Integrity-Based Fault Detection for GNSS Navigation Safety

Siddharth Tanwar
tanwar@stanford.edu

Tara Yasmin Mina
tymina@stanford.edu

1 Introduction

With the currently expanding industry for self-driving cars and autonomous vehicles, we are experiencing the birth of a new era of day-to-day transportation. Most of these autonomous platforms require the use of Global Navigation Satellites and Systems (GNSS) for localization in a global reference frame; however, in urban environments, corrupted GNSS measurements can lead to localization errors, due to signal reflections and poor satellite geometry. For life-critical navigation applications, it is essential to guarantee a paramount level of safety or **navigation integrity**. Not only must the agent maintain an accurate position solution, but the errors must be bounded within a **protection level**, with a high probabilistic guarantee.

In this work, we aim to classify a set of received GNSS measurements as `safe` or `unsafe` for navigation, given a user-specified probabilistic requirements of safety. The feature vector consists of the received GNSS measurements, including the computed measurement residuals as defined in Sec. 2.3 and the received signal-to-noise ratio (SNR) for each satellite in view. We develop and compare various models from two powerful classification architectures: Support Vector Machines (SVMs) and neural networks and test our classifiers in a simulated urban environment. Code is available at our github repository.

1.1 Related work

In the field of controls, verification means providing a guarantee of safety for a system operating in an uncertain environment, under all possible operating conditions. Verification is usually performed using either an analytical proof-based aproach or a data-driven approach. Proof-based approaches [1][2][3] provide theoretically provable guarantees; however, these methods have restrictive assumptions and are often difficult to construct. Data-driven approaches, on the other hand, rely on simulations to establish statistical guarantees [4][5][6]. Quindlen et al. [6] use an SVM from simulated training data to separate perturbations to the system parameters (such as aircraft weight) into `safe` and `unsafe` subsets. While we seek to perform a similar task, the key differences are: (1) our nonlinear system is not deterministic, as assumed in [6] and (2) we define the safety criteria using probabilistic bounding of the navigation position errors as opposed to metric temporal logic (MTP).

In the field of GNSS, the task of identifying GNSS measurements as `unsafe` is called Integrity Monitoring (IM). IM is frequently performed using analytical methods [7][8][9]. These methods assume a rigid structure for the noise characteristics of the received measurements and identify thresholds for classification by hand-tuning these parameters. A recent work by Chiou et al. [10], uses bag of word features and trains a random forest model to classify GNSS measurements as "excellent," "good," and "poor" ; however, the authors do not provide a detailed paper of their work in the proceedings. To the best of the author's knowledge, apart from the work by Chiou et al., data-driven approaches to perform classification have not been explored.

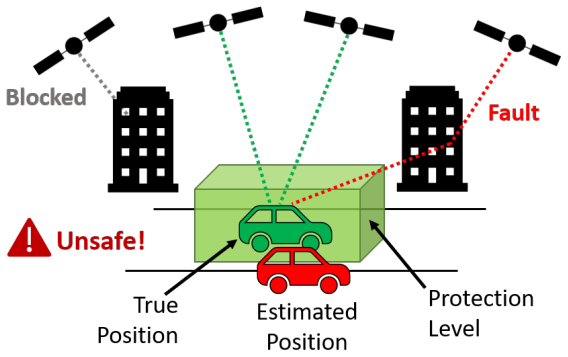

Figure 1: Illustration of an integrity fault. The GNSS measurements are unsafe for navigation since the resulting estimated position leads to a violation of the protection level about the true position, with probability ϵ.

2 Problem Formulation

2.1 GNSS Integrity Fault

Given the estimated position state $\hat{z}$ and the true position z, we define an *integrity fault* as in Eq. (1).

$$P(\hat{z} \notin PL(z)) \geq \epsilon \tag{1}$$

where $PL(z)$ defines the *protection level* about the true position, or the region of space which we seek our estimate x to lie within with high probability, and ϵ corresponds to the probabilistic threshold for which we are willing to tolerate a violation of the protection level.

In this work, the protection level PL corresponds to a bounding cuboid centered about the true position z, as illustrated in Fig. 1. We utilize Eq. (1) to detect the presence of an integrity fault and to define our true labels. In particular, if the expression is `true`, this indicates a fault has occurred and the set of GNSS measurements is labeled as **unsafe** to use for navigation. Otherwise, if the expression is `false`, the GNSS measurements are **safe**.

2.2 GNSS Measurement Corruption

An integrity fault may occur if the received GNSS measurements have been corrupted by the surrounding environment. Urban environments are especially challenging for satellite-based navigation due to the presence of large buildings, which can *block* or *reflect* the satellite signals and potentially induce large errors in the position estimate. When the satellite k is in direct line of sight, we model the received range measurement ρ_k as being corrupted by zero-mean, white Gaussian error ϵ:

$$\begin{aligned} \rho_k &= r_k + \epsilon \\ \rho_k &\sim \mathcal{N}\left(r_k, \sigma^2\right) \end{aligned} \tag{2}$$

where r_k denotes the true range from satellite k.

When the signal encounters a reflection, we model the range measurement as having an additional bias b:

$$\begin{aligned} \rho_k &= r_k + b + \epsilon \\ \rho_k &\sim \mathcal{N}\left(r_k + b, \sigma^2\right) \end{aligned} \tag{3}$$

2.3 Feature Vector

For each visible satellite k, the agent measures an estimated range from the satellite ρ_k and the signal-to-noise ratio (SNR) γ_k from the received signal. Since the agent continuously maintains its own position state estimate $\hat{z}$ and receives information about the satellite position z_k, the agent computes a corresponding range residual $\tilde{r}_k$:

$$\tilde{r}_k = \rho_k - ||z_k - \hat{z}|| \tag{4}$$

where $||\cdot||$ represents the Euclidean distance. Thus, our feature vector x is a concatenation of the range residuals and received SNR from each satellite:

$$x = [\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_N, \gamma_1, \gamma_2, \ldots, \gamma_N] \tag{5}$$

where N represents the number of satellites in view.

3 Simulation Environment

In this work, we developed a simulation environment for experimentation, in order to have access to the true agent position to provide accurate labels as well as to easily generate a large data sets from this environment.

3.1 Urban Environment

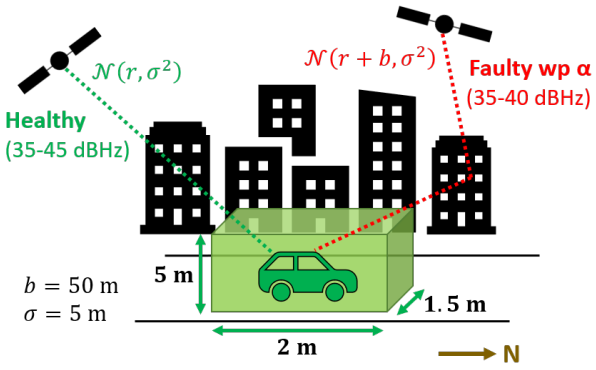

Figure 2: Agent trajectory (northward) with satellite measurement distributions in simulation environment

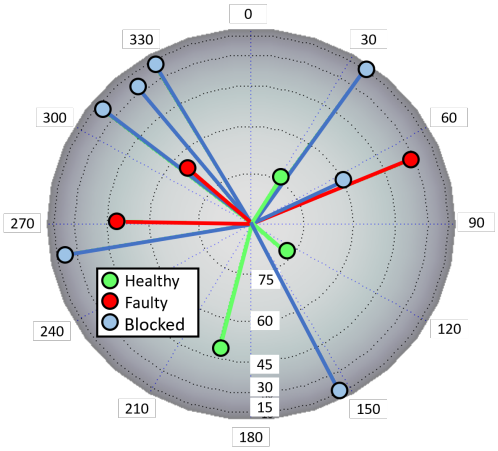

Figure 3: Sky plot of visible satellites. Green satellite are healthy due to the decreased presence of buildings in the along-street, North-South direction. Red satellites undergo signal reflection with probability 0.2, 0.1, and 0.5, from West to East. Blue satellites are blocked.

In our environment, we model the agent moving in the North direction in an urban scenario. The agent maintains its position state estimate using an Extended Kalman Filter (EKF) [11]. We use a real GPS satellite geometry, with satellite positions obtained from the Continuously Operating Reference System (CORS) [12], as maintained by the U.S. National Geodetic Survey (NGS). The satellite geometry is depicted in the sky plot in Fig. 3, where the clockwise-increasing angles correspond to degrees azimuth, (0° azimuth for North) and the concentric circles correspond to elevation angles (90° elevation for zenith).

With our urban model depicted in Fig. 2, all low-elevation satellites are blocked by buildings as shown in Fig. 3. Furthermore, the presence of urban buildings

on either side of the North-South street cause satellites along this direction to more frequently encounter signal reflections and have induced biases. These satellites are marked in red in Fig. 3.

3.2 Data Generation

From our simulation environment, we draw samples to generate testing and training data. As shown in Fig. 4, the complete environment incorporates our measurement model for generating healthy and biased measurements as defined in Eqs. (2) and (3), respectively, the agent state estimator as an EKF, and the integrity monitoring definition from Eq. (1) to provide the final true labels.

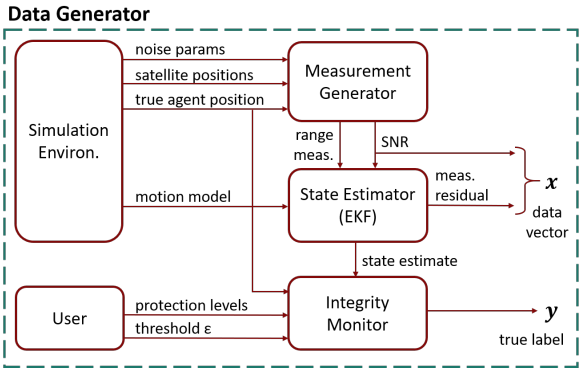

Figure 4: High-level architecture of simulation environment for generating data points

Drawing sample trajectories and measurements from our data generator module in Fig. 4, we created a data set of 10^5 training examples, 10^4 validation examples, and 10^4 test examples. The simulation parameters are defined in Tab. 1. The ranges for the received SNR are between $35-40$ dBHz if a satellite measurement undergoes a reflection and between $35-45$ dBHz otherwise.

4 Methods

We tested two machine learning architectures: Support Vector Machines (SVMs) and Neural Networks.

4.1 Support Vector Machine (SVMs)

SVMs solve the optimization problem in Eq. (6), which is a regularized reformulation of an optimal margin classifier to allow for misclassification errors, but with a penalty denoted by the summation of ζ_i term.

$$\min_{w,b,\zeta} \left(\frac{1}{2} w^\top w + C(\sum_{i=1}^{n} \zeta_i) \right)$$
$$\text{subj. to } y_i(w^\top \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, 2, .., n \qquad (6)$$

Parameter	Description	Value
ϵ	probability threshold	10^{-5}
PL (E,N,U)	protection level	$[1.5, 2.0, 5.0]$ m
b	induced range bias	50.0 m
σ_r	ranging std dev.	5.0 m

Table 1: Simulation parameters

To solve the optimization problem, one can convert it to its dual form and use the *kernel trick* to evade explicitly computing the feature map $\phi(x_i)$. We tested both a *linear* kernel, i.e. $K(x_i, x_j) = x_i^\top x_j$, and a Radial Basis Function (RBF) kernel, i.e. $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$. The linear kernel is useful if the data is separable by hyperplanes, whereas the RBF kernel transforms the input into an infinite-dimensional feature space and forms nonlinear separating hyperplanes in the original input feature space.

4.2 Neural Networks

A standard neural network consists of fully connected layers of neurons, where each neuron performs a nonlinear operation on an affine combination of the inputs. We tested with 90 neural network architectures in total, considering 1 and 2 hidden layers with the number of nodes in each layer from the set $\{2, 4, 6, 8, 12, 16, 32, 64, 128\}$. We chose ReLU activation function for the hidden layers, i.e. $g(z) = z\mathbb{1}\{z \geq 0\}$, and sigmoid activation for the output layer, i.e. $\sigma(z) = \frac{1}{1+\exp(-z)}$, to ensure the output lies in the interval $[0, 1]$. We then use the output for binary classification by evaluating $\mathbb{1}\{\hat{y} \geq 0.5\}$. To train the network, we used a binary cross entropy loss function with $L2$ regularization as in Eq. (7).

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{B} \sum_{i=1}^{B} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right)$$
$$+ \lambda \left(\sum_{j=1}^{n_l} ||W^{[j]}||^2 \right) \qquad (7)$$

5 Experimental Results

5.1 Metrics of Interest

For our application, we must choose a model which meets the user-defined threshold for *recall*, or fraction of accurately detected integrity faults, if such a model exists. Meeting a threshold for recall is important for this safety application, since if a fault is present, we want to detect it with high probability. Thus, we are interesting in choosing a model with the highest *precision*, or fraction of accurately detected healthy measurement sets, which meet a recall threshold of $\tau = 0.92$ for our application.

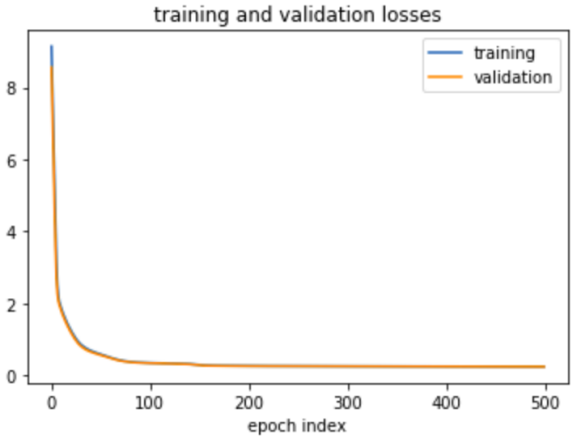

Figure 5: Training and validation losses for $\alpha = 10^{-2}$

λ	**10**	**1**	**0.1**	**0.01**	**0.001**
Acc	0.9174	0.9244	0.9034	0.9482	0.9631

Table 2: Accuracy for regularization coefficients λ

5.2 Tuning γ for SVM with RBF kernel

Using $C = 1$ for SVMs, we tested different γ values for the RBF kernel. From $\gamma \in \{10, 1, 0.1, 0.01, 0.001\}$, we perform a single-fold validation with a 90-10 split between training and validation data. We pick $\gamma = 10^{-3}$, since this gave us the best precision for recall values above the threshold $\tau = 0.92$, as described in Sec. 5.1.

5.3 Tuning α and Number of Epochs

For the neural networks, we use an *Adam optimizer* [13] with learning rate $\alpha = 0.01$ along with 500 epochs with a minibatch size of 10^4. We use this learning rate and number of epochs, since we observed steady, consistent convergence for this number of epochs as shown in Fig. 5.

5.4 Tuning λ for Neural Networks

We further experiment with the L2 regularization coefficient $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1\}$ for the neural networks. From Tab. 2, we observe the best accuracy on average for $\lambda = 0.01$.

5.5 Model Selection

For each trained models, we compute the **accuracy**, **precision**, and **recall** metrics in Table 3. We include the results from the SVMs with the two tested kernels as well as key neural network structures from the complete set of 90 architectures tested. We observe that the linear SVM has lower accuracy compared to other models on both training and test set, indicating that the linear SVM struggles to develop a linearly separating decision boundary for our classification problem. This is well expected, given the highly nonlinear measurement model and integrity metric.

Among the neural networks, we observe that the network with a single layer and only 2 neurons performs poorly, likely due to the lack of expressivity of this model. Furthermore, we observe $h_1 = 6, h_2 = 128$ to have the highest precision while remaining above the 0.92 recall threshold, thus we select this model for our application. On our test set, we observe the confusion matrix shown in Fig. 6 with the chosen model. Though the single-layer network with 6 neurons has slightly worse precision, this model also has comparable performance to the chosen model, but is much smaller with 85 parameters compared to 1103 parameters in the larger network. Tab. 4 shows the final performance of our chosen model on a separate test set of 10^4 points.

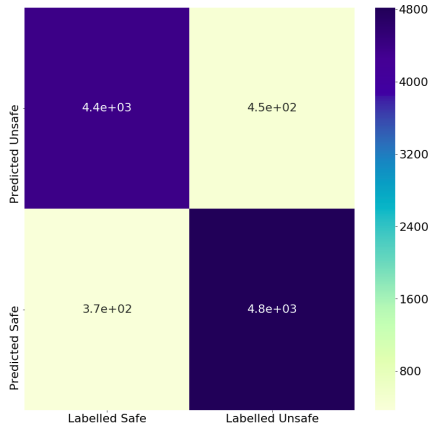

Figure 6: Confusion matrix on chosen model

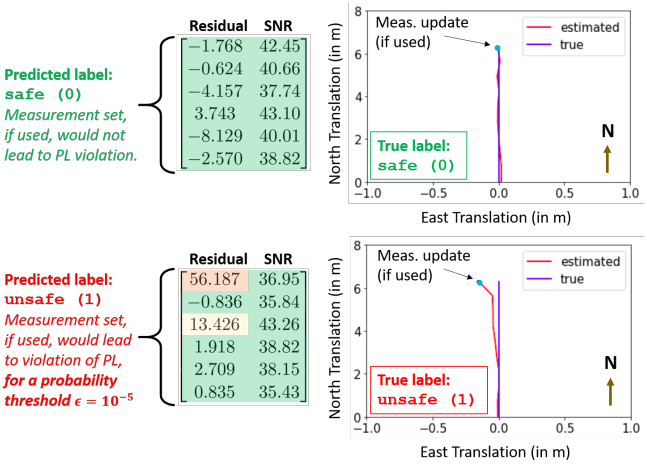

Figure 7: Example agent trajectories and received GNSS measurements. Using the chosen neural network model ($h_1 = 6, h_2 = 128$), the top example has correctly been detected as `safe` and bottom example has been correctly detected as `unsafe`, given the probability threshold for safety is $\epsilon = 10^{-5}$.

	Training (10^5 pts)			Validation (10^4 pts)		
Learning Model	**Acc**	**Rec**	**Prec**	**Acc**	**Rec**	**Prec**
SVM (`linear`)	0.7429	0.7577	0.7165	0.7414	0.7590	0.7160
SVM (`RBF`)	0.9243	0.9294	0.9127	0.9209	0.9258	0.9100
NN ($h_1 = 2$, 1 layer)	0.5597	0.9782	0.5205	0.5567	0.9759	0.5173
NN ($h_1 = 6$, 1 layer)	0.9231	0.9245	0.9143	0.9202	**0.9245**	**0.9097**
NN ($h_1 = 128$, 1 layer)	0.9235	0.9264	0.9135	0.9208	0.9256	0.9100
NN ($h_1, h_2 = 6, 128$)	0.9232	0.9230	0.9158	0.9202	**0.9218**	**0.9119**
NN ($h_1, h_2 = 128$)	0.9235	0.9256	0.9143	0.9204	0.9241	0.9104

Table 3: Training and validation results for SVMs tested and key neural network structures

	Test (Multiple Faults)			Test (Single Fault)		
Classifier	**Acc**	**Rec**	**Prec**	**Acc**	**Rec**	**Prec**
SVM (`RBF`)	0.9174	0.9244	0.9034	0.9482	0.9631	0.9313
NN ($h_1 = 6$, 1 layer)	0.9177	0.9253	0.9033	0.9478	0.9629	0.9307
NN ($h_1, h_2 = 6, 128$)	0.9185	0.9221	0.9073	0.9490	0.9627	0.9331
Residual RAIM[1]	-	-	-	0.9395	0.9431	0.9316

Table 4: Final results on testing data, which incorporates multiple faults, and comparison against RAIM, which can only handle single fault cases.

5.6 Qualitative Results

Fig. 7 shows two example trajectories using the chosen neural network model ($h_1 = 6, h_2 = 128$). In the top example trajectory, the model has correctly detected the measurement set to be `safe` according to our integrity monitor defined in Eq. (1). We notice in this example the agent position estimate remains consistent with the true agent position trajectory. In the bottom example trajectory, the chosen model has an correctly detected the measurement set to be `unsafe`. Indeed, we observe in this case that if the agent would accept the GNSS measurements for the position update, the estimated position would have significantly veered from the true position as shown by the green point. Though in this sampled trajectory the green point is not outside of the $1.5 \times 2 \times 5$ meter protection level, this still corresponds to an integrity fault, since the probability that the agent lies outside of the protection level exceeds the maximum threshold of $\epsilon = 10^{-5}$. Thus, the model accurately detects that this set of measurements is indeed unsafe for navigation.

5.7 Comparison to RAIM

We further compare a few of the best performing models to a standard integrity monitoring method called Receiver Autonomous Integrity Monitoring (RAIM)[7]. RAIM uses a test statistic based on central chi-squared test statistic for fault detection in GNSS satellites and requires hand-tuning to work. Because RAIM only handles single satellite faults, we generate test data with at most 1 faults to provide a fair comparison against RAIM. Tab. 4 shows that our models outperform RAIM in all 3 metrics: accuracy, precision, recall.

6 Conclusion

In this work, we train multiple machine learning architectures to perform classification on a set of received GNSS measurements as being `safe` or `unsafe` to use for navigation. We developed a simulation environment which models an urban scenario to provide access to the ground truth agent position and to generate labeled data for training and testing our classifier. We observe that our selected model performs better than existing integrity monitoring methods such as RAIM, with regards to both recall and precision, indicating both fewer missed detections of integrity faults as well as fewer false alarms. Furthermore, our model handles the presence of multiple faults, unlike RAIM. Due to the highly nonlinear decision boundary, we observe the neural networks performed better than the linear SVM. With the L2 regularization, the neural networks perform better than the RBF SVM at learning which combinations of measurements induce a fault given the satellite geometry, without overfitting to the training data.

7 Future Work

We plan to extend our feature vector to incorporate a sequence of measurements. Since GNSS measurements are temporally and spatially correlated, we can leverage this inherent property to improve detection of faults. We would also like to make our model generalize better to variations in satellite geometry to increase the scope of application to real-world situations.

Contributions

Both of us worked on designing the class structures, as well as the formulation of our simulation and learning architecture, with the precise integrity metric, motion model, and state estimation filter.

Siddharth conducted a literature survey and developed the code for the Extended Kalman Filter estimator as well as the simulation environment, including the agent and satellite trajectories.

Tara developed the code for the SVM classifier, including the integrity metric to define true labels, the measurement sampler and conversion to pseudoranges, as well as the training and testing processes.

Both of us also worked together to write our `main` script, as well as to debug the combined code and thoroughly verify the each component written.

References

[1] J. Moore and R. Tedrake, "Control synthesis and verification for a perching UAV using LQR-trees," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 3707–3714.

[2] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technische Universität München, 2010.

[3] U. Topcu, *Quantitative local analysis of nonlinear systems*. University of California, Berkeley Berkeley, CA, 2008.

[4] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Arechiga, "Simulation-guided lyapunov analysis for hybrid dynamical systems," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 2014, pp. 133–142.

[5] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-guided approaches for verification of automotive powertrain control systems," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 4086–4095.

[6] J. F. Quindlen, U. Topcu, G. Chowdhary, and J. P. How, "Active sampling-based binary verification of dynamical systems," in *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 1107.

[7] T. Walter and P. Enge, "Weighted raim for precision approach," in *PROCEEDINGS OF ION GPS*, vol. 8. Institute of Navigation, 1995, pp. 1995–2004.

[8] M. Joerger, F.-C. Chan, and B. Pervan, "Solution separation versus residual-based RAIM," *NAVIGATION: Journal of the Institute of Navigation*, vol. 61, no. 4, pp. 273–291, 2014.

[9] S. Bhamidipati and G. X. Gao, "Slam-based integrity monitoring using gps and fish-eye camera," *arXiv preprint arXiv:1910.02165*, 2019.

[10] T.-Y. Chiou, T.-E. Tseng, and A.-L. Tao, "Performance of machine learning models in determining the GNSS position usage for a loosely coupled GNSS/IMU system," *32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019), Miami, Florida*, 2019.

[11] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT press, 2005.

[12] National Geodetic Survey (NGS), "Continuously Operating Reference System (CORS)," 2017, http://www.ngs.noaa.gov/.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.