, where 100 different locations were queried for the closest match within a fixed search window of 2 meters. Some representative matches are shown in Figure 8 where the relocalizer accurately retrieved a keyframe with an average distance of 45 cm to the query frame. However, due to perspective differences in the recovered keyframe and the camera image, recovering the pose with direct alignment was successful 73% of the time. When successful, recovery took a nearly constant 0.21 ms average time for checking approximately 700 nearby keyframes. This extremely cost-efficient keyframe query allowed us to broaden the search region when required. However, in the case of failure, recovery took an arbitrarily long amount of time, until the camera left the regions where the relocalizer deterministically failed to find a keyframe with a significantly similar viewpoint. We also experimented with NID as the similarity metric: it took on average 2440 ms, and the average distance was 147 cm.

[1]discussed in personal communication

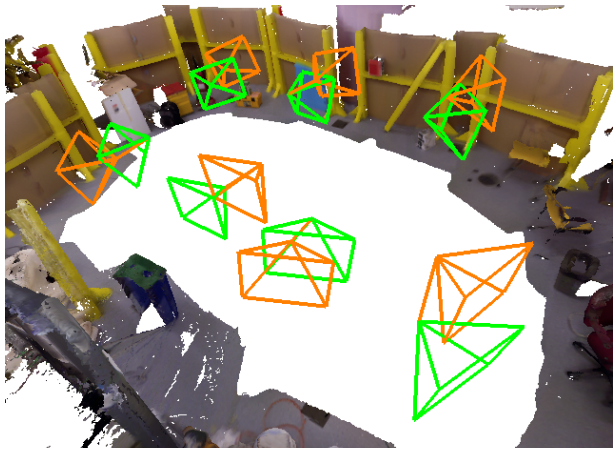


Fig. 8: Keyframes returned by our relocalizer are shown in orange, where each keyframe is connected to the query camera frame shown in green. For all of the queries, a perceptually similar keyframe was successfully found.

Using a single ORB descriptor in the relocalizer was several orders of magnitude faster, and returned a closer keyframe.

F. Office Experiment

In addition to the Vicon room, we tested STAR in a larger indoor environment with changes in illumination. Due to the varying illumination from nearby windows, there was a significant difference between the camera images and rendered keyframes. Moreover, small objects in the scene created occlusions, reducing the coverage of the mesh. While we do not have a ground-truth trajectory for this sequence, the similarity between the camera images and renderings at the pose estimates indicate successful tracking. Representative frames are shown in Figure 9 for qualitative assessment.

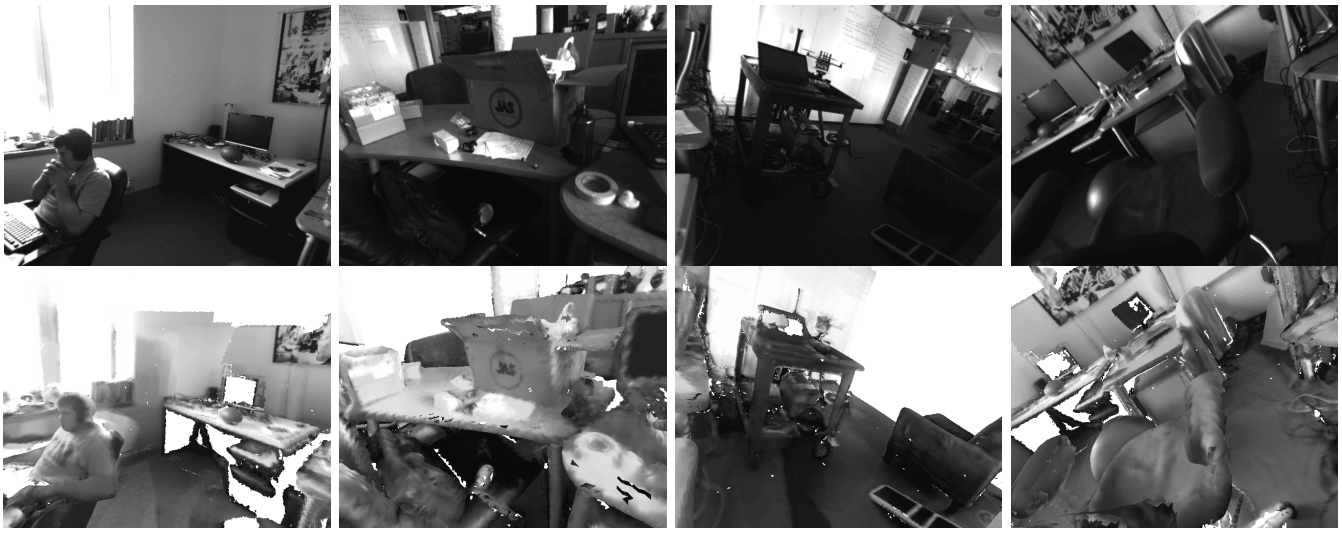
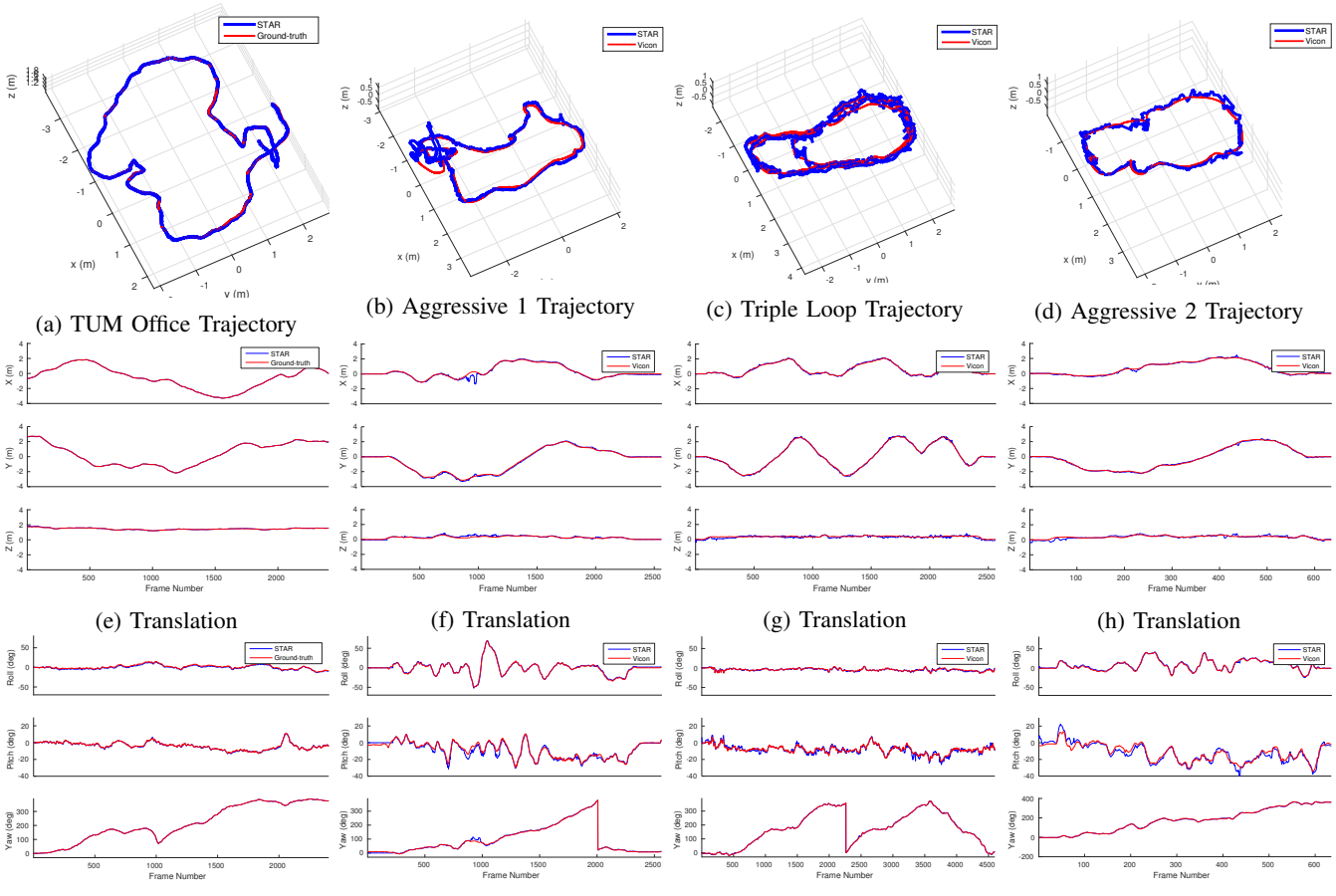


Fig. 9: RRG Office sequence shown for qualitative analysis. Due to changing illumination from the windows, there is a significant difference between the camera images and the rendered keyframes. Furthermore, there's a person in the sequence dynamically affecting the environment. Our algorithm is able to reject these areas for robust tracking.



(a) TUM Office Trajectory

(b) Aggressive 1 Trajectory

(c) Triple Loop Trajectory

(d) Aggressive 2 Trajectory

(e) Translation

(f) Translation

(g) Translation

(h) Translation

(i) Rotation

(j) Rotation

(k) Rotation

(l) Rotation

Fig. 10: The trajectory estimated by STAR (blue) and the ground-truth (red) are shown for various sequences.

IV. Conclusions

We have proposed Simultaneous Tracking and Rendering where we can localize a monocular camera given a mesh map of the environment. We achieved similar speed and accuracy to the state-of-the-art feature-based localization methods with a mean translational error of 14 cm and a mean rotational error less than 4 degrees, while using a planning-capable and view-independent mesh map. We improved the speed of mesh-based localization by more than 25 times compared to the current state-of-the-art. With real-time tracking using a planning-capable mesh map, STAR takes a step toward a fully autonomous monocular vision-based MAV.

Acknowledgment

This research was funded by the ONR under MURI N00014-10-1-0936 and the ARO MAST CTA. Their support is gratefully acknowledged.

References

[1] A. Bry, A. Bachrach, and N. Roy, "State estimation for aggressive flight in GPS-denied environments using onboard sensing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[2] M. F. Fallon, H. Johannsson, and J. J. Leonard, "Efficient scene simulation for robust monte carlo localization using an RGB-D camera," in *IEEE International Conference on Robotics and Automation*, 2012.

[3] T. Sattler, B. Leibe, and L. Kobbelt, "Fast image-based localization using direct 2D-to-3D matching," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011.

[4] H. Lim, S. N. Sinha, M. F. Cohen, M. Uyttendaele, and H. J. Kim, "Real-time monocular image-based 6-DoF localization," *The International Journal of Robotics Research*, 2015.

[5] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua, "Worldwide pose estimation using 3D point clouds," in *Computer Vision–ECCV 2012*, 2012.

[6] S. Lynen, T. Sattler, M. Bosse, J. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: large-scale, real-time visual-inertial localization," in *Robotics: Science and Systems*, 2015.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, 2004.

[8] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011.

[9] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors," *International journal of computer vision*, 2005.

[10] W. Churchill and P. Newman, "Practice makes perfect? managing and leveraging visual experiences for lifelong navigation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012.

[11] W. Maddern, M. Milford, and G. Wyeth, "CAT-SLAM: probabilistic localisation and mapping using a continuous appearance-based trajectory," *The International Journal of Robotics Research*, 2012.

[12] E. Johns and G.-Z. Yang, "Feature co-occurrence maps: appearance-based localisation throughout the day," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[13] G. Pascoe, W. Maddern, A. D. Stewart, and P. Newman, "FARLAP: fast robust localisation using appearance priors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[14] G. Caron, A. Dame, and E. Marchand, "Direct model based visual tracking and pose estimation using mutual information," *Image and Vision Computing*, 2014.

[15] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *The International Journal of Robotics Research*, 2015.

[16] S. Baker and I. Matthews, "Lucas-kanade 20 years on: a unifying framework," *International Journal of Computer Vision*, 2004.

[17] T. Gonçalves, A. Comport, *et al.*, "Real-time direct tracking of color images in the presence of illumination variation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011.

[18] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: large-scale direct monocular slam," in *Computer Vision–ECCV 2014*, 2014.

[19] K. Ok, D. Gamage, T. Drummond, F. Dellaert, and N. Roy, "Monocular image space tracking on a computationally limited mav," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2015.

[20] M. N. Galfond, "Visual-inertial odometry with depth sensing using a multi-state constraint kalman filter," PhD thesis, Massachusetts Inst. of Technology, 2014.

[21] R. S. Wright, N. Haemel, G. M. Sellers, and B. Lipchak, *OpenGL SuperBible: comprehensive tutorial and reference*. 2010.

[22] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, 2007.

[23] N. Sünderhauf and P. Protzel, "BRIEF-gist-closing the loop by simple means," in *Intelligent Robots and Systems (IROS), International Conference on*, 2011.

[24] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.

[25] F. Steinbrucker, J. Sturm, and D. Cremers, "Volumetric 3D mapping in real-time on a CPU," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.