AVP-SLAM++: Optimizing Visual-Only SLAM Performance Using Odometry Transformations in GraphSLAM

Vivek Jaiswal[1], Harsh Jhaveri [2], Chun-Hseuh Lee [3], and Devin McCulley [4]

***Abstract*— Traditional visual-odometry-based methods are popular and well-studied algorithms for SLAM. However, given their reliance on geometry-based features in a scene, the performance of these methods is limited in generalized environments. Semantic SLAM methods, like AVP-SLAM, making use of higher order features such as traffic signs and plants, are more robust in environments where conventional visual-odometry algorithms may fail. However, even the performance of these methods is limited based on the characteristics of their training datasets. Thus, we adapt an open-source implementation of AVP-SLAM, AVP-SLAM-PLUS, to utilize an offline deep learning-free method, GraphSLAM, in both batch and incremental solvers to optimize localization. In our work, we correct the simulation inconsistencies of AVP-SLAM-PLUS, implement noisy odometry transformations, and develop a novel approach to loop closure utilizing Euclidean distance. We demonstrate that our method produces an overall improvement in localization accuracy, underscoring its practical benefit for modern day systems. Our open source adaptation of AVP-SLAM-PLUS, AVP-SLAM++, can be found here.**

I. INTRODUCTION

A. Limitations of Traditional Visual-Odometry SLAM Methods

In autonomous driving applications, vehicles often operate in environments with many repeated features, appearance and illumination changes, and texture-less regions, ultimately making perception difficult. For parking lots and structures, these perception problems are exacerbated as these environments are often GPS-denied or GPS-limited, further limiting localization performance and robustness [1] Consequently, traditional visual-odometry SLAM performance is limited, potentially creating grave safety concerns for crowded and constrained urban environments.

B. Introduction to Semantic SLAM Methods and AVP-SLAM

Given the limitations of traditional visual-odometry SLAM methods, semantic SLAM methods utilize semantic segmentation on scenes to conduct point cloud registration. Compared to geometrical features used in conventional scene processing, semantic segmentation utilizes higher order features such as trees, traffic signs, road lines, etc. [2]. This form of feature detection results in point clouds that are more stable and robust, regardless of the repeated or ill-illuminated nature of an environment.

However, to fully benefit from the strengths of semantic segmentation, a deep learning model needs to be trained, which can potentially be a time consuming and error-prone process. Additionally, without a properly trained model, performance of point cloud registration, and thus the SLAM algorithm, is fundamentally limited.

AVP-SLAM, first published at IROS 2020, utilizes semantic features for point cloud registration and loop closure [1]. Additionally, IMU data and wheel encoders are utilized to produce a prior which is then combined with semantic features to produce a global visual semantic map. An example of this map is given in Figure 1. For an autonomous valet parking application in a controlled and relatively static environment, this method produces centimeter-level localization accuracy. A diagram of the published algorithm is given in Figure 2

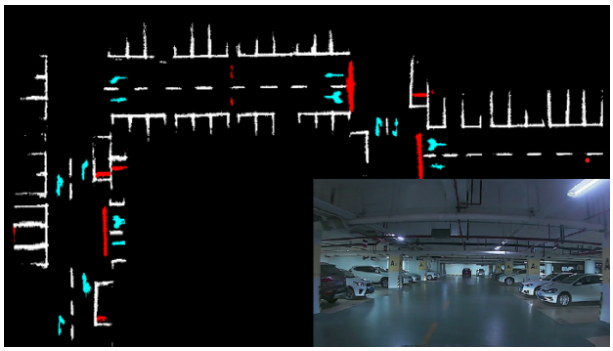


Fig. 1. Published example of the global visual semantic map produced by AVP-SLAM

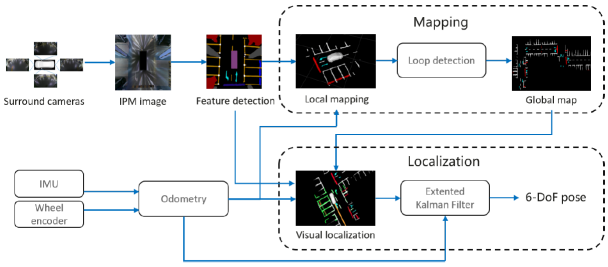


Fig. 2. Block diagram illustrating the nature of the published AVP-SLAM algorithm. The system inputs data from four surround RGB cameras before stitching them together and processing the scene. This information is then combined with odometry data to produce a semantic map of the entire environment and localize the vehicle within it.

C. Overview of AVP-SLAM-PLUS

When it was published, the authors of AVP-SLAM did not produce and make an official open-source implementation of their algorithm available. AVP-SLAM-PLUS is an

[1]MS Robotics, University of Michigan, vjaiswal@umich.edu
[2]MS Robotics, University of Michigan, hjhaveri@umich.edu
[3]MS Robotics, University of Michigan, chunhlee@umich.edu
[4]MS Robotics, University of Michigan, devmccu@umich.edu

unofficial open source implementation of AVP-SLAM, which works off of a simulated environment in Gazebo which is further visualized in RVIZ [3]. In addition to the base RGB implementation of AVP-SLAM, AVP-SLAM-PLUS can also utilize simulated RGB-D data if desired.

Intended to be a demonstrative example, AVP-SLAM-PLUS does not utilize deep learning based methods in order to semantically segment the environment. Given the ideality, features are pre-labeled based on color, and point cloud registration is executed accordingly.

Additionally, the algorithm does not use odometry information to produce a prior transformation between poses. AVP-SLAM-PLUS directly utilizes either iterative closest point (ICP) or normal distribution transformations (NDT). Each iteration of AVP-SLAM-PLUS utilizes the previously produced SLAM pose as a prior, and utilizes either ICP or NDT to solve for the current pose. Given the repetitive and ideal nature of features in the simulation environment, localization performance varies. Additionally, as developed, there are inconsistencies between the RVIZ and Gazebo environments, which does not allow for proper comparison between the SLAM pose and ground truth trajectory. However, once this is corrected for, the resulting optimized trajectory generally is not smooth and is "scaled down" compared to ground truth trajectory.

D. Related Works

As stated, semantic segmentation is a difficult problem to solve, with a variety of technical challenges. Most notably, properly labeled and diverse data sets are required for reliable model training and identification of features in unknown environments. As of February 2022, most available labeled datasets are created for urban autonomous vehicle domains, leaving many domains still yet to be explored [4]. Thus, the generalizability of algorithms such as AVP-SLAM to non-urban or infrastructure-less environments is starkly limited. Furthermore, while urban datasets exists, they are mostly two-dimensional. Additionally, there is scientific-interest to work with depth-based semantic segmentation, but there is not sufficient standardized training data available. In our case, AVP-SLAM-PLUS is able to produce point clouds from RGB-D cameras. However, without proper training data, semantic segmentation cannot be expanded to make use of this information.

Previous work has been done to produce robust localization results utilizing graphical methods. Mostly notably, ICP has been combined with pose graph optimization in order to produce accurate localization and loop closure over a trajectory [5]. Pose graph SLAM optimization, more commonly known as GraphSLAM, is a method first published in 2006 which combines SLAM posteriors and transformations into a graphical-model and optimizes the overall trajectory using least squares solvers offline [6]. Since then, in 2011, an incremental solver, iSAM2, was first published, enabling incremental smoothing and mapping, rather than just in batch upon completion of a trajectory or in periodic fashion [7]. This allows continuous optimization which can be done more "online," compared to conventional GraphSLAM.

While [5] utilizes point clouds derived from keyframes on LiDAR data, use of GraphSLAM to optimize a trajectory is promising. Additionally, this method employs batch graph processing, rather than iSAM2. In our work, we aim to utilize a similar batch method, but utilizing point clouds derived from our simulated semantic segmentation data.

E. Statement of Problem and Motivation

Given the limitations of AVP-SLAM-PLUS, a fully featured and accurate open source implementation of AVP-SLAM does not exist. However, if implemented as published, a deep learning vision model would have to be developed and trained, which requires a proper training set to work in a robust manner. Given the demonstrative and ultimately educational nature of AVP-SLAM-PLUS, use of deep learning methods would diverge from the original intent.

Thus, we set out to produce a robust simulation environment and implementation of AVP-SLAM that adheres more closely to the published algorithm by augmenting the AVP-SLAM-PLUS implementation. However, we aim to do this without using deep learning based methods, opting to keeping the naive semantic feature detection as is. Instead of stronger feature detection, we are motivated by the potential of utilizing GraphSLAM to optimize the trajectory globally, rather than utilizing local solvers such as ICP.

This work details the steps taken to achieve this end, including changes to the AVP-SLAM-PLUS simulation environment, addition of odometry transformations, and the implementation of a novel approach to loop closure.

II. SYSTEM OVERVIEW

A. Computing Environment

Given the various local development environments used by members of our team, we chose Docker as a unifying development platform. We first pulled the base image of Ubuntu 18.04 and required all necessary dependencies. Source code was kept locally and run in Docker by attaching it as a volume. This configuration proved to be quite useful as source code could be edited locally while still being compiled and run in the Docker container itself.

B. Simulation Environment

We simulated our environment in Gazebo as shown in Figure 3. Our environment has two roads configured in a L-Shape. Features in this environment are yellow parking spaces, yellow traffic arrows, and white dashed lines. At the start of the simulation, the robot model is initialized at the origin, defined as the junction of the two tracks. ROS was chosen as the middleware platform as it provides reliable and well documented integration with Gazebo and visualization tools such as RVIZ.

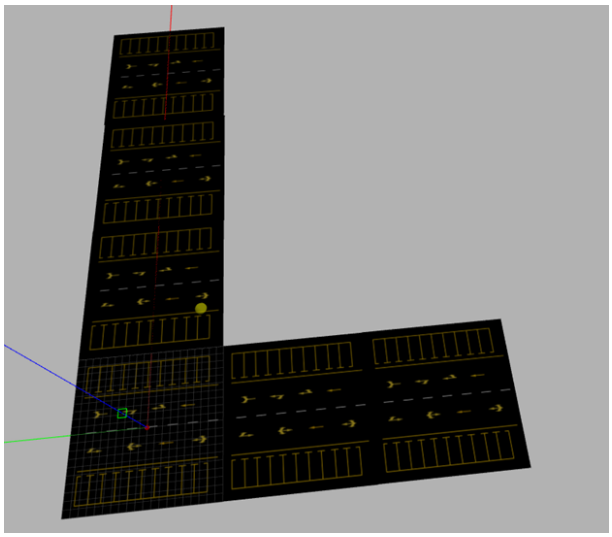


Fig. 3. Simulation Environment in Gazebo

C. Robot Model

The chosen robot model simulates a two wheel robot with a caster wheel. AVP-SLAM was published using data from a four-wheeled vehicle. While our model is not identical to this, both models have non-holonomic constraints, and thus, a two-wheeled model is sufficient enough to emulate car-like dynamics. Additionally, the robot is modeled with five cameras mounted 0.5m above the base of the vehicle. These cameras are arranged with minimally overlapping sensing cones in order to produce a proper surround view. Additionally, the model can be configured to utilize these cameras as either RGB or RGB-D.

III. METHODOLOGY

A. Existing AVP-SLAM-PLUS Framework and Limitations

As described in the introduction, AVP-SLAM-PLUS is a simplified implementation of the full AVP-SLAM algorithm intended for educational purposes. The localization and mapping code is written in C++, and the robot control code is written in Python2. Mapping is done by adding the point clouds formed by the current features to the global feature point cloud after transforming the current points to be same frame as the robot's current pose. The mapping code then publishes the updated global feature point cloud. Separate code conducts point cloud registration to generate the local features point cloud from the set of either RGB or RGB-D cameras. For RGB cameras, the extrinsic matrix is used to transform points in the camera frame to points on the XY plane. The RGB-D camera simply rotates all point clouds to camera0's frame and builds a scene on the XZ plane. In both cases, mapping and point cloud registration, points too far or too close to the robot and points that are the color of the simulated ground (i.e. black points) are ignored. In the RGB case, points that are the color of the sky are also ignored. These threshold distances and the sky color are all tunable parameters in the configuration file to allow for proper filtering.

In localization, the current pose is found by comparing the local feature point cloud with the global feature point cloud. Either ICP or NDT is used to find a transformation between the two point clouds, and this transformation is then applied to the previous pose to find the current pose with respect to the global map. There is a convenient launch file to do SLAM that launches both mapping and localization codes or just localization only. Point clouds are represented as objects of the point cloud library and the global feature point cloud is saved in .pcd format.

Additionally, the implemented robot controller allows the user to manually drive the robot in the simulated world. As implemented by AVP-SLAM-PLUS, the robot's motion was deterministic and there was no noise added into the robot motion model. In order to make the environment more realistic, we produced a noisy motion model and utilized odometry from the robot to correct for sensor noise.

B. Odometry Transformations

The robot controller provides a translational (v) and rotational (w) velocity for the differential wheeled robot. The control commands were taken to model the motion noise, where c_1, c_2, c_3, c_4 are hand tuned parameters [8].

$$R_t = \begin{bmatrix} c_1 v^2 + c_2 \omega^2 & 0 \\ 0 & c_3 v^2 + c_4 \omega^2 \end{bmatrix} \tag{1}$$

The robot state was defined as $X = [x \ y \ \theta]'$, and the Jacobian matrix of control input with respect to the robot state will be:

$$\begin{aligned} V_t &= \frac{\partial g\left(u_t, \mu_{t-1}\right)}{\partial u_t} \\ &= \begin{pmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial \omega_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial \omega_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta'}{\partial \omega_t} \end{pmatrix} \\ &= \begin{pmatrix} \frac{-\sin\theta + \sin(\theta + \omega_t \Delta t)}{\omega_t} & \frac{\partial x'}{\partial \omega_t} \\ \frac{\cos\theta - \cos(\theta + \omega_t \Delta t)}{\omega_t} & \frac{\partial y'}{\partial \omega_t} \\ 0 & \Delta t \end{pmatrix} \end{aligned} \tag{2}$$

$$\frac{\partial \theta'}{\partial \omega_t} = \frac{v_t\left(\sin\theta - \sin\left(\theta + \omega_t \Delta t\right)\right)}{\omega_t^2} + \frac{v_t \cos\left(\theta + \omega_t \Delta t\right) \Delta t}{\omega_t} \tag{3}$$

$$\frac{\partial y'}{\partial \omega_t} = -\frac{v_t\left(\cos\theta - \cos\left(\theta + \omega_t \Delta t\right)\right)}{\omega_t^2} + \frac{v_t \sin\left(\theta + \omega_t \Delta t\right) \Delta t}{\omega_t} \tag{4}$$

The covariance matrix of odometry Σ_t can be obtained by applying affine transformation V_t to R_t.

$$\Sigma_t = V_t R_t {V_t}^T \tag{5}$$

The command used for robot simulation will be sampled from Gaussian distribution centered at $[v\ \omega]'$ with covariance matrix R_t. [8]

$$u_t = \begin{bmatrix} v_{actual} \\ \omega_{actual} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} v \\ \omega \end{bmatrix}, R_t) \tag{6}$$

The kinematic model of differential wheeled robot is defined as the following:

$$x_{t+1} = x_t + v_{actual}\ \Delta t\ cos(\theta + \omega_{actual}\Delta t) \tag{7}$$

$$y_{t+1} = y_t + v_{actual}\ \Delta t\ sin(\theta + \omega_{actual}\Delta t) \tag{8}$$

$$\theta_{t+1} = \theta_t + \omega_{actual}\Delta t \tag{9}$$

C. Waypoint Following Controller

To automatically drive the robot on the desired path. A waypoint following controller is used. Control commands are calculated based on the distance and orientation difference between the robot's current pose and the assigned waypoints. Equations are showed below in 10 - 12. and the geometry of a mobile robot is shown in Figure 4 [9].

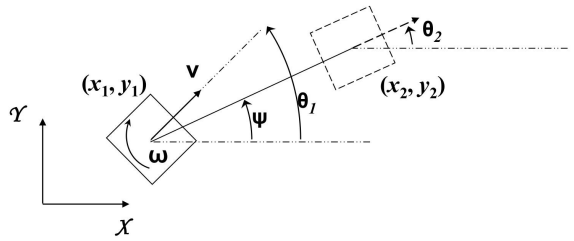


Fig. 4. Inverse kinematics of differential wheeled mobile robots. x_1 and y_1 are the current coordinate of the robot, x_2 and y_2 are the coordinate of the robot at the next time step. θ_1 and θ_2 are yaw angles of the robot at current and next time step.

$$\mathrm{v} = k_\mathrm{v} \cos(\psi - \theta_1) \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{10}$$

$$\omega = k_\omega (\psi - \theta_1) \tag{11}$$

where

$$\psi = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{12}$$

D. Loop Closure Detection

Due to the fact that the odometry drifts as traveled distance increases, loop closure must be used to correct the accumulated error. The first attempted method for loop closure used local feature point clouds recorded to a rosbag and ICP. This method used odometry for both the vertexes and adjacent edges in the pose graph. The saved point clouds were assigned indexes equal to the latest odometry vertex. After rotating the point clouds by the robot's heading so that they aligned with the world frame, a brute force approach was used to find matching point clouds. If the ICP fitness score was lower than a tuned threshold, a loop was detected and another edge was added to the factor graph. This method resulted in varying results, and was not utilized overall.

On second attempt, AVP-SLAM-PLUS poses were used for more accurate loop closure detection. The team observed that although the pose estimation of AVP-SLAM-PLUS has a scaling error. The relative position of each pose is correct as it compares the current point cloud with the global map, and computes a transformation with either ICP or NDT. The team used the AVP-SLAM-PLUS estimation for loop detection. A loop is be detected if the estimated poses at times i and time j, which are P_i and P_j, are detected to be closer together than a user defined parameter ϵ. In both methods, when a loop closure is detected, an edge with 0 translation and appropriate rotation is added to the factor graph between the vertexes at i and j. Given the nature of the solver, the covariance of the transformation cannot be found deterministically, and thus, this was also chosen to be a tunable parameter.

$$||P_i - P_j|| < \epsilon \tag{13}$$

E. Factor Graph Optimization

For graph optimization, open source factor graph library GTSAM is utilized in this project. In the factor graph, edges are built by odometry. The transformation between poses at times i and time j can be obtained as follow:

$$T_i^j = \left(T_0^i\right)^{-1} T_0^j \tag{14}$$

Where T_0^i and T_0^j is the odometry SE2 poses, and the covariance matrix of odometry is used as covariance in edges.

The loop closure edges will be built once loops are detected. The transformation will be formed solely by rotation by $\theta_i - \theta_j$, as there is not translation. The covariance matrix of loop closure edges is a hand tuned covariance matrix with diagonal values smaller than one and non-diagonals values set to 0.

The SLAM poses from AVP-SLAM-PLUS will be used as initial values for vertices in factor graphs. Both batched optimization using Gauss-Newton optimizer and iSAM (Incremental Smoothing and Mapping) is implemented in this project.

F. Calculation of Error and Standard of Comparison

For the standard of comparison, the RMSE (Root Mean Square Error) was calculated for odometry, the SLAM pose produced by ordinary AVP-SLAM-PLUS, and our proposed method with respect to the ground truth obtained in Gazebo. The equation of RMSE is shown below:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}} \tag{15}$$

IV. EXPERIMENTAL RESULTS

The point cloud orientation for RGB camera mode from AVP-SLAM-PLUS had the y-axis inverted when mapping. This is because when points in the camera's frame are translated to the robot's frame, it keeps the y-axis from the

bottom of the image to the top which is opposite the y-axis used in right-hand rule. Inverting all pixel y-values in the current feature point cloud in pointCloudFromRGB.cpp made the map more accurate. The point cloud orientation for RGB-D camera mode used camera 0's frame so the map was built in the XZ plane as seen in Figure 5. The RGB-D camera mode was fixed by inverting pixel y-values and rotating the current feature point cloud from camera 0's frame to the robot's frame in pointCloudFromRGB-D.cpp.

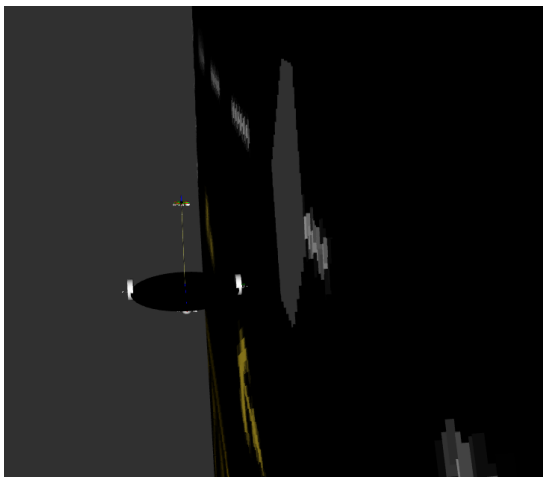


Fig. 5. Original map orientation for RGB-D camera mode

Loop closure detection using ICP globally was not successful and was never fully tested on all data sets. When debugging we found that loop closures were either detected in the wrong poses or not at all depending on the parameter tuning. In the case of too many loops detected, we visualized matching point clouds and noticed that they would mostly match, with only slight differences that were often hard for human eyes to detect. A trial where 199 loop closures were detected is shown in Figure 6 For these reasons, the AVP-SLAM-PLUS based loop closure detection described in the methodology section was the chosen method for AVP-SLAM++.

Error compared to Ground Truth	
SLAM	3.739
Odometry	3.736
Batch Opt	3.372
ISAM Opt	3.372

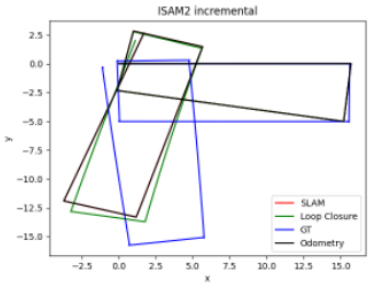
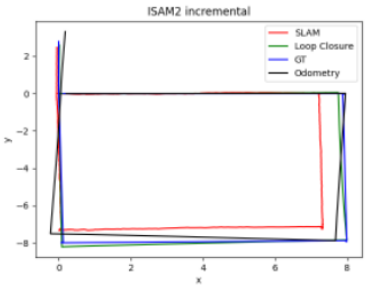


Fig. 6. Resulting trajectories and RMSE compared to ground truth. SLAM and Odometry are both poses from dead reckoning of odometry but SLAM has loop closure edges detected from ICP. Loop Closure is the optimized trajectory. GT is the ground truth.

Multiple different optimized trajectories are shown in Figure 7 along with RMSE. The square path was designed to be a simple test case with a single guaranteed loop closure. The SLAM path in the square trial performed the worst because the scale of the mapping was wrong. Our optimized path performed better than odometry because it kept the same shape as SLAM but corrected for scale. The L shape path was intended to be a challenging test case with many loop closures. This long path shows how large odometry error can grow after an extended time. The loop closures relying on AVP-SLAM-PLUS poses were able to correct for this drift and the optimized paths had the lowest RMSE value. Lastly, there is the M shape path that does not have any loop closures. This test case shows that even without loop closures, the pose graph optimization can return a better path than AVP-SLAM-PLUS by correcting the map scaling and non-smoothness problem. A video demonstration of our results can be found here.

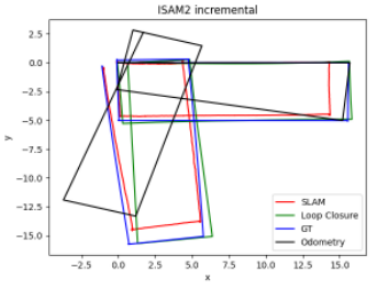
i. Square with overlap

Error compared to Ground Truth	
SLAM	0.655
Odometry	0.392
Batch Opt	0.139
ISAM Opt	0.139



ii. L shape

Error compared to Ground Truth	
SLAM	0.864
Odometry	3.780
Batch Opt	0.659
ISAM Opt	0.658



iii. M shape

Error compared to Ground Truth	
SLAM	0.528
Odometry	0.348
Batch Opt	0.481
ISAM Opt	0.481

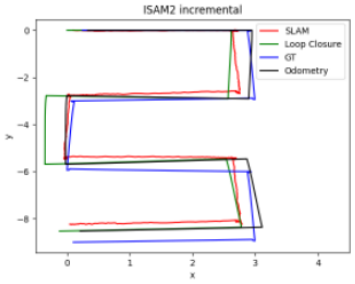


Fig. 7. Resulting trajectories and RMSE compared to ground truth. SLAM is the AVP-SLAM-PLUS poses. Loop Closure is the optimized trajectory using AVP-SLAM loop closures. GT is the ground truth. Odometry is poses from dead reckoning of odometry.

V. CONCLUSION

A. Discussion

While adapting the AVP-SLAM-PLUS repository and using it to record data trials, we created a docker image and fixed the orientation of the map point clouds. We also made the simulation more realistic by adding noise to the motion

model and odometry, which can be seen in the odometry trajectories from our experimental results. Loop closure and pose graph optimization were implemented and shown to improve RMS error and produce smoother trajectories. Loop closure detection using AVP-SLAM-PLUS poses worked better than just using point clouds with ICP. Both batch and incremental pose graph optimization was used although, we did not notice a difference in the results produced by both methods. These changes along with supporting documentation have improved the open source AVP-SLAM-PLUS code base. Our open source adaptation of AVP-SLAM-PLUS, AVP-SLAM++, can be found here.

B. Limitations of Results

The results where more improvement is still possible are in the correction of the RGB camera mode point cloud orientation. The camera extrinsic matrix used to get coordinate values from pixel values likely has a rounding error that leads to a map scaled down by about 9%. The effect of this scaling can be seen in Figure 8 where RGB and RGB-D point clouds are compared. Secondly, the ICP loop closure method did not perform well, and through multiple code iterations, we could not detect accurate loop closures using the local feature point clouds from each AVP-SLAM-PLUS pose. This is likely due to the repetitive nature of our simulated environment and the small distinctions in local features for each pose.

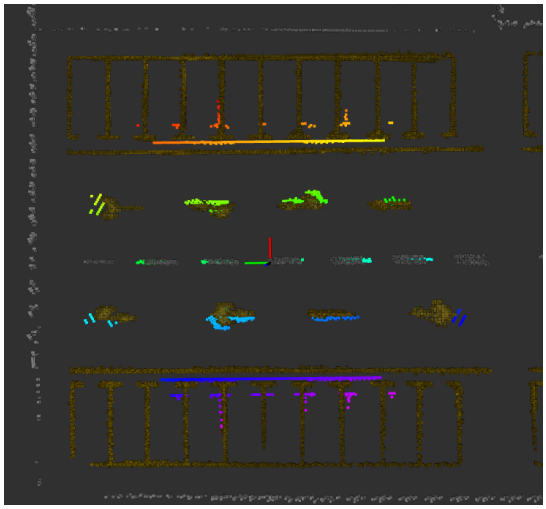


Fig. 8. RGB-D map (rainbow colored) and RGB map. The scaling issue present in the RGB map can be clearly seen. Given the extended dimensionality of the RGB-D point clouds, the output is more aligned with the ground truth map.

C. Future Work

The improvements made in AVP-SLAM++ can be extended by bringing our code changes to an online system running in a real environment. The scaling issue from RGB camera mode would be addressed while implementing feature segmentation on actual images. A deep learning model trained on an image dataset along with multi-camera triangulation could lead to realistic real world maps. A vehicle with onboard odometry could do live loop closure detection and run an incremental iSAM2 pose graph optimization to bring AVP-SLAM++ properly online.

ACKNOWLEDGMENTS

Firstly, we want to thank the ROB 530 instructional team, Maani, Justin, and Jingyu, for their support and encouragement throughout the entire process. Additionally, we'd like to thank the original AVP-SLAM team, Tong Qin, Tongqing Chen, Yilun Chen, and Qing Su. Lastly, we'd like to acknowledge the developer of AVP-SLAM-PLUS, Liu Guitao. The original implementation of AVP-SLAM-PLUS can be found here.

REFERENCES

[1] T. Qin, T. Chen, Y. Chen, and Q. Su, "Avp-slam: Semantic visual mapping and localization for autonomous vehicles in the parking lot," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5939–5945.

[2] M. Thoma, "A survey of semantic segmentation," *CoRR*, vol. abs/1602.06541, 2016. [Online]. Available: http://arxiv.org/abs/1602.06541

[3] liuguitao, "Avp-slam-plus," January 2022, [Online; posted 11-January-2022]. [Online]. Available: https://github.com/liuguitao/AVP-SLAM-PLUS

[4] I. Ulku and E. Akagündüz, "A survey on deep learning-based architectures for semantic segmentation on 2d images," *Applied Artificial Intelligence*, vol. 0, no. 0, pp. 1–45, 2022. [Online]. Available: https://doi.org/10.1080/08839514.2022.2032924

[5] E. Mendes, P. Koch, and S. Lacroix, "Icp-based pose-graph slam," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2016, pp. 195–200.

[6] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006. [Online]. Available: https://doi.org/10.1177/0278364906065387

[7] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3281–3288.

[8] S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics," pp. 132–215, 2005.

[9] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. The MIT Press, 2011.