GRAPH BASED SLAM FOR AUTONOMOUS DRONE RACING

BY

ROBERT AZARCON

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Computer Engineering
in the Undergraduate College of the
University of Illinois at Urbana-Champaign, 2025

Urbana, Illinois

Adviser:

Professor Sayan Mitra

ABSTRACT

This research proposes and evaluates a method for performing SLAM (Simultaneous Localization and Mapping) using a factor graph optimization framework in a drone racing scenario. It is expected for the drone to localize both itself and the positions of all the gates within the drone racing course, given that it is equipped only with a camera and an IMU sensor. A convolutional neural network is used to extract corner points from gates visible to the drone's camera, and a separate tracking algorithm is used to track the corners as the drone moves. A separate, tightly coupled VIO module is also used in parallel to generate odometry. Finally, gate corner detections and odometry are paired over time in a factor graph. The method is demonstrated in a simple custom simulation environment inside Unreal Engine 5.5 and is tested in an offline scenario.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Simultaneous Localization and Mapping

1.1.1 Overview

For a robot to operate safely and efficiently in an unknown environment, it needs to create an understanding of its surroundings and its location. This problem is known as Simultaneous Localization and Mapping (SLAM), and solving it directly supports the robot's ability to plan, control, and make decisions. A robot performs SLAM by processing sensor measurements and using them to construct a map of its immediate environment, while also locating itself within the map. The map is an abstract concept representing some structure within the robot's environment. A map can take many forms, such as a list of landmark locations, an occupancy grid, a dense point cloud, and surfels. The choice of map representation depends heavily on the specific application, hardware resources, and the required level of detail and accuracy.

1.1.2 Applications for Drone Racing

In variations of autonomous drone racing where the race track is initially unknown or partially known, SLAM is a powerful method allowing a drone to estimate its state while discovering new gates during a race. Additionally, drone data can be saved and analyzed by a SLAM algorithm running in an offline mode setting, which can help gain close to ground truth information of a racetrack for testing purposes. However, it's important to note that if a precise map of the racetrack is available beforehand, like in some works such as [2], the SLAM problem reduces to localization within that known map, reducing the need for the mapping component of SLAM. Furthermore, drone racing methods exist that don't perform explicit state estimation altogether, such as purely vision-based, end-to-end approaches where control commands are learned and generated directly from raw pixel inputs [3]. Nevertheless, explicit state estimation via SLAM remains a viable framework for scenarios involving unknown racetrack information to support adaptive navigation within a drone race.

1.1.3 Challenges

Implementing SLAM for autonomous drone racing brings significant challenges that can push the boundaries of state-of-the-art approaches. Aggressive maneuvers such as rapid rotations and accelerations can induce motion blur in cameras and corrupt IMU measurements. Variations in race environments can also pose difficulties, such as changes in lighting, a lack of distinct visual features, or gates that are harder to detect. Furthermore, drones operate under strict size, weight, and power constraints, which limit the availability of onboard computational resources and sensors. As a result, SLAM systems need to be robust enough to detect noise and environmental variations while being responsive and computationally efficient enough to support real-time planning and control.

1.2 Sensor Modalities

1.2.1 Stereo Camera

Visual sensors are widely utilized in nearly all robotics applications because they provide fast, detailed information about an environment's appearance at relatively low cost and weight. A single-lens camera system is often called monocular and only captures a single image at a time, while a system with two or more lenses is called stereo. A stereo camera system produces multiple images simultaneously, which can be used to extract depth information. Assuming a standard pinhole model and rectified two camera stereo setup is used, if a point $\boldsymbol{P} = [X, Y, Z]^T$ in the camera frame projects to pixel coordinates (u_L, v_L) in the left image and (u_R, v_R) in the right image, the depth $\boldsymbol{Z}$ can be calculated using the disparity $\boldsymbol{d} = u_L - u_R$:

$$\boldsymbol{Z} = \frac{f \cdot b}{d} = \frac{f \cdot b}{u_L - u_R} \tag{1.1}$$

where f is the camera's focal length in pixels and b is the fixed baseline offset between the two cameras. In practice, the accuracy of the produced depth image decreases with range. Additionally, the variations in the environment can introduce errors during the stereo matching step, making it harder to find (u_R, v_R) that correspond to the same observed 3D point in space.

1.2.2 Inertial Measurement Unit

Inertial Measurement Units (IMUs) combine accelerometers, gyroscopes, and sometimes magnetometers to measure a robot's acceleration, angular velocity, and orientation in 3D space. Accelerometers measure the force of gravity and the force resulting from the sensor's acceleration relative to an inertial frame. Acceleration can then be derived after measuring force using Newton's second law. An acceleration measurement $\hat{\boldsymbol{a}} \in \mathbb{R}^3$ measured in $\mathrm{m\,s^{-2}}$ is modeled as:

$$\hat{\boldsymbol{a}} = R^T(\boldsymbol{a} - \boldsymbol{g}) + \boldsymbol{b}_a + \boldsymbol{n}_a \tag{1.2}$$

where $\boldsymbol{a}$ is the true acceleration we want to recover, R is the sensor orientation in the world frame, $\boldsymbol{b}_a$ is time varying bias, and $\boldsymbol{n}_a \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ is Gaussian white noise.

Gyroscopes measure the angular velocity of the IMU with respect to the sensor frame. An angular velocity measurement $\hat{\boldsymbol{\omega}} \in \mathbb{R}^3$ is modeled as:

$$\hat{\boldsymbol{\omega}} = \omega + \boldsymbol{b}_g + \boldsymbol{n}_g \tag{1.3}$$

Like the accelerometer measurement model, $\boldsymbol{b}_g$ represents time-varying bias and $\boldsymbol{n}_g$ represents Gaussian white noise. In practice, both accelerometers and gyroscopes produce spiky measurements which can be helped managed by statistical filtering techniques such as the Kalman Filter [4]. Furthermore, it is essential to note that there is an inherent error in obtaining orientation and position when integrating linear acceleration and angular velocity. This leads to significant drift over time, and as a result, they are almost always used in combination with other sensors.

1.3 Factor Graphs for SLAM

1.3.1 Overview of Factor Graphs

Since robot measurements are inherently probabilistic, Probabilistic Graphical Models (PGMs) [5] serve as a useful tool for representing estimation problems such as SLAM. PGMs, simply put, are graphs in which nodes represent random variables and edges represent conditional dependencies between the variables. The overall structure of a PGM then defines a joint probability distribution among the variables within the graph.

A factor graph is a specific type of probabilistic graphical model (PGM). It is a bipartite graph that contains two classes of nodes: variable nodes, which represent unknown values we wish to estimate, and factor nodes, which represent probabilistic constraints between variable nodes. Edges within the factor graph only connect factor nodes and variable nodes, which is why factor graphs are bipartite. Additionally, an edge only exists between a factor node and one or more variable nodes if the factor depends on the variable nodes to which it is connected. The key idea behind factor graphs is that they encode the posterior distribution $P(X|Z)$, where Z are measurements, as a product of factor functions f_i:

$$F(X) = \prod_{i=a}^{b} f(i) \tag{1.4}$$

Maximum A Posteriori (MAP) inference is performed by maximizing $F(X)$.

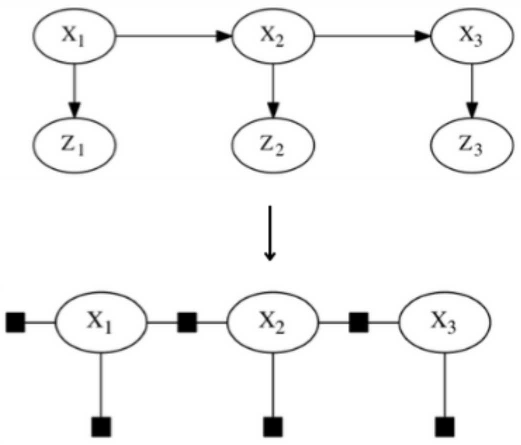

Figure 1.1: Example of a Bayes net and its corresponding factor graph representation. (by Frank Delleart [1])

1.3.2 Landmark-Based SLAM

A factor graph can formulate a landmark-based SLAM problem where the random variables include robot poses at various timesteps: $\boldsymbol{X} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in$ SE(3) and stationary observed landmark positions $\boldsymbol{L} = (\boldsymbol{l_1}, \dots, \boldsymbol{l_n}) \in \mathbb{R}^3$. Motion model factors connect temporally adjacent pose variables $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t)$, which could be derived from IMU measurements. If a camera is used to observe landmark locations, a visual projection and/or ranging factor can constrain the robot's pose and the position of any seen landmark (s). Then, the most optimal set of states $\boldsymbol{\Theta}^* = (\boldsymbol{X}, \boldsymbol{L})$ can be found by solving:

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \prod_i f_i(\boldsymbol{\Theta}_i, Z_i) \tag{1.5}$$

where Z represents the robot sensor measurements.

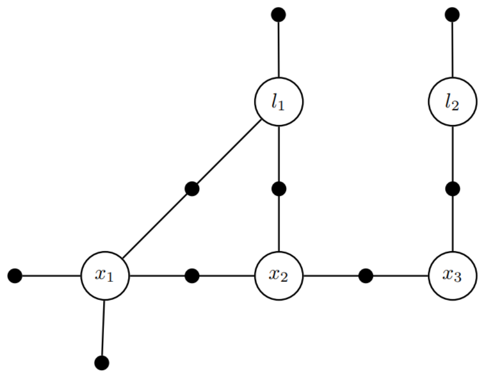

Figure 1.2: Example of Landmark-Based SLAM as a factor graph. (by Frank Delleart [1])

The factor nodes in figure 1.2 that are connected to only one variable node represent prior factors. Prior factors encode prior knowledge about certain variables. For example, if it is known that a landmark is located in a particular region within the world frame with some known covariance, a prior factor can be connected to its corresponding variable to constrain it to that particular region.

1.3.3 MAP Inference as Least Squares

Solving for the MAP estimate $\boldsymbol{\Theta}^*$ is equivalent to minimizing the sum of squared errors derived from the factor functions. Suppose we have a factor function defined as follows:

$$f_{i,k}(\boldsymbol{x}_i, \boldsymbol{l}_k) = P(\boldsymbol{z}_{i,k}|\boldsymbol{x}_i, \boldsymbol{l}_k) \propto \exp\left(-\frac{1}{2}\left\|h(\boldsymbol{\Theta}_i) - \boldsymbol{z}_{i,k}\right\|^2_{\Sigma_{i,k}}\right) \tag{1.6}$$

where h is a function that predicts what the measurement $\boldsymbol{z}_i$ with gaussian noise $\Sigma_{i,k}$ should be given the current state estimate. By solving for the negative logarithm (ignoring unnecessary constants) of the initial product given in equation (1.5), we can reformulate the solution $\boldsymbol{\Theta}^*$ as a minimization problem:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \prod_{i,k} \left\|h(\boldsymbol{\Theta}_i) - \boldsymbol{z}_{i,k}\right\|^2_{\Sigma_{i,k}} \tag{1.7}$$

Now the problem is a least squares problem with error $e_{i,k} = h(\boldsymbol{\Theta}_i) - z_{i,k}$. However, the function h is typically nonlinear (for example, the camera projection factor). So we cannot use a closed-form solution via the normal equations, as in linear least squares. Solving nonlinear least squares instead requires approximating the error term $e_{i,k}$ around a linearization point and performing a series of smaller linear solving steps, updating the estimate $\hat{\boldsymbol{\Theta}}$ until convergence:

$$\begin{aligned} e_{i,k}(\hat{\boldsymbol{\Theta}} + \Delta\boldsymbol{\Theta}) &\approx \approx e_{i,k}(\hat{\boldsymbol{\Theta}}) + \boldsymbol{J}_{i,k}\Delta\boldsymbol{\Theta} && (1.8) \\ (\boldsymbol{J}^T\Sigma^{-1}\boldsymbol{J})\Delta\boldsymbol{\Theta} &= -\boldsymbol{J}^T\Sigma^{-1}e && (1.9) \\ \boldsymbol{\Theta}^{\boldsymbol{t+1}} &= \boldsymbol{\Theta}^{\boldsymbol{t}} + \Delta\boldsymbol{\Theta}^{\boldsymbol{t}} && (1.10) \end{aligned}$$

This optimization process can be performed by algorithms such as Gauss-Newton or Levenberg-Marquardt, which are described further in [6]. It is important to note that the Hessian matrix $\boldsymbol{H} = \boldsymbol{J}^T\Sigma^{-1}\boldsymbol{J}$ is sparse in SLAM since each factor only involves a small subset of all the variables. This sparsity allows for efficient solving methods from sparse linear algebra to be applied.

CHAPTER 2

SYSTEM DESIGN AND METHODOLOGY

2.1 SLAM System Overview

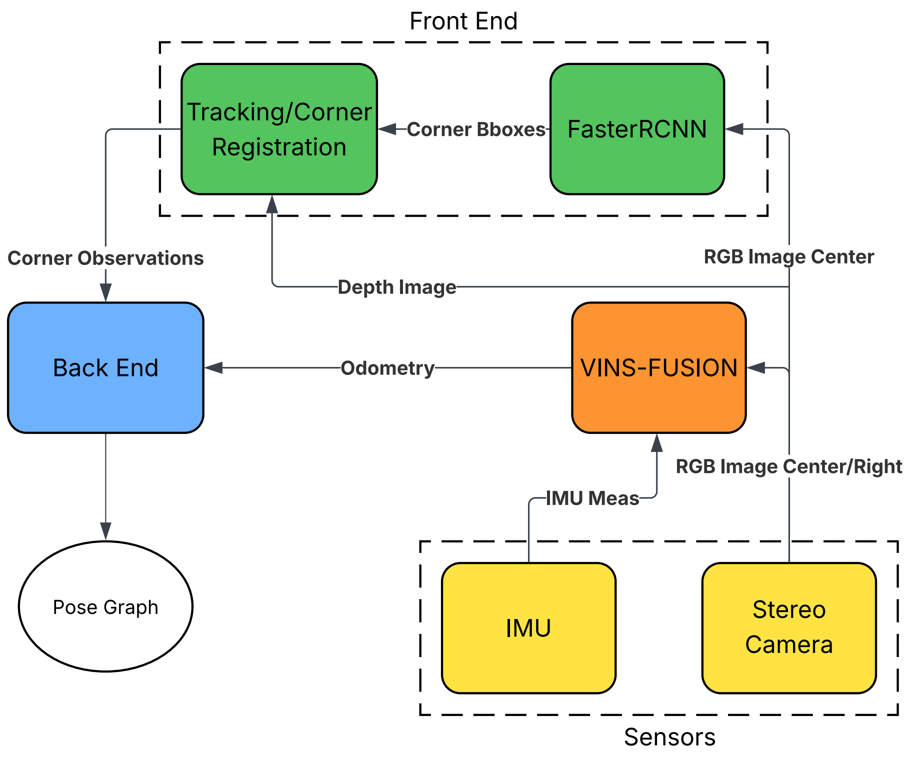

Figure 2.1: Proposed SLAM System Design

The SLAM system receives inertial data from an IMU sensor and synchronized RGB and depth images from a stereo camera. VINS-FUSION, a Visual Inertial Odometry (VIO) module, listens to incoming IMU measurements and RGB image pairs. It then outputs estimated odometry information. At the same time, the Front End receives depth images and RGB images taken from only one of the stereo camera lenses. FasterRCNN [7] is used to generate bounding boxes for any detected gate corners, and these boxes are then further processed in a subsequent refinement, validation, and tracking step. The Front End then outputs data corresponding to 2D gate corner observations with depth information. The Back End then takes this information, along with the odometry information generated by VINS-FUSION [8] [9] [10] [11], to construct and solve a factor graph, which will give estimated poses of the drone over time as well as the positions of previously detected drone gates.

2.2 Front End

2.2.1 Gate Corner Detection

Known for its competitive accuracy in object detection tasks, Faster-RCNN is used to output bounding boxes representing detected gate corners. The model is trained on the 2019 *AlphaPilot Challenge* dataset, which comprises around 9,000 real-world, annotated images of the *AlphaPilot* drone gate. Similar to the training method used in [12], a copy of each image is created, and random rotations and minor adjustments to saturation, brightness, hue, and contrast are applied to it.

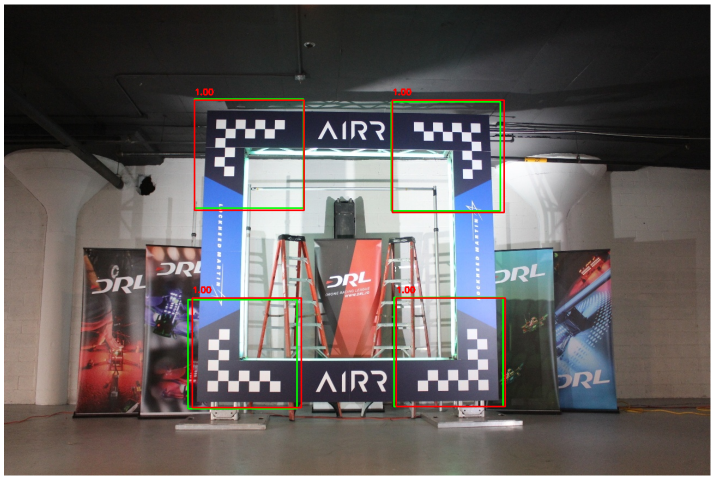

Figure 2.2: Faster RCNN training example. Inference (red), label (green).

2.2.2 Corner Point Refinement

If no gate corners are currently being tracked and at least four bounding boxes have appeared, a corner point refinement step is conducted within each bounding box region. This step attempts to extract a precise target point representing a corner of the inner edge of the gate. The bounding boxes are applied as smaller regions of interest (ROI) onto the depth image. A median filter is applied to the depth image beforehand to remove salt and pepper noise. Then a Gaussian blur is applied, followed by Canny edge detection. Afterwards, the Hough transform algorithm is used to extract line segments. The 2D target point is then selected as the closest intersection of line segments to the center of the bounding box. If a gate corner exists within the bounding box, then ideally, there should be two lines representing the inner edges of the gate that create an intersection point close to the center of the bounding box. If no intersection points are detected, the bounding box is discarded.

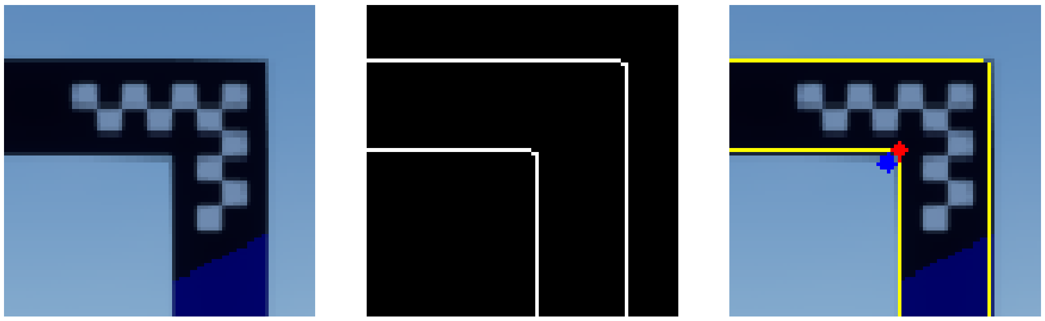
(a) Original RGB ROI (b) Canny Edge Filter (c) Hough lines (yellow), bbox center (blue), and target point (red)

Figure 2.3: Corner point refinement

2.2.3 Gate Registration and Tracking

If at least four target points are successfully extracted, the closest four points based on depth are chosen for further examination. Using the camera's intrinsic, each target point and its corresponding depth are back-projected into 3D space. Since the gate dimensions are known beforehand, a geometric verification step is performed by checking the Euclidean distances between one of the 3D target points and the other. Denoting the set of 3D target points as $P = (p_0, p_1, p_2, p_3) \in \mathcal{R}^3$ and selecting p_0 as the reference point, the following criteria (up to a small tolerance) must be satisfied for P to be considered as forming a gate:

1. Distance $\mathbf{d}$ between p_0 and exactly two other points $p_q, p_r \in (P - \{p_0\})$ is approximately the same as the gate's true inner edge length.
2. Distance between p_0 and the remaining point $p_s \in (P - \{p_q, p_r, p_0\})$ is approximately $d \cdot \sqrt{2}$

The criteria, which utilize the geometric properties of a square, verify whether the four selected target points form a square that matches closely with the dimensions of the gate.

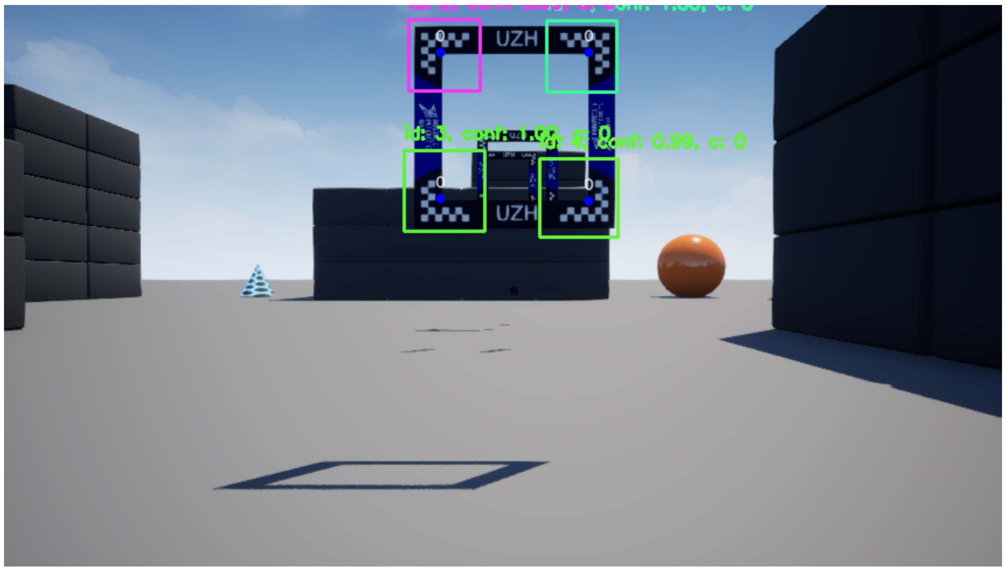

Figure 2.4: Full annotated image with target point locations and depth

If a candidate gate is found, the Front End enters tracking mode and will track the four 2D target points until they are lost. The tracking algorithm selected for the SLAM system is ByteTrack [13], a state-of-the-art (SOTA) tracking algorithm with a very low computational footprint. Once all target point tracks have been lost, the Front End returns to its previous mode, and the process repeats as new gates are found.

2.3 Back End

2.3.1 Visual Projection Factors

The Back End takes tracked 2D target points from the Front End and adds the proper constraints to its factor graph. First, visual projection factors are added, constraining estimated 3D corner positions to the current drone pose via world coordinate to image plane projection errors. To formulate this, we first define the camera's pose in the world frame $\mathbf{T}^{WC}$ as follows:

$$\mathbf{T}^{WC} = \mathbf{T}^{WB}\mathbf{T}^{BC} = \begin{pmatrix} \mathbf{R}^{WC} & \mathbf{t}^{WC} \\ \mathbf{0}^{T} & 1 \end{pmatrix} \in \mathrm{SE}(3) \tag{2.1}$$

where $\mathbf{T^{WB}}$ is the drone's pose in the world frame and $\mathbf{T^{BC}}$ the camera's pose in the drone's frame. Using the camera's intrinsic matrix $\mathbf{K}$, the camera's rotation in the world frame $\mathbf{R}^{WC} \in \mathrm{SO}(3)$, and the camera's translation in world frame $\mathbf{t}^{WC} \in \mathbb{R}^3$, a 3 x 4 projection matrix can be constructed which maps 3D corner points to 2D points on the camera image plane:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K}\left[\mathbf{R}_{WC}|\mathbf{t}_{WC}\right] \begin{pmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{2.2}$$

The projection factor will then push the factor graph solving to minimize the squared norm of the projection error (with some measurement noise) between the measured 2D point (u', v') and the predicted 2D point $(\hat{u}, \hat{v})$ based on the estimate of the corner position $\hat{P} = (\hat{X}_W, \hat{Y}_W, \hat{Z}_W)$. After adding a projection factor between the drone's current pose variable and creating a new 3D corner position variable, an initial guess for the corner position is calculated by backprojecting the 2D corner measurement with its associated depth from the pose generated by VINS-FUSION.

2.3.2 Ranging Factors

The known physical geometry of the square gates is leveraged to add rigid constraints between adjacent gate corners variables $\boldsymbol{l}_i$ and $\boldsymbol{l}_j$ with known inner edge distance d_{inner}. This prior knowledge is subsequently incorporated as a ranging factor that aims to minimize the cost function:

$$C_{ij} = \| \|\boldsymbol{l}_i - \boldsymbol{l}_j\|_2 - d_{\text{inner}} \|^2_{\Sigma_{ij}} \quad \text{where} \quad \Sigma_{ij} = [\sigma^2_{ij}] \in \mathbb{R}^{3\text{x}3} \tag{2.3}$$

This cost function, like all other cost functions for factors, is represented as a squared Mahalanobis distance to account for unpredictability modeled by a covariance matrix. To ensure the rigidity of this ranging factor, since the distances between the gates are known, the covariance matrix is set to $\Sigma = \sigma I_{3\text{x}3}$ where variance σ is a value close to 0.

A ranging factor is also used to constrain drone pose variables in SE(3) and landmark variables in $\mathbb{R}^3$ using depth information when corner points are observed. The cost function is subsequently represented as:

$$C_{ij} = \| \|p_i - l_j\|_2 - z_{i,j} \|^2_{\Sigma_{ij}} \quad \text{where} \quad \Sigma_{ij} = [\sigma^2_{ij}] \in \mathbb{R}^{3\text{x}3} \tag{2.4}$$

where p_i is the pose at timestamp i, l_j is an observed landmark at that timestep, and $z_{i,j}$ is the corresponding depth from the depth image.

2.3.3 Between Factors

A between-factor is used to constrain temporally adjacent drone pose variables using the odometry information provided by VINS-FUSION. A between factor constrains two pose variables T_1,$T_2 \in$ SE(3) with a relative rigid-body transformation between the two poses. The cost function for the between factor is:

$$C_{ij} = \| \operatorname{Logmap}((T_i^{-1} T_j) Z_{ij}^{-1}) \|^2_{\Sigma_{ij}} \quad \text{where} \quad \Sigma_{ij} = [\sigma^2_{ij}] \in \mathbb{R}^{6\text{x}6} \tag{2.5}$$

The term $T_i^{-1}T_j$ represents the relative transformation from frame i to frame j where i is the timestep just before j. Z_{ij} represents the measured transformation between the two frames. Therefore, the product of the predicted relative transformation $T_i^{-1}T_j$ and the inverse of the measured relative transformation Z_{ij} will create a transformation representing the error between the predicted and measured relative transforms. This matrix, which belongs to the Lie group SE(3), is then mapped to its corresponding Lie algebra $\mathfrak{se}(3)$ with the Logmap operator. This is done to convert the matrix into its vector representation, as the optimization algorithms used for solving factor graph problems operate on vector spaces.

CHAPTER 3

IMPLEMENTATION

3.1 Software Used

The proposed SLAM algorithm was implemented in Python version 3.10 using the following software packages:

Table 3.1: Software components used in the implementation.

Software	**Version**	**Purpose**
`NumPy`	1.26.4	Array algorithms and frame transformations
`GTSAM`	4.2	Factor graph modeling and optimization.
`OpenCV`	4.11.0	Image filtering, transformations, Hough transform, and non-maximum suppression.
`PyTorch`	2.6.0	FasterRCNN training and inference.
`ByteTrack`	1.0	Bounding box tracking.
`VINS-FUSION`	1.0	Visual Inertial Odometry (VIO)
`Cosys-AirSim`	3.2.0	Drone Simulation in Unreal Engine 5

3.2 Simulation Environment

3.2.1 Cosys-AirSim

The proposed SLAM algorithm was evaluated using Cosys-AirSim [14], a significantly enhanced fork of Microsoft's AirSim [15]. Leveraging the photorealistic graphics and interactive API provided by its Unreal Engine 5 foundation, Cosys-AirSim introduces modifications such as extended sensor simulation and improved ground truth annotation features, which are helpful in testing robot perception algorithms. This simulator was chosen for its native ROS 2 support. It facilitates data serialization, an active development team that ensures continuous improvements, and an enhanced computer vision mode that offers improved data collection. Furthermore, its ease of use and physically accurate drone dynamics simulation make it an ideal platform for this research.

3.2.2 Experiment Setup

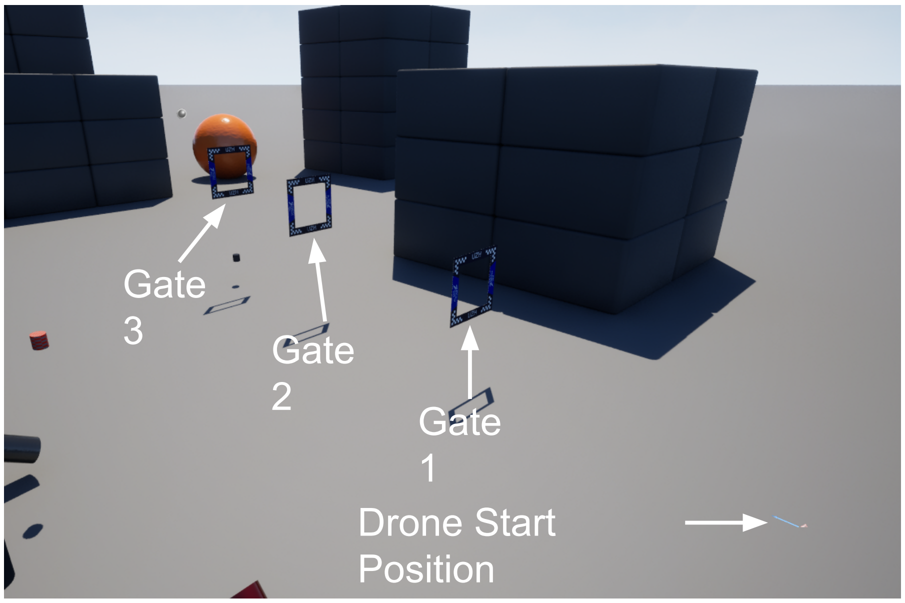

Figure 3.1: Simulation configuration for the experiment

For evaluating the SLAM algorithm, a modified version of the default Unreal Engine 5's blocks world is used. Three drone gates are placed approximately five to ten meters apart in the world as shown in figure 3.1. The experiment starts with the drone placed on the ground approximately 20 meters from the first gate. After the drone takes off a couple of meters off the ground, it travels on a straight-line trajectory at 3 meters per second towards the third gate. The drone's path ends right before it crosses through the third gate, around eight meters above the ground.

The stereo camera and IMU sensors are configured with the following settings:

Table 3.2: Simulated Stereo Camera Parameters

Parameter	**Value**	**Units**
Resolution (Width)	854	pixels
Resolution (Height)	480	pixels
Field of View (FOV)	90	degrees
Baseline	0.5	meters
Update Rate	20	Hz

Table 3.3: Simulated IMU Noise and Bias Parameters (using siunitx)

Parameter	**Value**	**Units**
AngularRandomWalk	0.3	$°/\sqrt{\mathrm{h}}$
GyroBiasStabilityTau	500	s
GyroBiasStability	4.6	$°\,\mathrm{h}^{-1}$
VelocityRandomWalk	0.24	$\mathrm{m/s}/\sqrt{\mathrm{h}}$
AccelBiasStabilityTau	800	s
AccelBiasStability	36	µg*

* Unit displayed as µg represents micro-gravity where $1\mathrm{g} \approx 9.81\,\mathrm{m\,s^{-2}}$.

CHAPTER 4

EXPERIMENTAL EVALUATION

4.1 Results

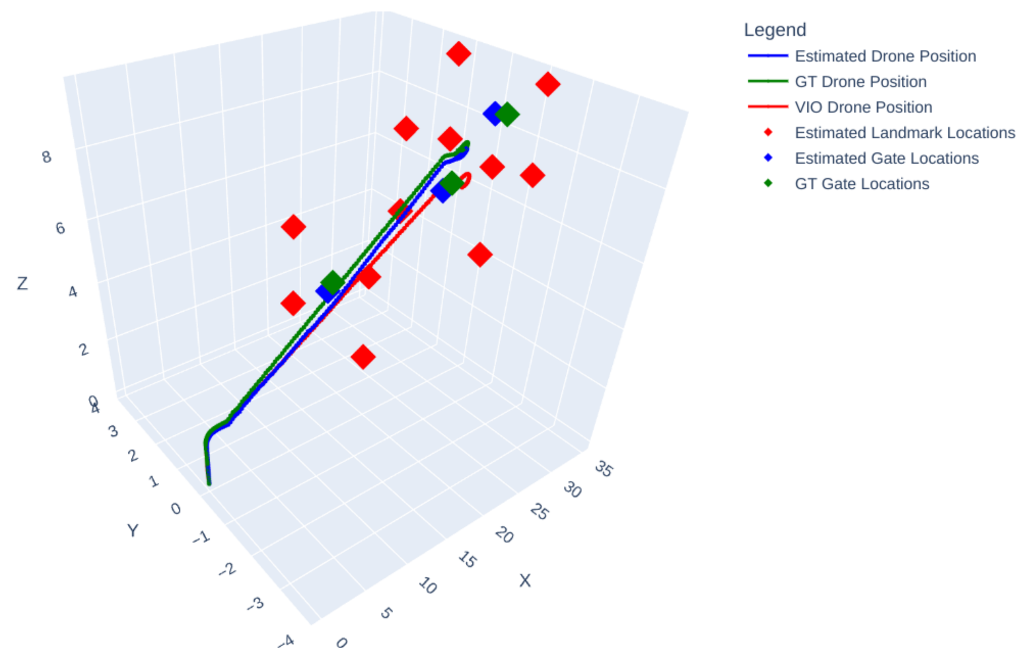

Figure 4.1: 3D plot showing the output of the pose graph solve with ground truth information

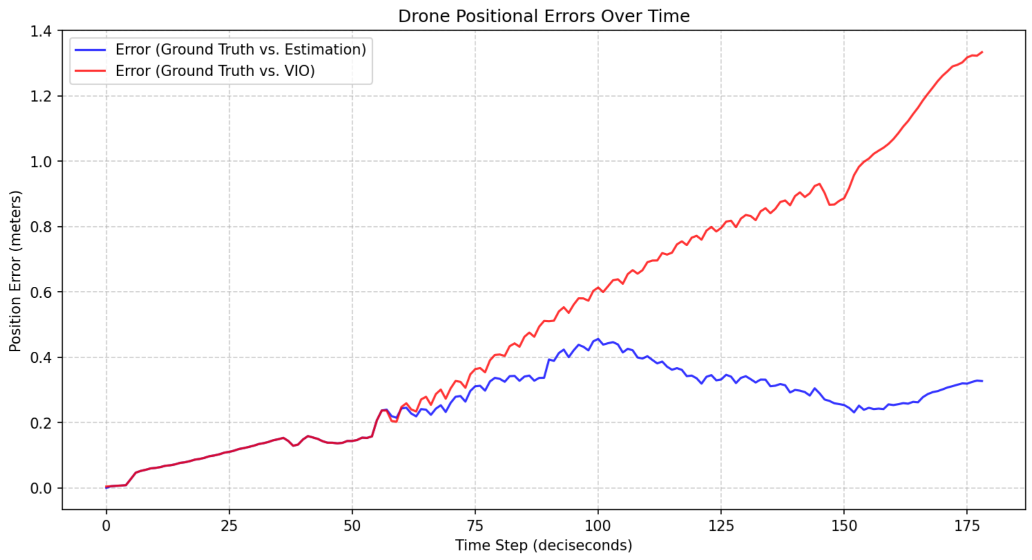

Figure 4.2: 2D graph showing the drone positional error over time (L_2 norm with ground truth)

Table 4.1: Drone Positional Errors with Proposed SLAM

Axes	RMSE (m)	Std Dev (m)
X-Axis	0.0571	0.0468
Y-Axis	0.1069	0.0675
Z-Axis	0.2484	0.1163
XYZ Axes (L_2 Norm)	0.2764	0.1145

Table 4.2: Drone Positional Errors with only VINS-FUSION

Axes	RMSE (m)	Std Dev (m)
X-Axis	0.2651	0.2228
Y-Axis	0.0418	0.0360
Z-Axis	0.6003	0.3435
XYZ Axes (L_2 Norm)	0.6576	0.3868

Table 4.3: Drone Rotational Errors with Proposed SLAM

Axes	RMSE (degrees)	Std Dev (degrees)
Roll	0.6245	0.4370
Pitch	1.2170	1.0633
Yaw	0.4147	0.3143
Roll-Pitch-Yaw (L_2 Norm)*	1.4188	0.9475

Table 4.4: Drone Rotational Errors with only VINS-FUSION

Axes	RMSE (degrees)	Std Dev (degrees)
Roll	0.0306	0.0260
Pitch	1.0693	1.0692
Yaw	0.0108	0.0101
Roll-Pitch-Yaw (L_2 Norm)*	1.0698	0.9839

* Calculated by taking L_2 norm of the logmap of the measured rotation matrix.

Table 4.5: Comparison of Ground Truth and Estimated Gate Positions

Gate	Ground Truth Position			Estimated Position			Distance Error
#	x (m)	y (m)	z (m)	x (m)	y (m)	z (m)	(m)
1	14.300	−0.200	5.190	14.003	−0.130	4.914	0.411
2	26.800	−0.600	7.190	26.492	−0.437	6.920	0.441
3	34.900	−0.300	8.390	34.575	−0.037	8.322	0.423

4.2 Discussion of results

The SLAM method has proven to be capable of detecting and estimating the positions of the gates within the course while also estimating the drone's position. The average error in the estimated drone position compared to the ground truth position is around 0.27 meters, while VINS-FUSION's average positional error is around 0.65 meters. This indicates that including the gates as SLAM landmarks has significantly reduced the VIO drift from VINS-FUSION. Intuitively, this makes sense as the gates provide additional geometric information that can improve the positional state estimate of the drone. On the other hand, the average angular error increases by around

0.35 degrees when using the proposed SLAM method, when it is expected to decrease. This discrepancy could be considered negligible due to how minor the angular errors are, but one potential cause could be in the measurement noise models of the gate projective factors. Perhaps increasing the variance in the pixel measurements within the projection factors could allow more leeway for the factor graph solver to converge on a more optimal solution.

It is interesting to note that the drone positional error is, in fact, the same as the VINS-FUSION positional error up until around 60 deciseconds into the experiment, as seen in figure 4.2. This further validates the effectiveness of including gate estimation because the point where the VINS-FUSION error and the SLAM method error start to diverge is when the first gate is detected.

CHAPTER 5

CONCLUSION

This research presented and evaluated a Simultaneous Localization and Mapping (SLAM) system tailored for autonomous drone racing, utilizing a factor graph optimization approach. The system integrates data from a camera and an IMU sensor to concurrently localize the drone and map the positions of race gates. Key components include a convolutional neural network for gate corner detection, a tracking algorithm for these corners, and a tightly coupled Visual Inertial Odometry (VIO) module for odometry generation. These elements are combined within a factor graph framework.

An experimental evaluation within a custom Unreal Engine simulation demonstrated the system's capability to estimate drone pose and gate locations with reasonable accuracy (average gate and drone positional error < 0.6 m). The results confirm that integrating gate detections significantly improves drone positional accuracy compared to using VIO alone.

Future improvements to the proposed SLAM system include running it in real time, testing with real-world data, and creating more complex factor noise models. Due to limitations with the inference speed of FasterRCNN, a different object detection model can be used instead to perform much faster inference, allowing the algorithm to run in real time. Additionally, testing with data from a drone going through a real-world course will allow a more comprehensive examination of the perception module's robustness to visual disturbances such as lighting artifacts and camera distortion. Lastly, the noise models within the factor graph framework were statically and arbitrarily chosen. Changing the noise models dynamically, depending on particular scene conditions and the state of the drone, could improve the accuracy of the state estimation, especially in more challenging scenarios.

REFERENCES

[1] F. Dellaert, “GTSAM introduction,” GTSAM Website Tutorial, accessed: 2025-04-20. Available at: https://gtsam.org/tutorials/intro.html.

[2] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, “Champion-level drone racing using deep reinforcement learning,” *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023. [Online]. Available: https://doi.org/10.1038/s41586-023-06419-4

[3] D. Gelez, E. Kaufmann, A. Loquercio, and D. Scaramuzza, “Autonomous drone racing in dynamic environments with recurrent attention policies,” in *Proc. Robotics: Science and Systems (RSS)*, Delft, The Netherlands, July 2024.

[4] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[5] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques.* Cambridge, MA: The MIT Press, 2009.

[6] F. Dellaert and M. Kaess, *Factor Graphs for Robot Perception*, ser. Foundations and Trends(R) in Robotics. Hanover, MA, USA: Now Publishers Inc., 2017, vol. 6, no. 1–2.

[7] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[8] T. Qin, P. Li, and S. Shen, “Vins-mono: A robust and versatile monocular visual-inertial state estimator,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[9] T. Qin and S. Shen, “Online temporal calibration for monocular visual-inertial systems,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2018, pp. 3662–3669.

[10] T. Qin, J. Pan, S. Cao, and S. Shen, “A general optimization-based framework for local odometry estimation with multiple sensors,” 2019.

[11] T. Qin, S. Cao, J. Pan, and S. Shen, “A general optimization-based framework for global pose estimation with multiple sensors,” 2019.

[12] A. Schmidt, “A comparison of gate detection algorithms for autonomous racing drones,” in *2022 IEEE Aerospace Conference (AERO)*. Big Sky, MT, USA: IEEE, 2022, pp. 1–13.

[13] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, “ByteTrack: Multi-object tracking by associating every detection box,” in *Proc. European Conference on Computer Vision (ECCV)*, 2022.

[14] W. Jansen, E. Verreycken, A. Schenck, J.-E. Blanquart, C. Verhulst, N. Huebel, and J. Steckel, “Cosys-airsim: A real-time simulation framework expanded for complex industrial applications,” in *2023 Annual Modeling and Simulation Conference (ANNSIM)*, 2023, pp. 37–48.

[15] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065