Regionally Adaptive Covariance Estimation (RACE) for SLAM in Uncertain Dynamic Environments

Braden Eichmeier, Shaun Ryer, Alex Withers, Quentin Cheng
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Email: {beichmei, sryer, awithers, hantingc}@andrew.cmu.edu
Git Repository: https://github.com/eichmeierbr/adaptiveSLAM

***Abstract*—Modern approaches to solving the simultaneous localization and mapping (SLAM) problem optimize a factor graph to better estimate an agent's pose and surrounding environment. However, the standard optimization uses the sensors' expected covariance to weight their influence during the optimization call, which causes error in the optimized path if the covariance in the environment is different than expected. We propose using a method that utilizes a Neural Network (NN) to weight the sensors based on an online standard deviation computation relative to the other sensors, a metric we coined as psuedo-covariance. Using this method we are able to dynamically re-weight and influence the optimizer to produce a result close to the true path even with very large noise and model changes. Preliminary simulation testing for this method shows a 5.7x improvement compared to relying on expected covariance measurements.**

I. Introduction

Modern approaches to solving the simultaneous localization and mapping (SLAM) problem focus on optimizing a factor graph of sensor measurements and poses. In this formulation a sparse matrix is formed and solved using a least squares approach. Due to the linear optimization approach, this method lends itself best to purely linear systems. However, established techniques exist for nonlinear systems as well.

Formulating the SLAM problem as an optimization involves minimizing the error between a sensor measurement vector z_i and the sensor model vector $h_i(\theta)$. Linearizing the sensor model at an estimated state (H_i) reduces the computation to minimizing the Mahalanobis distance at the linearization point (1). Clever redefinition of the system with the covariance matrix Σ_i reduces the computation further into a more concise linear optimization (2) [4].

$$X_{opt} = \underset{x}{\operatorname{argmin}} ||H_i x - (z_i - h_i(x))||^2_{\Sigma_i} \tag{1}$$

$$X_{opt} = \underset{x}{\operatorname{argmin}} ||\Sigma^{-1/2}(H_i x - (z_i - h_i(x)))||^2 \tag{2}$$

The expected standard deviation for each factor can be viewed as a weighting factor that emphasizes the relative importance of each sensor measurement. Indeed, it can be seen as a generic confidence metric (w) of a given sensor at a given time (Eqn. 4).

$$X_{opt} = \underset{x}{\operatorname{argmin}} ||\Sigma^{-1/2} * (A_i x - b_i)||^2 \tag{3}$$

where

$$A_i = H_i$$

$$b_i = z_i - h_i(x)$$

In well structured environments this weighting factor is mostly constant and can reliably be used throughout the course of the algorithm execution. Application in varied situations in the wild may cause sensors to change in expected quality of performance over time. As such, we represent the previous optimization with a time varying weighting vector:

$$X_{opt} = \underset{x}{\operatorname{argmin}} ||w(X,t) * (A_i x - b_i)||^2 \tag{4}$$

As an example we consider the setting where a vehicle is equipped with GPS, visual odometry, and wheel odometry. As the car drives through rural regions on the highway effective localization can utilize purely the GPS with intermediate updates performed with the noisy odometry. Visual sensing in these regions may suffer due to the lack of features. However, as the vehicle travels into a metropolitan area with high buildings the GPS may begin to suffer while the visual component begins to perform reliably. In this transient setting, the optimization would need to effectively re-balance the weighting factors to achieve satisfactory results.

We present a regionally adaptive weighting scheme to the optimization procedure. Specifically, we convert the joint sensor measurements in a given region into a pseudo covariance estimation. This is done by iteratively assuming each sensor is perfect, and comparing the other sensors' output to the assumed true sensor. A neural network learns to untangle the relationships the psuedo-covariance values and the underlying distribution for each sensor. Finally, the weights are computed from these predicted standard deviations.

II. Related Works

A. Robust SLAM

Adaptive factor weighting falls under a broad range of research across various fields of estimating the covariance in data. In recent SLAM literature this problem is framed under Robust Map Optimization and focuses on removing outliers. Several works along this front have focused on defining the

likeliness of the outlier according to a binary switching variable [3]. The original formulation simply attempted to quantify whether or not a single data point as an outlier, but heavily increased computational complexity [9]. This work developed via marginalization in an approach titled Dynamic Covariance Scaling [1][2]. The same author continued the work using a max-mixture, where the probability p($z_i|x$) is computed as the maximum of a mixture of Gaussian samples [7]. Work to evaluate the algorithms has found good results, but slightly worse performance than iSAM2 [8]. Watson improved the max mixture approach using Gibbs sampling of a Dirichlet process to characterize the covariance [11].

B. Covariance Estimation for State Estimations

There has also been many attempts at learning the covariance matrices of sensors for the state estimation problem. CELLO [10] approximated the covariance of a sensor at a given state as the sample covariance of neighboring states in a hand-coded feature space. It was able to automate the covariance tuning step for EKFs and allow for varying covariances within the algorithm. However, the need for hand-engineering features somewhat defeats the purpose. Similarly, [5] also relies on hand-coded features, albeit providing a more streamlined approach to learning the covariance mapping.

After deep learning gained significant popularity, the authors of CELLO proposed a new algorithm, DICE [6], utilizing convolutional neural networks to learn the mapping from raw sensor measurements to the covariance matrices. It does so through maximizing the likelihood of producing a training set of sensor errors given the predicted covariance. The training sensor errors are generated in a controlled environment with ground truth sensors such as the Vicon cameras, and subtracting the raw sensor data from it. The authors were also clever in adding a layer of post-processing between the NN output and the covariance matrix to relieve the need of producing positive definite outputs from the NN.

While this approach shows promise in single sensor, especially visual sensor scenarios, it does not include meaningful information from the map when the sensor output is simplistic. For example, learning the mapping from a GPS's output directly to the covariance matrix does not in fact consider the effect of nearby skyscrapers, and learning the mapping from wheel encoder output to the covariance matrix cannot encapsulate tire slippage. Furthermore, when applying to the SLAM problem, by having the learning purely offline the accuracy of past estimations and their relationship to current and future estimations are lost. Instead, we look at using other sensors' measurements to help in predicting the current sensor's uncertainty.

III. Methods

A. Robot and Sensor Models

The robot motion and sensors all use a basic linear scheme to focus more time and effort on the optimization and adaptive weighting algorithms. Accordingly, the robot motion model follows a basic (x,y) point control where the controller directly commands the change in robot state (5).

$$p_{t+1} = p_t + u_t \tag{5}$$

The robot has 3 methods to sense the environment. The first method is an odometry sensor that outputs the estimated change in pose at each time step (6). The second sensor is a relative position sensor that mimics sparse feature detection (7). The final sensor is a global position sensor that reports the true position of the robot (8). Each of these sensors are linear with Gaussian noise added to the true values.

$$Z_{odom} = p_{t+1} - p_t + N(0, \sigma) \tag{6}$$

$$Z_{feature} = l_i - p_t + N(0, \sigma) \tag{7}$$

$$Z_{GPS} = p_t + N(0, \sigma) \tag{8}$$

In practice each sensor in a localization system will have its own expected covariance. In this simulation, we assume the sensors have an equal level of noise. As such, the expected covariance for each sensor will be set to 1. We also make the assumption that the noise is drawn from a zero meaned Gaussian.

B. Simulation World

For the simulation environment we created a 2-D world populated with areas where the robot can go and areas that the robot cannot. The simulation is passed between the robot and environment as shown in Fig. 1. This environment produces a set of noisy data for the SLAM problem and true data for verification and reporting.

The simulation maps are divided into quadrants. Quadrant 1, where the robot begins, produces sensor results according to the expected sensor noise. The remaining quadrants increase the noise of one of the sensors by an order of magnitude. Applying the sensor noise in this manner creates stark changes in sensor quality without notice to the algorithm.

C. On the Fly Covariance Estimation

1) Pseudo-Covariance Computation: The main difficulty in computing the covariance of a sensor is the true path is unknown. We propose extracting the covariance of each sensor by computing the covariance between sensors over a small period of time. In this process we assume the measurement noise is i.i.d between both sensors and measurements. To resolve the problem of not knowing the true path, we present a method to create some semblance of ground truth.

Because the final optimized path will be some weighted combination of the paths generated by the individual sensors, each sensor individually can be used to generate a nominal path. For a nominal path created by fully trusting one sensor, "pseudo" standard deviations (σ') for the other sensors can be computed relative to the nominal path. Figure 2 shows an

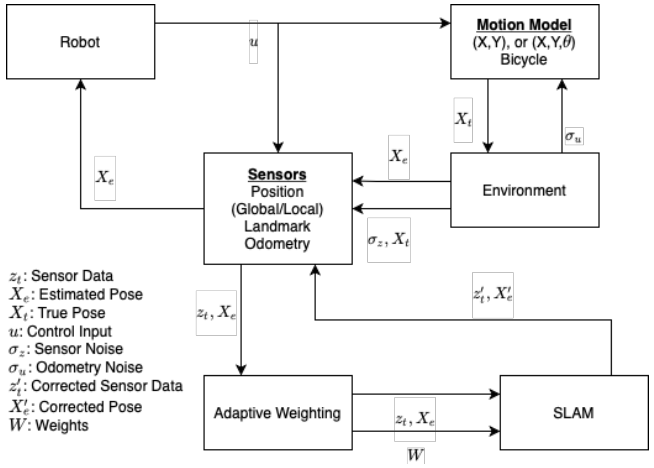

Fig. 1. Outline of Simulation, Robot, and SLAM systems.

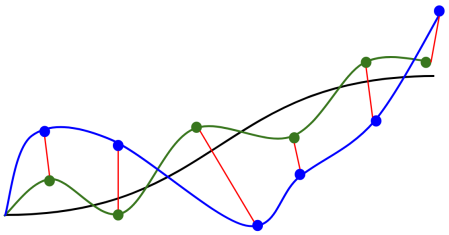

Fig. 2. The pseudo convariance between one sensor (green) and another (blue) can be found as the standard deviation of the error between believed poses of each sensor. True path is shown in black.

example of computing this pseudo covariance between two sensors.

2) Extracting Weights from the Pseudo-Covariance Matrix: We use a neural network to map the pseudo covariances directly to the sensor standard deviations. The neural network learns the underlying sensor error function with respect to the sensor outputs, and maps the pseudo covariances into final covariance values that following Gaussian distributions. The base features to the model are the upper-triangular elements in the pseudo covariance matrix. In many settings, external knowledge can create a prior to better bias the network's prediction. These priors could come from environment knowledge, or nominal sensor noise. We augment the base features with a prior to improve prediction accuracy.

To protect from prediction instabilities, bounds are placed on the predictions to enforce both maximum and minimum standard deviation predictions. Initial testing in this paper computes the resultant weights as the inverse of the bounded network predictions. However, more complex non-linear functions may be explored in future testing to produce more stable results. Algorithm 1 shows the basic outline for the weight extraction procedure.

D. Optimization

The SLAM component of the algorithm uses the Levenberg-Marquardt algorithm to optimize the factor graph. The optimizer runs at a regular interval (every 10 simulation steps currently). In each optimization call, the algorithm optimizes a set of unknown values (the poses and landmarks) according to some constraints (sensor readings). An error function uses these data to create reducible cost for the optimizer, which iteratively computes the optimal values. In this framework, each sensor must have a an error function that returns an error based on corresponding pair of estimated values and measurement.

Algorithm 1 Pseudo Covariance Weight Extraction

procedure EXTRACTWEIGHTS($\vec{z}$, $\vec{p}$)
  Input $\vec{z}$: Sensor data from m sensors
  Input $\vec{p}$: Prior estimate of sensor covariance
  Initialize: $\sigma'_{m,m} = 0$
  Compute nominal path X_i for each sensor z_i
  for $i \in \{0, m\}$ **do**
    for $j \in \{i+1, m\}$ **do**
      $\sigma'_{i,j} = std(X_i - X_j)$
  $\Sigma = predict(\sigma', \vec{p})$
  $w = \Sigma^{-1/2}$
  return w
procedure PREDICT(σ', $\vec{p}$)
  Input σ': Pseudo covariance values
  Input $\vec{p}$: Prior estimate of sensor covariance
  while Not converged **do**
    $\vec{p} = NN\_predict([\sigma', \vec{p}\,])$
  return w

At optimization time, the previous 2n (20 in this setup) time steps are sent for optimization. This means each data point is called twice and is done to correct poorly optimized points once new data gives more insight to the error model both before and after a time step. The weights for each time step are computed using a sliding window of size n. The first n/2 time steps use the first n measurements. The next n time steps use the n/2 measurements before the time step and the n/2 measurements after the time step to compute the weights. The final n/2 data points use the last n measurements. Equation 9 shows this setup. Once the weights are computed, the optimizer runs the weighted least squares optimization given by Eqn. 4.

$$w_t = \begin{cases} extractWeights(\vec{z}_{0:n}), & t < n/2 \\ extractWeights(\vec{z}_{n:2n}), & t > 3n/2 \\ extractWeights(\vec{z}_{t-n/2:t+n/2}), & otherwise \end{cases} \quad (9)$$

E. Neural Network Design and Prediction

The neural network used to extract the individual sensor measurements from the pseudo uses a 2-layer MLP regression scheme. Each layer has 30 nodes with ReLU activation functions. The network trained using a dataset comprised of simulated sensor distribution paths. Each sample drew the

target covariances from a bounded [1,50] uniform distribution. Then the pseudo covariances were computed by sampling trajectories from each of the distributions. Finally, zero-mean noise (σ_p) is added to the true covariances, which are added as a prior to the feature. The final sample is given by Eqn. 10.

$$X_i = ([\sigma_i', N(\Sigma_i, \sigma_p)], \Sigma_i) \tag{10}$$

At prediction time the pseudo covariance values are concatenated with the prior and fed into the network. In settings where the prior is informative, a single prediction tends to give informative results. However, when the underlying sensor distribution changes suddenly the prior can hinder accurate prediction. In this paper we assume the weighting algorithm is environment agnostic and only has access to the raw sensor data. To alleviate the results of a poor prior, we use the prediction from the network as a new prior to iteratively converge to a solution. Testing showed this method converged to a value within few iterations, and the resulting prediction values were similarly accurate as a single prediction with an informative prior.

IV. Results

To test the system a simplistic simulation environment was created. This was populated with features, sensors, and a robot model. The models are as general as possible so that sensors or models could be changed later on. Currently only linear models for the sensor and robot are being used, so that the team could focus on the adaptive component of the algorithm.

In the following figures, black areas are zones that the robot cannot enter (i.e. walls). The blue line or noisy path shows the path the robot would think it's traversed if only odometry values were used. As seen in Fig. 3 the noise continuously compounds and experiences drift over the course of the path. The green line shows the true path travelled by the robot. The yellow path (optimize path) shows the path optimized by the SLAM algorithm. We also included the true and estimated positions of detected features in blue and orange respectively. Figure 3 shows effective path optimization with low noise reliable data.

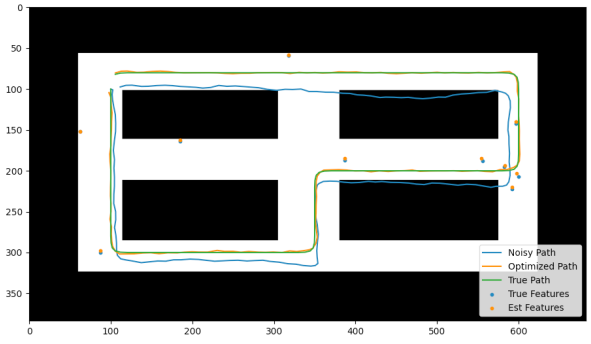

Fig. 3. SLAM using low noise sensors.

Figure 4 shows the importance of finding a dynamic weighting approach for factor graph SLAM optimization. In this test, the world is split into regions, each region has a different color and noise for each sensor. In the gray region, noise exist but is minimal, for the red regions noise in the GPS is increased by 50 times. The blue region increases noise in the odometry sensor 50 times, and in the green region feature detection is increased 50 times. In the real world, this can be caused by significantly high noise, or sensor failures caused by damage. Due to the changing noise models the optimizer was unable to produce good results.

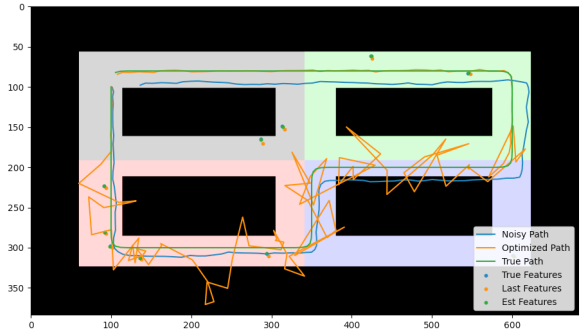

Fig. 4. SLAM without dynamic weighting (different colored regions have different noise values).

Using the adaptive weighting scheme discussed in methods, the same path was run again, the results can be seen in Fig. 5. This resulted in a significantly better path than the path found in Fig. 4.

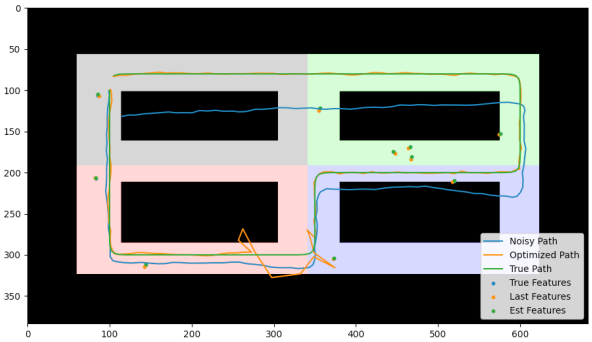

Fig. 5. SLAM with dynamic weighting (different colored regions have different noise values).

For the dynamic weighting the optimization proves decent results, the only errors appear close to the changes from high noise in one sensor to high noise in another. To show this behavior in more clarity, we plotted the different levels of noise of each of the sensors (red green and purple) as well as a baseline using odometry measurements (orange) and our dynamic weighting system (blue) in Fig. 6. The area between high noise of the GPS (red) and odometry (purple), there exist a large peak in the error of the dynamic weighting system.

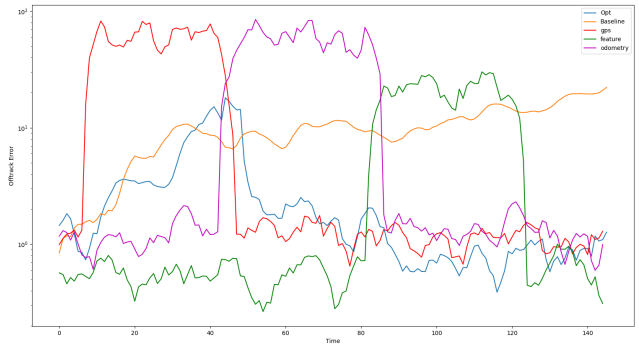

Fig. 6. SLAM without dynamic weighting (different colored regions have different noise values).

This occurs because when the robot initially enters a new region, it takes time to detect a bad region and re-weight the sensors, this means some readings from sensors will have the wrong weighting and lead to errors in optimizations.

In the real world, sharp discontinuities usually aren't as prevalent and gradient would be a better model, however, a significant amount of these errors can be fixed by using sliding window when optimizing the path. Fig. 7 shows the improvement over Fig. 5 when using a sliding window instead of batch for the optimization. This error between regions could possibly be further reduced by using a smoothing function and outlier rejection.

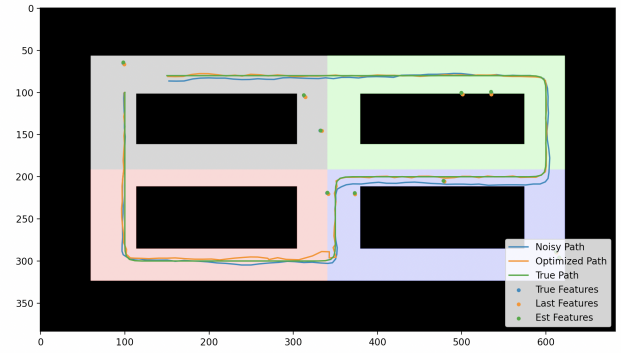

Fig. 7. SLAM without dynamic weighting using a sliding window (different colored regions have different noise values).

To further clarify this point, the error from the true path was collected for the following algorithms: equal weighting for each sensor, dynamic weighting using batch optimization, and dynamic weighting using a sliding window. The results, shown in Table. I, show the major improvement of the dynamic weighting as well as the benefit of using a sliding window when compared to the control (equal weighting). The dynamic weighting shows a 5.7x improvement compared to results from equal weighting.

We also plotted the algorithms by their average off-track error with respect to the errant noise of the sensors, shown in Fig. 17. This illustrates how increasing the noise will usually increase the error quite significantly, as seen in the equal

TABLE I
AVERAGE ERROR OF ALGORITHMS

Algorithm	Average Error
Equal Weighting	2271.66 ± 162.35
Dynamic Weighting (Batch)	540.70 ± 87.01
Dynamic Weighting (Sliding)	392.89 ± 88.57

weighting case. However, our two methods are impacted less by larger noise values when compared to equal weighting.

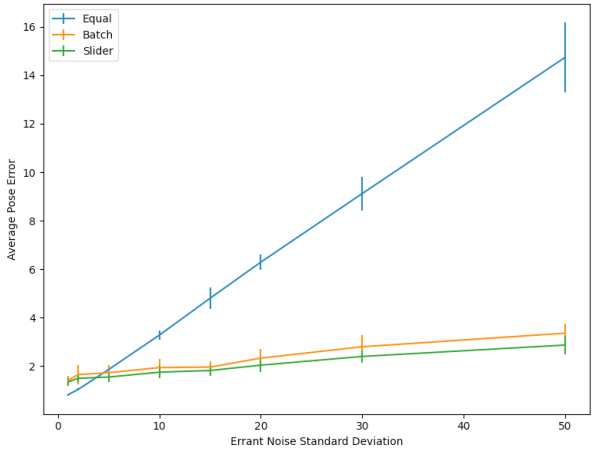

Fig. 8. Error of equal weighting, dynamic weighting (batch), and dynamic weighting (sliding window).

In assessing the robustness of our algorithm, instead of various regions of constant standard deviations, we tested linearly increasing/decreasing it within a region, and having more than one sensor with poor standard deviation at the same location. Figure 9 shows the test case, where on the left half all sensors have low noise, but on the right half, GPS has increasing noise from left to right, and Feature Sensor has decreasing noise. As we can see, before weighting with our algorithm localization is poor.

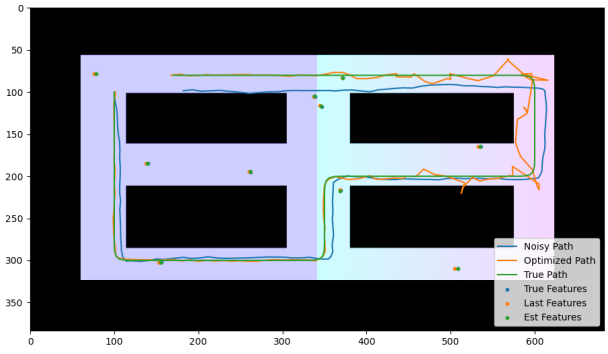

Fig. 9. Linearly Changing Noise GPS and Feature Sensors with no weighting

Figure 10 shows the results after optimization. As we can

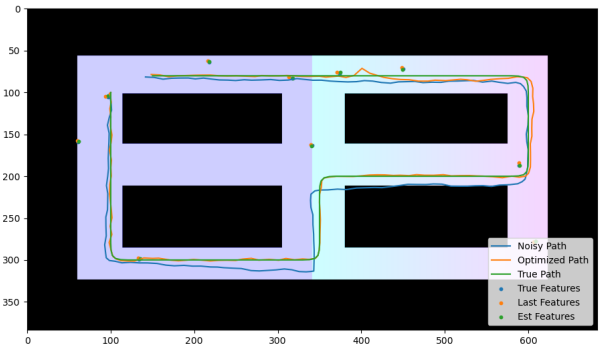

Fig. 10. Linearly Changing Noise GPS and Feature Sensors Optimized

see, even when 2 out of 3 sensors have 20x noise, our algorithm is able to learn to trust the correct sensor for accurate localization.

We also increased the number of sensors to 5, where 3 of them have simple Gaussian errors on the position (GPS like), and tested how many high noise sensors the system is able to withstand at a time. Figures 11 through 14 shows our results. The top left quadrant always has 0 noisy sensors, while the other three quadrants has more than 1 noisy sensors present at a time. Our algorithm is able to handle up to 3 out of 5 noisy sensors at a location, and when even 1 of the sensors has low noise as shown in Figure 13 the algorithm maintains somewhat reasonable tracking. It is only when all 5 sensors have high noise that our system fails.

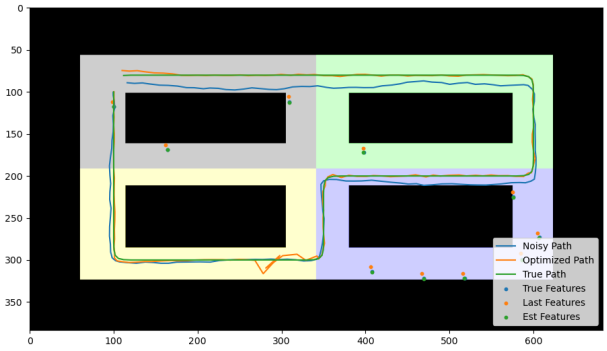

Fig. 11. 2 out of 5 Noisy Sensors - optimized

Lastly, we wanted to see how our system would perform in more realistic scenarios with more gradient noise and less structured paths. Fig. 15 and Fig. 16 shows a different environment with equal weighting and dynamic weighting with sliding window respectively. As expected our system performs much better in this new environment compared to a equal weighting system.

We further the performance analysis beyond the visual confirmation of improved performance between Fig. 15 and Fig. 16. Numerically, this can be shown using the same approach described in Fig. 17. The results mirror the same

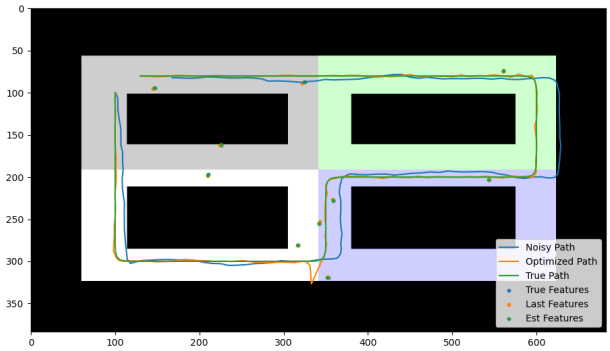

Fig. 12. 3 out of 5 Noisy Sensors

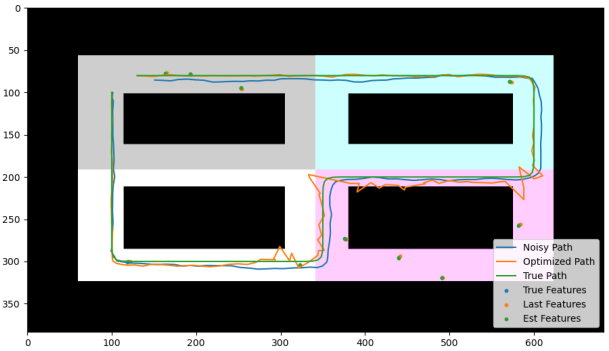

Fig. 13. 4 out of 5 Noisy Sensors - optimized

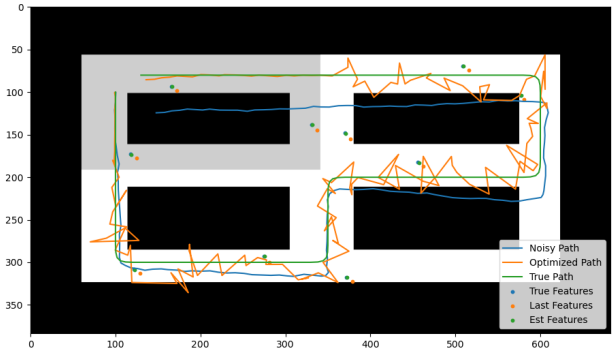

Fig. 14. 5 out of 5 Noisy Sensors

behavior as the previous test: sliding window is the most effective with equal weighting being the least effective, and all methods increase linearly with noise. The equal weighting method performs better than the approximation methods at low noise, up to about a standard deviation of 5, which further confirms the canonical method is optimal when the sensor covariance is known. This testing environment also has higher error for the proposed methods that can likely be attributed to the sensor errors increasing simultaneously near the middle of the map.

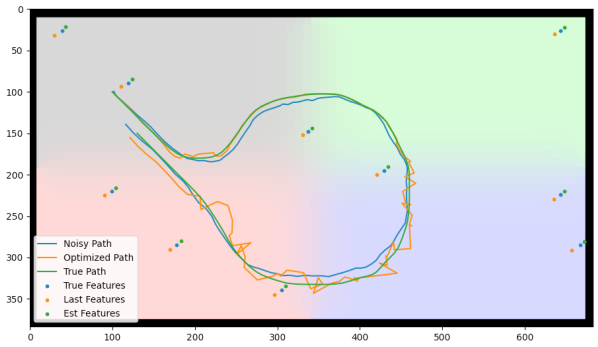

Fig. 15. Gradient noise and circular path using equal weighting.

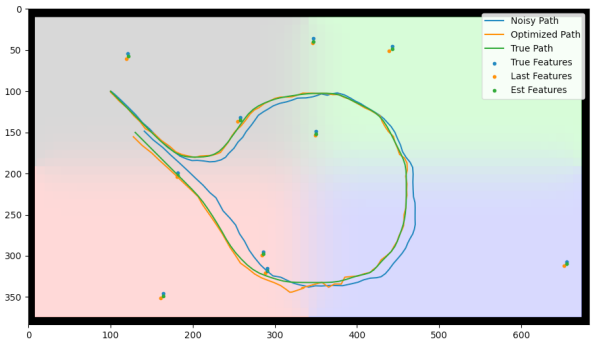

Fig. 16. Gradient noise and circular path using dynamic weighting (sliding window).

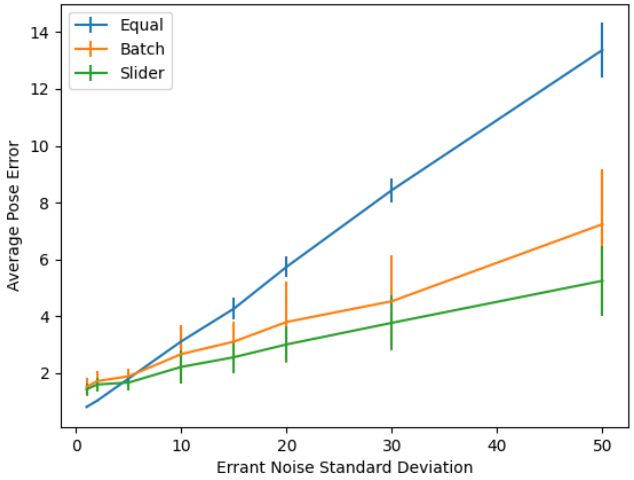

Fig. 17. Error of equal weighting, dynamic weighting (batch), and dynamic weighting (sliding window).

V. Conclusion and Future Work

We present a novel method for estimating sensor covariance for multi-sensor localization. Preliminary testing using a low-fidelity simulator shows promising results for identifying poor sensor behavior. Without assuming any distribution in the sensors, the algorithm effectively corrects poor sensor readings in an environment with drastically changing sensor reliability. The only regions of poor performance are when the sensors experience an extreme sudden change in performance. However, the errors from this poor performance are fairly localized if a global position sensor is used.

Future work for this project would involve testing the algorithm with real world data. An ideal data set would have True position and/or GPS, odometry, and an indirect odometry sensor such as laser scans or visual odometry. The goal of testing on real world data would be to verify the algorithm produces similar results when the nominal sensor covariance is accurate throughout a scenario. The second goal is to verify the system corrects poor sensor measurements if the sensor distribution varies through the optimization. If open source datasets do not have segments of sensor unreliability, they would need to be modified to introduce extra noise to the data to further evaluate the algorithm. Candidate data sets for future testing would include the KITTI or Victoria Park datasets.

The other thrust of future work is to improve the response to abrupt shifts in sensor data quality. The sliding window method given by Eqn. 9 gives better results than using a fixed optimization over n data points. However, this aspect of applying the covariance estimation could likely be tuned to further reduce the divergent behavior on sharp boundaries. Further testing in the low-fidelity simulator could also implement more realistic non-linear sensor and motion models. These may include a non-linear range and bearing feature sensor or an ackermann steering odometry model.

References

[1] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. Robust map optimization using dynamic covariance scaling. In *2013 IEEE International Conference on Robotics and Automation*, pages 62–69, 2013. doi: 10.1109/ICRA.2013.6630557.

[2] Pratik Agarwal, Giorgio Grisetti, Gian Diego Tipaldi, Luciano Spinello, Wolfram Burgard, and Cyrill Stachniss. Experimental analysis of dynamic covariance scaling for robust map optimization under bad initial estimates. pages 3626–3631, 05 2014. doi: 10.1109/ICRA.2014.6907383.

[3] Nicholas Carlevaris-Bianco. Long-term simultaneous localization and mapping in dynamic environments. 01 2014.

[4] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017. ISSN 1935-8253. doi: 10.1561/2300000043. URL http://dx.doi.org/10.1561/2300000043.

[5] H. Hu and G. Kantor. Parametric covariance prediction for heteroscedastic noise. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3057, 2015. doi: 10.1109/IROS.2015.7353798.

[6] K. Liu, K. Ok, W. Vega-Brown, and N. Roy. Deep inference for covariance estimation: Learning gaussian noise models for state estimation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1436–1443, 2018. doi: 10.1109/ICRA.2018.8461047.

[7] Edwin Olson and Pratik Agarwal. Inference on networks of mixtures for robust robot mapping. *The International Journal of Robotics Research*, 32(7):826–840, 2013. doi: 10.1177/0278364913479413. URL https://doi.org/10.1177/0278364913479413.

[8] T. Pfeifer, P. Weissig, S. Lange, and P. Protzel. Robust factor graph optimization - a comparison for sensor fusion applications. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2016. doi: 10.1109/ETFA.2016.7733598.

[9] N. Sünderhauf and P. Protzel. Switchable constraints for robust pose graph slam. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1879–1884, 2012. doi: 10.1109/IROS.2012.6385590.

[10] W. Vega-Brown, A. Bachrach, A. Bry, J. Kelly, and N. Roy. Cello: A fast algorithm for covariance estimation. In *2013 IEEE International Conference on Robotics and Automation*, pages 3160–3167, 2013. doi: 10.1109/ICRA.2013.6631017.

[11] Ryan Watson, Jason Gross, Clark Taylor, and Robert Leishman. Batch measurement error covariance estimation for robust localization. 09 2018. doi: 10.33012/2018.15974.