SplattingTAPIR: Utilizing TAPIR and 3D Gaussian Splatting for Efficient and Robust Reconstruction

Tim Nguyen[1,2]

[1]Boston Latin School
[2]School of Engineering and Applied Sciences, Harvard University

April 26, 2024

Abstract

The recent rise of foundation models has been shown to have immense power in pattern recognition and performing "zero-shot" predictions. A key field that could greatly benefit from leveraging these pattern recognition techniques is computer perception. Whether it is the Simultaneous Localization And Mapping (SLAM) algorithms found in our autonomous vehicles or Structure from Motion (SfM) algorithms found in 3D mapping, computer perception has played a pivotal in bridging the gap between computers and our 3D world. While these classical techniques are generally robust in perfect scenarios, the imperfect conditions present in the real world, such as texture-less materials, often create critical failure points for these algorithms. I present SplattingTAPIR, a deep-learning twist on a classical visual odometry problem. SplattingTAPIR is a robust and efficient end-to-end pipeline that can transform a set of 2D images into a 3D scene using Google Deepmind's Tracking Any Point with per-frame Initialization and temporal Refinement (TAPIR)[9] network for 2D keypoint detection with a modern twist on monocular pose estimation and 3D reconstruction through the use of the Depth Anything foundation model[28], SIM-Sync[29], and 3D Gaussian Splatting[14].

Introduction

Constructing a 3D scene from a series of images has always been an area of great research interest. Pipelines, like COLMAP[24], have been immensely popular as they produce a 3D reconstruction of a scene through a series of photos. COLMAP's pipeline uses keypoint detectors like SIFT[16] for feature extraction and then uses a variety of classical techniques, including epipolar geometry to extract the 3D point clouds and camera poses from the series of photos. These results are refined using a bundle-adjustment algorithm, like the Ceres Solver[1], which tries to minimize the reprojection between the 3D points and their corresponding 2D point from the images. COLMAP has been especially popular with the rise of Neural Radiance Fields (NeRFs) as it allows camera poses to be extracted from the set of images at relatively high precision. However, keypoint detectors like SIFT often produce false matches are struggle when there is a high presence of feature-less materials, like solid-colored walls.

Foundation models have proven to be extremely popular recently in natural language processing. With models like OpenAI's GPT-4[20] powering chatbots that perform human-like conversations and reasoning, these models have proven to perform exceptionally well when it comes to zero-shot" predictions. SplattingTAPIR aims to harness this by using models like Tracking Any Point with per-frame Initialization and temporal Refinement (TAPIR)[9] and Depth-Anything [28] to replace some of the classical components within the Structure from Motion pipeline. Using a video sequence of object "scans," random keypoints are tracked using TAPIR and lifted using Depth-Anything, replacing the traditional process of relying on epipolar geometry for lifting 2D keypoints into the 3D space. Once the points have been lifted, the camera pose and the 3D point cloud are extracted using SIM-Sync in which the distance between corresponding points is minimized. The point clouds and camera poses are then used to perform a 3D reconstruction of the scene using 3D Gaussian Splatting.

Related Work

Structure from Motion: Similar pipelines that offer a 3D reconstruction of a scene from a series of photos include COLMAP[24], OpenMVG[18], and Meshroom[11]. These photogrammetry pipelines use classical techniques such as keypoint detectors and surface reconstruction to form the scene. Other programs such as Polycam[22] can combine LiDAR and IMU with the photogrammetry sequence to create an even more robust reconstruction.

Radiance Fields: In addition to Structure from Motion pipelines, SplattingTAPIR uses 3D Gaussian Splatting for the final scene representation. Related works that predate 3D Gaussian Splatting include NeRFs[17] like Zip-NeRF[3] and Mip-NeRF 360[2].

SpalttingTAPIR

The SplattingTAPIR pipeline can be broken into three major points: feature and depth extraction, camera pose es-

timation through bundle adjustment and 3D scene representations. Each step contributes to the end goal of transforming a video or a sequential set of frames into a 3D scene. TAPIR and Depth-Anything are primarily used in the first step during the feature and depth extraction, SIM-Sync is used for bundle adjustment and camera pose extraction, and 3D Gaussian Splatting is used to transform the camera poses and mesh cloud into a comprehensive and hyper-realistic 3D scene.

Algorithm 1 SplattingTAPIR's Pipeline

$Chunk \leftarrow 64$ ▷ Default Chunk Size
$nPoints \leftarrow 5000$ ▷ Default number of points to track
$Vid \leftarrow input.mov$
$useChunk \leftarrow True$
$Frames, Height, Width, nFrames \leftarrow processVid(Vid)$
if $useChunk == True$ **then**
    $nFrames = Chunk$
end if
$pts \leftarrow rndPts(Height, Width, nFrames, nPoints)$
$TAPIR = compileModel(Height, Width, nFrames, pts)$ ▷ Runs a dummy inference to pre-compile and caches the TAPIR model for future inferences. Can also fetch pre-compiled models.
$track, visibles = TAPIR(Frames, Height, Width, pts)$
$track = cleanPoints(track, visibles)$ ▷ Remove points that are not visible in that specific frame.
$depthMaps = DepthAnything(Frames)$
$liftedpts = depthLift(depthMaps, track)$
$CameraPoses, PointCloud = SIMSync(liftedpts)$
$outputscene = 3DGS(CameraPoses, PointCloud)$ ▷ The outputscene can be exported as a rendered video or an interactive experience where one can "walk" through the scene using a viewer.

Feature and Depth Extraction

This process harnesses the power of both TAPIR and Depth-Anything to track 2D keypoints from the images and "lift" them into the 3D space.

Background: Classical Approach

Classical approaches that are found in the SfM when it comes to feature and point extraction often involve using keypoint detector algorithms like SIFT[16] uses the Gaussian blurring process, where each image is resized and blurred eight times each, for a total of sixty-four image transformations [25]. This process removes noise and finds "high-quality" points that are able to last through these transformations. However, this process can fail when there is a huge presence of texture-less materials, such as walls. Once these points are found, the "descriptors" of each point, or vectors that describe each keypoint, are used to match up corresponding keypoints. The corresponding keypoints are then "lifted" using epipolar geometry, where the epipolar lines are lines from the optical centers of the images that are shot through the keypoints, and intersect at a hypothetical point in the 3D space. The intersection of these lines is used to determine the 3D point of these 2D keypoints. Through this process, the relative camera rotation R, the relative camera translation t, and the 3D point X are found. P is the projection matrix, which is found using the camera intrinsic matrix. Since this is all in the relation between two images, these transformations can become "global" when the rotations and translations of each image pair relate back to the first pair of images.

$$x = P\left[R \mid t\right] X \tag{1}$$

Tracking with TAPIR

TAPIR, or Tracking Any Point Refinement is very different compared to the classical keypoint detector and descriptor networks built on JAX [4]. TAPIR is primarily focused on tracking points throughout a video but is leveraged as a keypoint detector.

The TAPIR model is loosely based upon TAP-Net and Persistent Independent Particles (PIPs) [12]. As mentioned in the paper, the TAPIR model takes in the given point and tracks it with every other frame to create an initial estimate, which is then used to be refined in which the points are checked with other local neighbors at a higher-quality stage. This refinement allows TAPIR to compute the path it thinks the point has taken throughout the given frames. The combination of TAP-Net and PIPs in the sense that it is leveraging the TAP-Net's ability to estimate the points indecently throughout the frames and PIPs' ability to create neighbors to compare these features on a much broader scale [9].

Given that JAX is built upon Just-In-Time compilation (JIT), where the Python code is converted into a more efficient abstract form with jaxpr[4], this step allows JAX to run efficiently at the cost of the initial compilation step taking much longer than subsequent runs. To address this issue, SplattingTAPIR compiles a dummy model with the same sized input parameters (number of frames, width of the frame, height of the frame, amount of points tracked), and caches this model for later use. This model can then be reused whenever the input parameters are of the same size.

Although it is best to perform point tracking on every point throughout all the frames, performing inferences on longer videos often causes out-of-memory issues. To address this, a chunk-based modification of TAPIR is used. The randomly selected points to be queried are broken up into chunks. The size of each chunk can be adjusted depending on the number of frames processed and the amount of GPU memory available, but larger chunks typically yield faster results. The `query_chunk_size` parameter model is also adjusted and is set to the size of each chunk for the most efficient and optimal runs when it comes to point tracking. When compiling the dummy model, SplattingTAPIR sets the number of points with the chunk size to ensure efficient compilation and caching.

The TAPIR model produces a `track` array of where the points are throughout the video, and a `visible` array of which points are visible in each frame. Points that are not visible in the frame tend to be inaccurate compared to visible points. The `track` is cleaned by removing points that

are not `visible`.

Lifting Points with Depth-Anything

Once the points have bee been tracked with TAPIR, these points can then be lifted using Depth-Anything[28]. This replaces the epipolar geometry step found in classical pipelines and instead uses a foundation model to estimate depth and "lift" these 2D keypoints into 3D.

Depth-Anything is a monocular depth estimation foundation model, which means that it is a machine-learning model that can predict the depth that each of the objects in the image has in relation to the center of the camera using a single image. This makes Depth-Anything unique as classical approaches to depth estimation often require stereoscopic cameras, in which two cameras are taking the same image with a slight offset. This offset is crucial in determining depth as epipolar geometry can be used to estimate depth, which is the same notion behind how our eyes work and why many classical SfM algorithms work with image pairs. However, in many cases, the depth maps generated through this method aren't super detailed or robust and are very susceptible to noise.

Depth- Anything instead leverages the power that foundation models have when it comes to "zero-shot" predictions and uses it to extract depth from images. Depth-Anything can perform relative depth estimation where the depth of each point is in relation to other points in the image but also can perform metric depth estimation[28]. Metric depth estimation allows Depth-Anything to accurately predict how far objects in the frame are from the camera in the desired unit. SplattingTAPIR leverages metric depth estimation as this is the closest method to getting stable depth estimation for all the frames within the video, as relative depth estimation can change on each frame depending on the maximum and minimum depth found.

Depth-Anything can robustly perform these zero-shot predictions in part due to how the model is designed and the data used to train it. Due to the limited amount of labeled depth maps, Depth-Anything's dataset uses a data engine to accurately segment unlabeled images [28]. This allows Depth-Anything to be trained on 1.5 million labeled images and over 62 million unlabeled images, allowing the model to be trained in a comprehensive manner[28]. Depth-Anything also leverages large vision models, like DINO-V2[21]. DINO-V2 can perform high-quality object segmentation with images, and since the depth of an object is based upon its shape, DINO-V2 uses it as a loss to ensure that it is making high-quality predictions[28].

In the SplattingTAPIR pipeline, the image frames from the sequence are fed into Depth-Anything to perform metric depth estimation. Once the meteric depth map is found, the corresponding depth at each keypoints is found and lifted into the 3D space. The points are lifted using the metric depth values, but also take into account the focal lengths and distortion that come naturally with the camera lenses.

Pose Recovery

Once the points are tracked and lifted using TAPIR and Depth-Anything, the camera poses and the 3D point cloud need to be extracted. In a traditional SfM pipeline, epipolar geometry is used to create an initial estimate of the poses and the point cloud and is optimized using bundle adjustment. Instead, the poses are optimized with the lifted keypoints using SIM-Sync, which has many commonalities with bundle adjustment.

Background: Classical Bundle Adjustment

Since classical approaches often have images being reliant on one another for poses, this means that there is a graph of poses and points that are in relation to each other. Bundle adjustment is essentially a non-linear least-square sums problem, and can be denoted as shown.

$$\sum_{n=1}^{i} \|P_n - a_n\|^2 \tag{2}$$

With P_n being the 3D triangulated point being reprojected back to the 2D space and a_n being the corresponding 2D image keypoints' coordinates [27]. P_n can be found using the previous equation that was used to calculate the triangulated 3D points from the relative translation matrix and rotation vector (1). This equation can be used once again, but instead of solving for the 2D coordinates of the 3D triangulated points, and also uses the global rotation matrix and translation vector instead of the relative matrices from the epipolar estimation. Bundle adjustment aims to minimize the errors found with the reprojections, which should yield an accurate result.

Popular bundle adjustment packages like the Ceres Solver[1] can handle complex problems, but due to the nature of the problem itself, it is very hard to efficiently find the most optimal solution. Because bundle adjustment is often a large-scale minimization problem, the constraints of time and computational power often mean that these solvers coverage at a "false minimum," in which the problem is partially optimized, leading to wrong poses and point clouds.

SIM-Sync

SIM-Sync[29] provides a more robust solution while also extracting camera poses. SIM-Sync can come up with a *certifiably optimal* solution, or one that is robustly proven to be the correct solution. SIM-Sync proposes to view the minimization problem as a quadratically constrained quadratic program (QCQP), or which it is trying to minimize a set of variables within a boundary[7]. In this case, the 3D keypoints are minimized and the bounds are sanity checks to ensure that only possible solutions can be found.

Within the SplattingTAPIR pipeline, the 3D points are first cleansed before running through SIM-Sync. Outlier points, such as ones that have negative depth values, depth values that are too large, and ones that fail a two-view homography check are removed. The cleansed points are then fed into SIM-Sync to produce the camera poses and point clouds, which will then be passed to be used to create immersive 3D scenes.

3D Scene Representations

The last major part of SplattingTAPIR's pipeline is the Radiance Fields. Unlike the traditional photogrammetry process where the triangulated points are connected using a surface reconstruction program like Poisson [13]. These programs try to connect points in the most efficient and watertight manner but tend to create unrealistic and jagged renderings. They are also super reliant on the amount of high-quality 3D triangulated points produced. Models with a lot of keypoints generated typically create a more detailed mesh, but the accuracy of these models is dependent on how high-quality these points are.

Neural Radiance Fields (NeRFs)

Recently, there has been a rise in radiance fields, like NeRFs [17] where rays or frustums are cast from 2D images and then used to calculate the opacity of the clouds at each step of the ray, NeRFs can generate much higher quality and precise models. However, NeRFs are heavily reliant on a network of NLPs, and they can become very computationally expensive to render when there is a huge set of photos to work with or a large scene to render. Current state-of-the-art NeRFs like Mip-NeRF 360 [2] take up to 68 times slower than 3D Gaussian Splatting while providing similar render quality restyle when comparing PSNR, it is evident that 3D Gaussian Splatting [14] is the way forward for an efficient and robust pipeline [15].

3D Gaussian Splatting

Unlike NeRFs, according to the paper, 3D Gaussian Splatting creates "splats," which are cloud-like blobs that expand or shrink depending on the image. Like NeRFs, these blobs are overfitted. These "splats" are decimated or joined depending on the size of the [14]. These splats overlap to form objects and create a hyper-realistic scene. By integrating 3D Gaussian Splatting as our method of scene reconstruction, SplattingTAPIR is a pipeline that can create robust and efficient 3D scene reconstructions from a set of 2D images.

Results

The results of SpalttingTAPIR can be split into two parts: comparing TAPIR with classical keypoint detectors like SIFT and the performance of the SplattingTAPIR pipeline compared to a comparable classical pipeline. All tests were performed on a Lambda Vector workstation that featured an AMD Ryzen with a 64-core CPU, 512 GB of RAM, and two NVIDIA RTX A6000s GPUs designed for machine learning.

Comparing Keypoint detectors

For the TAPIR models, two models are used, one was pre-compiled to process two frames at a time and track a total of 200 keypoints (TAPIR-2). The other model was pre-compiled to process a set of 50 frames and track a total of 5000 keypoints (TAPIR-50). Each of the keypoints that are tracked with the TAPIR models is randomly generated with no pre-processing involved. For these tests, the KITTI Visual Odometry Dataset[10] was used. For these tests, TAPIR and SFIT are added to the classical SfM pipeline built using OpenCV[5] and using epipolar geometry techniques. This pipeline is very similar to the one found in my previous paper, "Lightweight Low-Poly Photogrammetry with Object Classification Validation with OpenCV and PointNet"[19]

Table 1: Average Time to Process 50 Frames in Seconds

KITTI #	TAPIR-50	TAPIR-2	SIFT
01	15.06	7.34	5.12
02	14.07	7.08	5.07
03	14.04	7.08	5.81
04	14.37	7.08	5.36
05	14.29	7.08	4.89

Table 2: Average Translation Deviation from Ground Truth Path (unit vectors)

KITTI #	TAPIR-50	TAPIR-2	SIFT
01	1.19	1.19	13.91
02	6.28	6.28	12.96
03	2.14	2.13	3.60
04	4.26	4.27	4.28
05	6.46	6.44	6.85

Table 3: Average Rotation Deviation from Ground Truth Path (degrees)

KITTI #	TAPIR-50	TAPIR-2	SIFT
01	2.52	2.76	2.35
02	0.31	1.23	0.15
03	0.21	1.09	0.14
04	0.22	0.52	0.17
05	0.30	0.35	0.26

TAPIR's performance is slightly slower than SIFT as seen on Table 1. The original TAPIR model is used with the chunk set at the default value of 16, but the only parameters changed are the input and output size. The input and output sizes are changed to closely match the original resolution of the images in the KITTI dataset. TAPIR-50 on average is 173% slower than SIFT and TAPIR-2 is on average 35% slower than SIFT. TAPIR-50 is looking at 50 frames and trying to track 5000 points throughout all of those points, while the TAPIR-2 model is only looking at 2 frames at a time and trying to track 200 points.

In regards to pose accuracy, SIFT experiences over two times greater translation drift than any of the TAPIR models while both SIFT and TAPIR perform relatively similarly with rotational deviation [Tables 2, 3]. The translational deviation is measured by calculating the distance from each ground-truth point to the corresponding camera pose found with TAPIR and SIFT.

When plotting the ground truth with the initial poses generated by these models as seen in Figures 1, 2, 3, 4, 5,

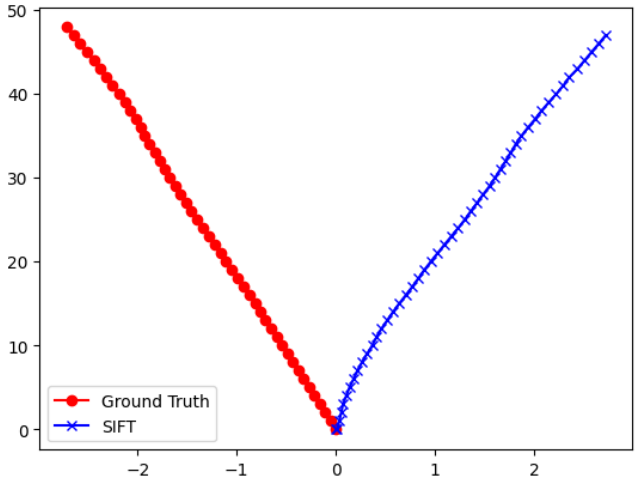

Figure 1: Initial SIFT poses on KITTI-00

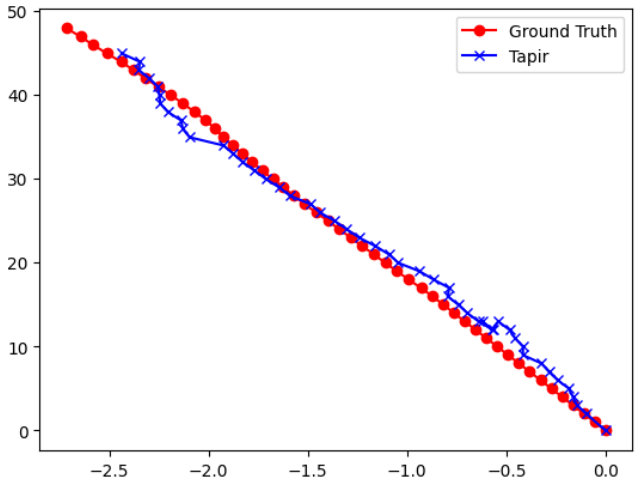

Figure 2: Initial TAPIR-2 poses on KITTI-00

and 6, TAPIR tends to produce tighter poses that those found with SIFT, highlighting the power these machine learning models have. This is especially apparent with dataset KITTI-00, where SIFT's trajectory is in the complete opposite direction.

In figures 7, 8, and 9, These are the matches shown after being filtered using RANSAC. These are the points used for 3D triangulated points and camera pose estimation. These images are the first and second frames of KITTI dataset 01. As shown, TAPIR can track "texture-less" aspects of the image, like parts of the road, with greater confidence and with fewer points than that of SIFT.

SplattingTAPIR Pipeline Performance

In these next tests, the performance of SplattingTAPIR is compared with that of the classical SfM pipeline built using OpenCV. For the bundle-adjustment step, PyCeres[23] is used. Note: SplattingTAPIR uses the TUM dataset[26] instead of KITTI. TUM is an indoor dataset of office space, while KITTI is one collected with an autonomous vehicle and suite of sensors. Due to this, the relative time taken to run each program and errors will be looked at and com-

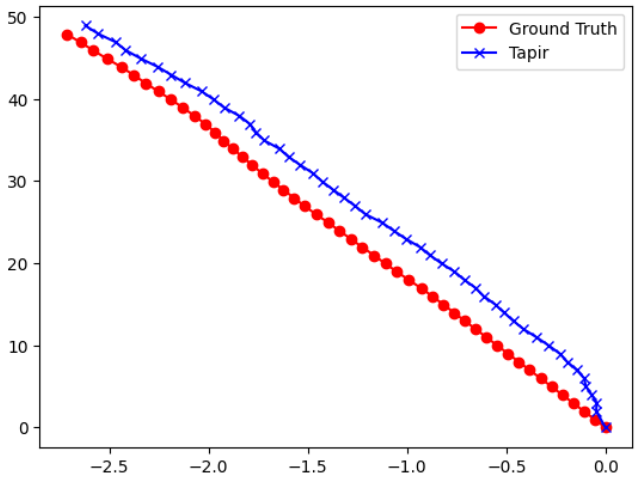

Figure 3: Initial TAPIR-50 poses on KITTI-00

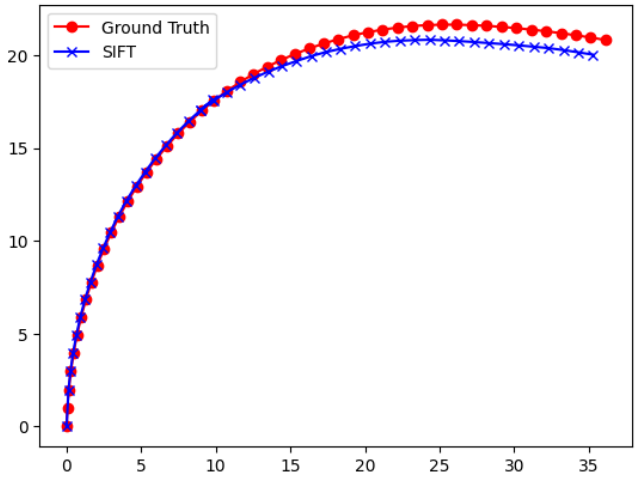

Figure 4: Initial SIFT poses on KITTI-01

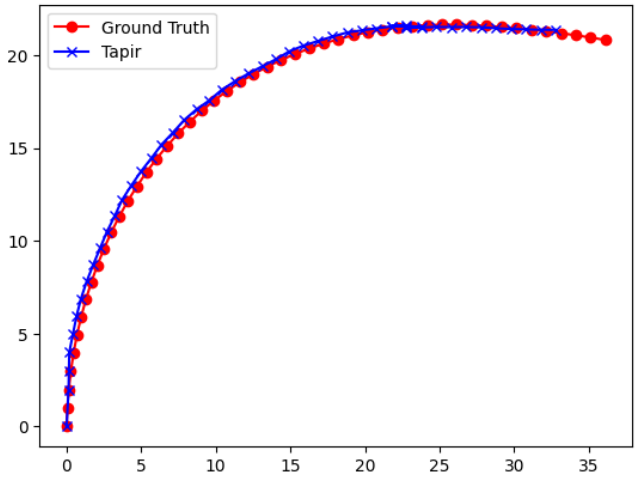

Figure 5: Initial TAPIR-2 poses on KITTI-01

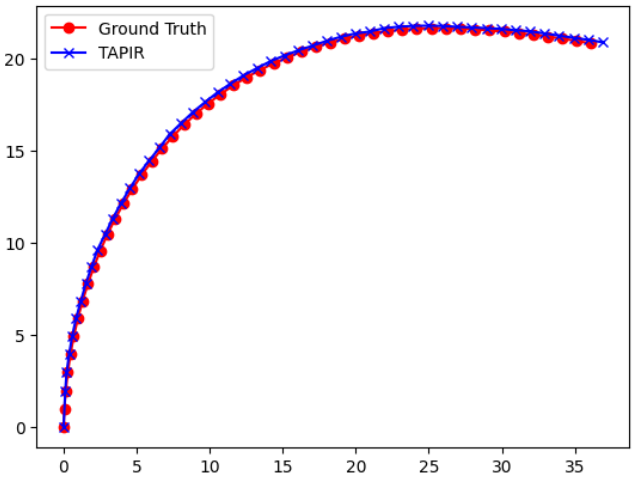

Figure 6: Initial TAPIR-50 poses on KITTI-01

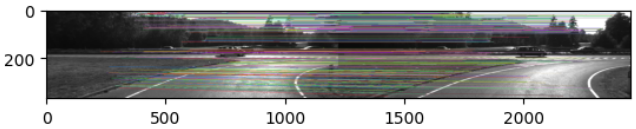

Figure 7: Sample image where 363 Points Matched with SIFT

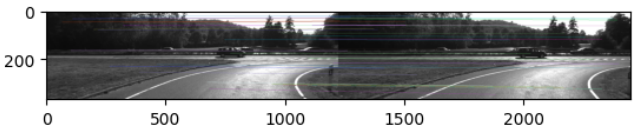

Figure 8: Sample image where 31 Points Matched with TAPIR-2

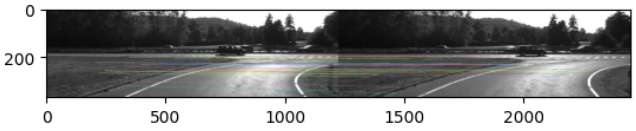

Figure 9: Sample image where 32 Points Matched with TAPIR-50

Table 4: Average Time to Process 50 Frames in Seconds (BA)

KITTI #	TAPIR-50 (BA)	TAPIR-2 (BA)	SIFT (BA)
01	3524.59	42.59	4033.67
02	3746.05	43.49	4138.35
03	3910.56	32.83	78564.08
04	1423.05	16.08	8219.58
05	4057.79	25.06	17654.47

Table 5: Average Translation Deviation from Ground Truth Path (unit vectors) (BA)

KITTI #	TAPIR-50 (BA)	TAPIR-2 (BA)	SIFT (BA)
01	14.05	14.09	1.22
02	13.13	15.28	6.27
03	3.67	2.87	2.13
04	4.28	4.46	4.26
05	6.68	6.50	6.46

Table 6: Average Rotation Deviation from Ground Truth Path (degrees) (BA)

KITTI #	TAPIR-50 (BA)	TAPIR-2 (BA)	SIFT (BA)
01	81.81	82.85	70.21
02	42.71	54.12	31.81
03	14.84	8.68	14.97
04	2.53	6.65	2.05
05	11.37	7.63	13.16

Table 7: SplattingTAPIR's Component Time Taken (Seconds)

TUM	TAPIR-50	Depth-Anything	SIM-Sync	Total
Freiberg Room	5.48	10.40	398.70	414.588
Freiberg XYZ	4.65	11.29	188.16	204.1

Table 8: SplattingTAPIR's Average Errors

TUM	Trans. Error (û)	Rot. Error (°)
Freiberg Room	0.01	2.44
Freiberg XYZ	0.2	2.25

pared with each other.

BA denotes bundle adjustment and the TAPIR models are the same as the ones found in the previous set of tests. For the sake of time, bundle adjustment stops after any minimization iteration that lasts longer than one hour. However, for SplattingTAPIR, the TAPIR-50 model is upgraded to BootsTAP [8], which is an upgraded TAPIR model with a teacher model built in for more efficient and accurate predictions. SplattingTAPIR's chunk size is also increased to 64 chunks, instead of 16 chunks, which should explain the speed up.

When looking at the results from tables 4, 5, and 6, the results are much worse than those found in tables 1, 2, and 3. Bundle adjustment increased time crossed the board exponentially, and the errors in both rotation and translation mostly increased dramatically. This signifies that bundle adjustment is converging at a false minimum, which is creating these uncertain solutions despite being labeled as such. This phenomenon can be seen with figures 10 and 11 where inaccurate initial poses can converge to a solution to have poses that are closer to the ground truth, while more accurate initial poses converge to a solution to one that is completely wrong.

Looking at the results produced from SplattingTAPIR, the average rotational and translation error is much less than those found in the initial SfM pipeline with and without bundle adjustment, showing how SplattingTAPIR can certifiably find the most optimal solutions. SplattingTAPIR's errors are much more consistent than those found with prior methods, but its total time including optimization is magnitudes faster than that with traditional bundle adjustment, highlighting how robust and efficient this pipeline is at extracting camera poses and point clouds.

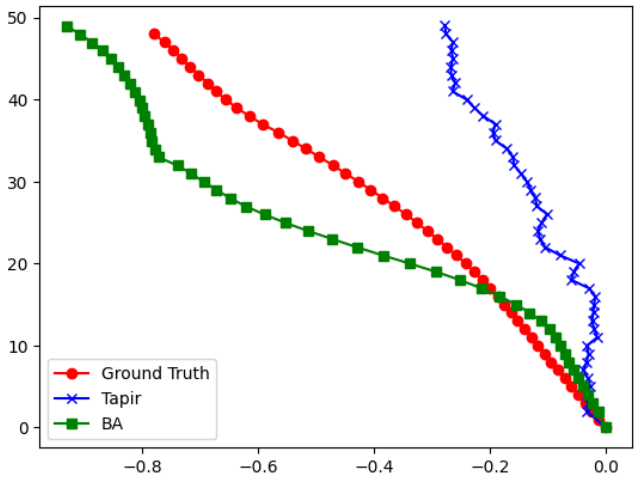

Figure 10: Partial success with classical bundle adjustment using PyCeres where initially wrong poses are getting more optimized

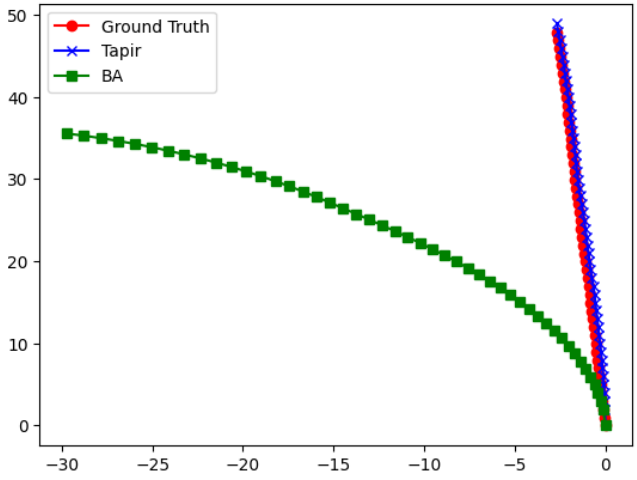

Figure 11: Failure with classical bundle adjustment using PyCeres where a good initial estimate of poses is getting more inaccurate with bundle adjustment.

Figure 12: Sample image generated using Mip-NeRF 360. The model took 42 hours to completely generate this scene.

Figure 13: Sample image using 3D Gaussian Splatting. The model took 30 minutes to completely generate this scene.

Comparing Radiance Fields

In this test, the performance between the state-of-the-art NeRF, Mip-NeRF 360, and 3D Gaussian Splatting is compared. Both models have their poses initialized with COLMAP[24] using a set of images that were captured inside the BLS Topol room. COLMAP is sued for NeRF compatibility. The views that are shown are completely generated by these radiance fields and are not found in the sequence of photos captured. Both models are running the unmodified version found in their respective papers.

Not only does the render produced by 3D Gaussian Splatting feature fewer artifacts, but it's also over 84 times faster than Mip-NeRF 360. 3D Gaussian Splatting proves to be the clear winner when it comes to efficiency as it can produce renders that are more visually appealing in a fraction of the time when compared to the latest radiance field methods. The artifacts found in both renders can be attributed to COLMAP's pose estimation. COLMAP was chosen instead of SplattingTAPIR for the test to ensure compatibility with Mip-NeRF 360. COLMAP's pose estimation yielded unfavorable results in certain parts of the scenes, making these radiance fields have a tougher time generating clouds for the scene, which can be seen with the artifacts.

Conclusions

SplattingTAPIR highlights the power that can be harnessed by recent developments in foundation models, and how this can transform the Structure from Motion pipeline. The SplattingTAPIR pipeline is a simple pipeline that leverages recent advancements in keypoint tracking, monocular depth estimation, pose recovery, and radiance fields that can be efficiently used to create immersive and hyper-realistic 3D scenes.

Future Directions

SplattingTAPIR's robust pipeline has so many potential uses. With the recent rise of Cryo-EM[6], a process where proteins are mapped in 3D, in the biotech industry, there is a push for using image reconstruction techniques to make these processes much faster. 3D offers so much more information than 2D, and has the potential to change how information can be perceived. Mapping proteins in 3D with Cryo-EM is much faster than older techniques and can be used to accelerate drug discovery. SpalttingTAPIR can also revolutionize storytelling, as the immersive experience that 3D Gaussian Splatting brings is unparalleled to other techniques in terms of realism and detail.

Acknowledgements

I would like to acknowledge Ms. Bowles and Mr. Mikalaitis from the BLS Senior Capstone class and Mr. Balicki and Ms. Bateman for their support of this project. They've played an invaluable role by providing direction and feedback along each step of the way. I would also like to acknowledge Professor Heng Yang, Haoyu Han, Wency Suo, and the Harvard Computational Robotics Lab for providing strong technical support with this project. I am honored to have collaborated with them on this project and to have been exposed to advanced topics in computer vision and machine learning during my internship at Harvard. Lastly, I would like to thank Jingnan Shi from MIT and Jon Barron from Google Research for the insight they have put into this project. This project was also supported with Cloud TPUs from Google's TPU Research Cloud (TRC).

References

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. *Ceres Solver*. Version 2.2. Oct. 2023. URL: https://github.com/ceres-solver/ceres-solver (cit. on pp. 1, 3).

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". In: *CVPR* (2022) (cit. on pp. 1, 4).

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. "Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields". In: *ICCV* (2023) (cit. on p. 1).

[4] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax (cit. on p. 2).

[5] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000) (cit. on p. 4).

[6] Marta Carroni and Helen R Saibil. "Cryo electron microscopy to determine the structure of macromolecular complexes". en. In: *Methods* 95 (Feb. 2016), pp. 78–85 (cit. on p. 8).

[7] Perceval Desforges. *Quadratic optimization with constraints in python using CVXOPT*. May 2022. URL: https://towardsdatascience.com/quadratic-optimization-with-constraints-in-python-using-cvxopt-fc924054a9fc (cit. on p. 3).

[8] Carl Doersch, Yi Yang, Dilara Gokay, Pauline Luc, Skanda Koppula, Ankush Gupta, Joseph Heyward, Ross Goroshin, João Carreira, and Andrew Zisserman. *BootsTAP: Bootstrapped Training for Tracking-Any-Point*. 2024. arXiv: 2402.00847 [cs.CV] (cit. on p. 6).

[9] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. "TAPIR: Tracking Any Point with per-frame Initialization and temporal Refinement". In: *ICCV* (2023) (cit. on pp. 1, 2).

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on p. 4).

[11] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, Benoit Maujean, Gregoire De Lillo, and Yann Lanthony. "AliceVision Meshroom: An open-source 3D reconstruction pipeline". In: *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*. ACM Press, 2021. DOI: 10.1145/3458305.3478443 (cit. on p. 1).

[12] Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. "Particle Video Revisited: Tracking Through Occlusions Using Point Trajectories". In: *ECCV*. 2022 (cit. on p. 2).

[13] Michael Kazhdan and Hugues Hoppe. "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graph.* 32.3 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2487228.2487237. URL: https://doi.org/10.1145/2487228.2487237 (cit. on p. 4).

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (July 2023). URL: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/ (cit. on pp. 1, 4).

[15] Jonas Kulhanek. *jkulhanek/nerfbaselines*. Feb. 15, 2024. URL: https://github.com/jkulhanek/nerfbaselines (cit. on p. 4).

[16] David G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (2004), 91â110. DOI: 10.1023/b:visi.0000029664.99615.94 (cit. on pp. 1, 2).

[17] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. 2020 (cit. on pp. 1, 4).

[18] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. "OpenMVG: Open multiple view geometry". In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, pp. 60–74 (cit. on p. 1).

[19] Tim Nguyen. *Lightweight Low-Poly Photogrammetry with Object Classification Validation with OpenCV and PointNet*. 2023. URL: https://thisistimnguyen.com/low-poly-recon.html (cit. on p. 4).

[20] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL] (cit. on p. 1).

[21] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV] (cit. on p. 3).

[22] Polycam Inc. *Polycam - LiDAR & 3D Scanner*. Version 2.2.18. Aug. 8, 2022. URL: https://apps.apple.com/us/app/polycam-lidar-3d-scanner/id1532482376?platform=ipad (cit. on p. 1).

[23] Paul-Edouard Sarlin, skydes, Philipp Lindenberger, Dmytro Mishkin, Joshua O'Reilly, Pablo Speciale, Tobias Fischer, and nnop. *cvg/pyceres*. Feb. 19, 2024. URL: https://github.com/cvg/pyceres (cit. on p. 5).

[24] Johannes Schönberger et al. *colmap/colmap*. Feb. 23, 2024. URL: https://github.com/colmap/colmap (cit. on pp. 1, 7).

[25] Aishwarya Singh. *SIFT: How to use SIFT for image matching in Python*. Dec. 2020. URL: https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python (cit. on p. 2).

[26] 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)*. Oct. 2012 (cit. on p. 5).

[27] Lieven Vandenberghe. *Nonlinear least squares*. https://www.seas.ucla.edu/~vandenbe/133A/lectures/nlls.pdf. 2017 (cit. on p. 3).

[28] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. "Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data". In: *CVPR*. 2024 (cit. on pp. 1, 3).

[29] Xihang Yu and Heng Yang. *SIM-Sync: From Certifiably Optimal Synchronization over the 3D Similarity Group to Scene Reconstruction with Learned Depth*. 2023. arXiv: 2309.05184 [cs.RO] (cit. on pp. 1, 3).