A machine learning approach for UAV-based ground target geolocation in aerial images.

Project-I (EC4D001) Report submitted in partial fulfillment for the award of degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Anurag Paul
20EC01045

Under the supervision of

Dr. Niladri Bihari Puhan

SCHOOL OF ELECTRICAL SCIENCES

INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR

Acknowledgement

I express my deep sense of gratitude and sincere regards to my supervisor Dr. Niladri Bihari Puhan. The friendly discussion helped immensely in selecting this topic and the generous encouragement throughout my dissertation work helped in continuing this project work. Sir has immensely assisted in providing all opportunities and facilities for the project work. His critical reviews aided in implementing the topic. I am thankful to sir for helping in this work.

I am indebted to my parents who have inspired us to face all the challenges and win all the hurdles in life. Finally, I would like to thank all those who directly or indirectly guided me during my work since inception.

Abstract

This report proposes geolocating a ground target with an unmanned aerial vehicle (UAV) in a simulation of urban environment. UAVs such as quadcopter and multirotor aircrafts have been widely used in recent years. Examples include information and intelligent warfare. For geolocating multiple targets of interest on the ground from an aerial platform, object detection with the trained network will be applied to the target pixel locations from a drone footage.

Within academia and industry, there has been a need for expansive simulation frameworks that include model-based simulation of sensors, mobile vehicles, and the environment around them. This report makes use of a quadcopter aircrafts simulation environment based on open-source AirSim framework. It is a popular community-built system that fulfills some of those needs. However, the framework required adding systems to serve some complex industrial applications, including designing and testing new sensor modalities, Simultaneous Localization And Mapping (SLAM), autonomous navigation algorithms, and transfer learning with machine learning models as required by a user.

This article describes the framework of this work and the communication way between different parts. Furthermore, we show the various applications and use cases the framework can serve.

Contents

List of Figures

Chapter 1

Introduction

With the continuous progress of technology, quadcopter aircrafts are widely used in security, rescue, plant protection, transportation, and other fields. Currently, there are two main test methods for quadcopter. The first way is to use a real quadcopter for the test. The other way is to use the current simulation environment for the test. Firstly, using an actual quadcopter for the test is a direct way to verify the aircraft. Undoubtedly, the effect is the most real. However, actual flights are inconvenient for the quadcopter test.

The main problems are as follows. One is the safety issue, an unverified quadcopter is directly used for the test can easily injure the people. Secondly, untested aircrafts are extremely easy to crash, resulting in great economic losses. The third is an efficiency problem. A real flight needs to consider many issues, such as assembly, maintenance, battery and so on.

These factors will greatly affect flight efficiency and extremely extend the development cycle. Therefore, many researchers use simulation environments for the quadcopter test. At present, many simulation platforms can be used for tests such as Gazebo [1] [figure 1.1], jMavSim [2] [figure 1.2] and so on. Among them, Gazebo is a feature-rich simulation platform, which is widely used in the robots, cars, and drone simulation.

However, some limitations, like inconvenient construction, are not real enough. Which is extremely important for the quadcopter simulation tests. Usually, the real environment is extremely complex and has a wide variety of objects. If the simulation environment is inconducive for building large complex scenes, it will greatly extend the development time. Next, if the simulation environment scenes are not real enough, it will have much negative effect on the experiment, especially in some visual tests.

Therefore, aiming at these shortcomings above, this thesis proposes a new simulation environment based on AirSim [3]. AirSim provides us with a more realistic simulation environment and rich data interface as shown in figure 1.3. This enhances the authenticity of the simulation. Compared to figure 1.1 and figure 1.2, AirSim's images are more realistic. Unreal Engine or many GitHub repositories have some ready-made scenes in application store that contain blocks, forest, snow mountains, and so on.

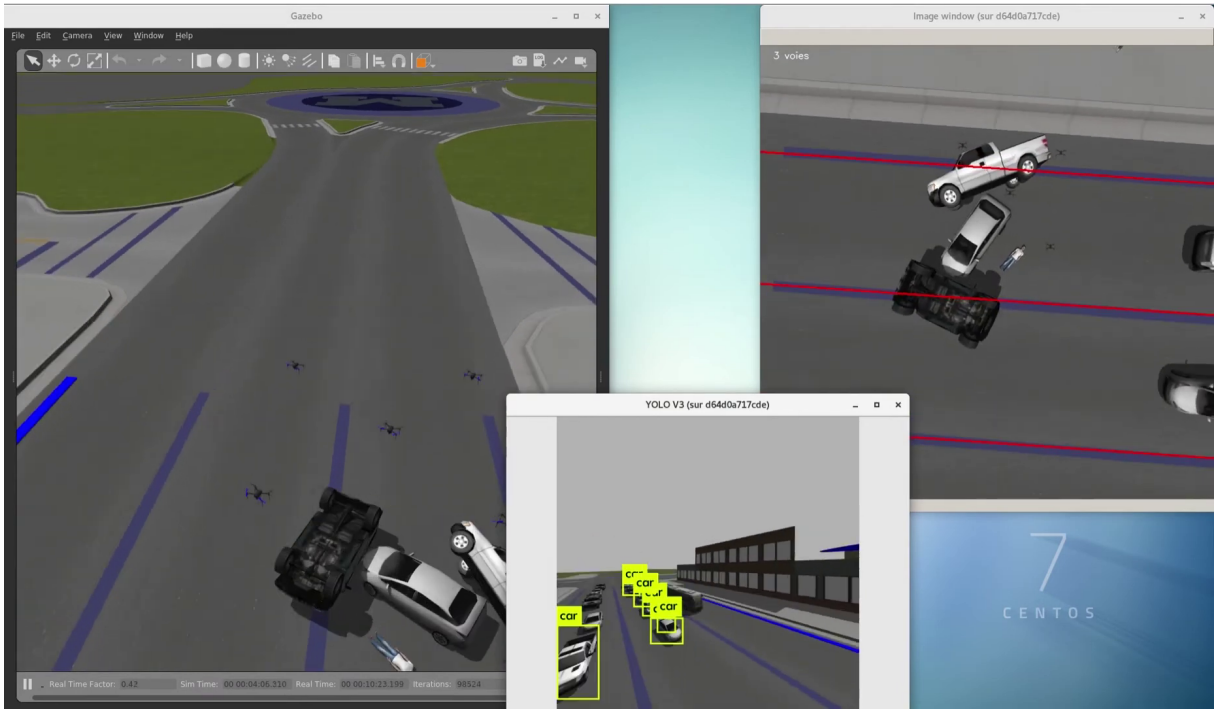


Figure 1.1: Gazebo drone fleet simulation.

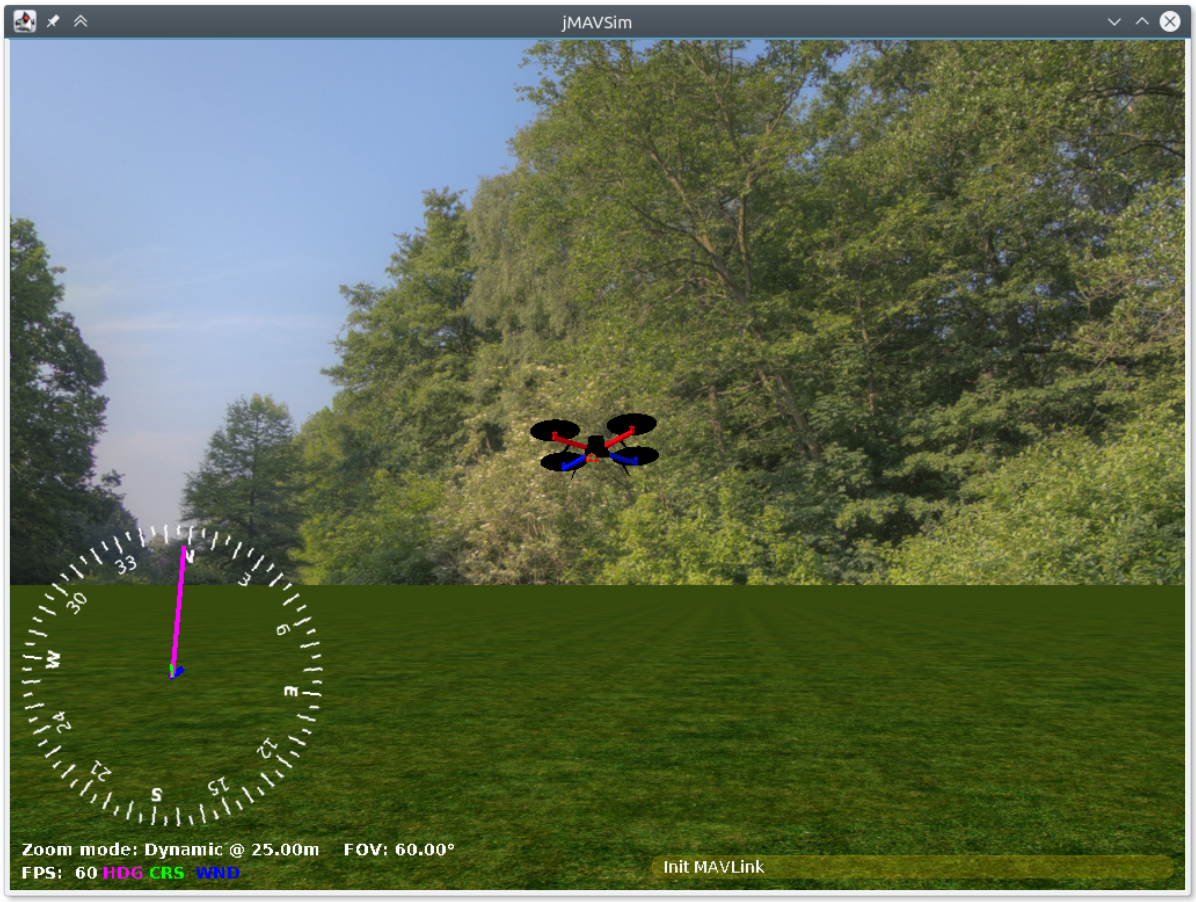


Figure 1.2: jMavSim, a simple multirotor simulator with MAVLink protocol support.

Figure 1.3: AirSimNH environment.

Chapter 2

Applications

Modern-day industrial applications, however, often require some form of flexibility in simulation. These applications include deciding ideal-sensor placement, validating sensor design parameters, testing autonomous robotic algorithms, generating training datasets for neural networks, and much more. Therefore, having a single, modular, and open-source framework allows for tackling the many requirements of such applications across several academic or industrial projects and takes less time to develop.

Commercial frameworks seem to provide the necessary tools to make these applications work. Most often, these commercial frameworks are designed for ADAS (Advanced Driver-Assistance System)/AD (Autonomous Driving) simulation that can do software-in-the-loop and even hardware-in-the-loop simulation of vehicles within detailed environments. However, to be more accessible to the general public, smaller companies, and academic researchers, opensource frameworks are desired, which can also be used for different application types than ADAS/AD.

Quadcopter can be used to set all kinds of sensors, and has low cost and high flexibility, thus can be applied to a variety of different tasks, package across target tracking, disaster rescue, crop monitoring, etc. Quadcopter can quickly spread, a big reason is that the development of the open source flight control, in the complicated products, four rotor quadcopter with its advantage of simple structure, convenient use and low cost, first came to the attention of the public.

Unmanned aerial vehicle learning in a virtual environment can greatly reduce the loss of the body and increase the speed of training. The virtual unmanned aerial vehicle needs to be transplanted to a real unmanned aerial vehicle after the virtual environment training is completed.

Chapter 3

Methodology

3.1 Simulation Framework Overview

The AirSim framework was designed for AI research and experimentation. While it focused on unmanned aerial vehicles, it was designed to be modular to accommodate new types of mobile platforms, sensors, and environments. In July 2022, Microsoft announced the end of its support to the original AirSim research project: it would evolve into a commercial closed-source version. This type of event further highlights the need for long-term supported, community-driven simulation frameworks.

The interaction between the AirSim Python library and the underlying Unreal Engine was used. AirSim allows recording of trajectories for vehicles; because the simulation is running in real-time, the data capturing through the API has to run in real-time, which was achieved. This enables the generation of datasets at a much faster speed than in real life because multiple simulations can run at the same time.

3.2 Virtual Environments

AirSim already provides basic weather simulation; hence the performance degradation of a sensor such as camera in rainy conditions regarding the intensity and range measurements can be tested. This influences this last form of quality attenuation. The proposed method is divided in to virtual and real environments. First we train the unmanned aerial vehicle to complete a specific task in a virtual environment. We can also use these sensors to get information about GPS and distance from the ground. Actions: There are six of them. The X-axis, Y-axis and z-axis and their positive and negative movement.

3.3 Realistic Environment

In order to transfer the model training in virtual environment to the real environment. We may use a programmable or manual unmanned aerial vehicle that can operate the

UAV through commands. The UAV connect with server. Because the virtual world cannot fully simulate the real world. Unmanned aerial vehicle will do a small amount of training in the real world so that the model can deal with the tasks in reality more effectively.

Haversine distance

$$D(x,y) = 2\arcsin[\sqrt{\sin^2((x_{lat} - y_{lat})/2) + \cos(x_{lat})\cos(y_{lat})\ sin^2((x_{lon} - y_{lon})/2)}]$$

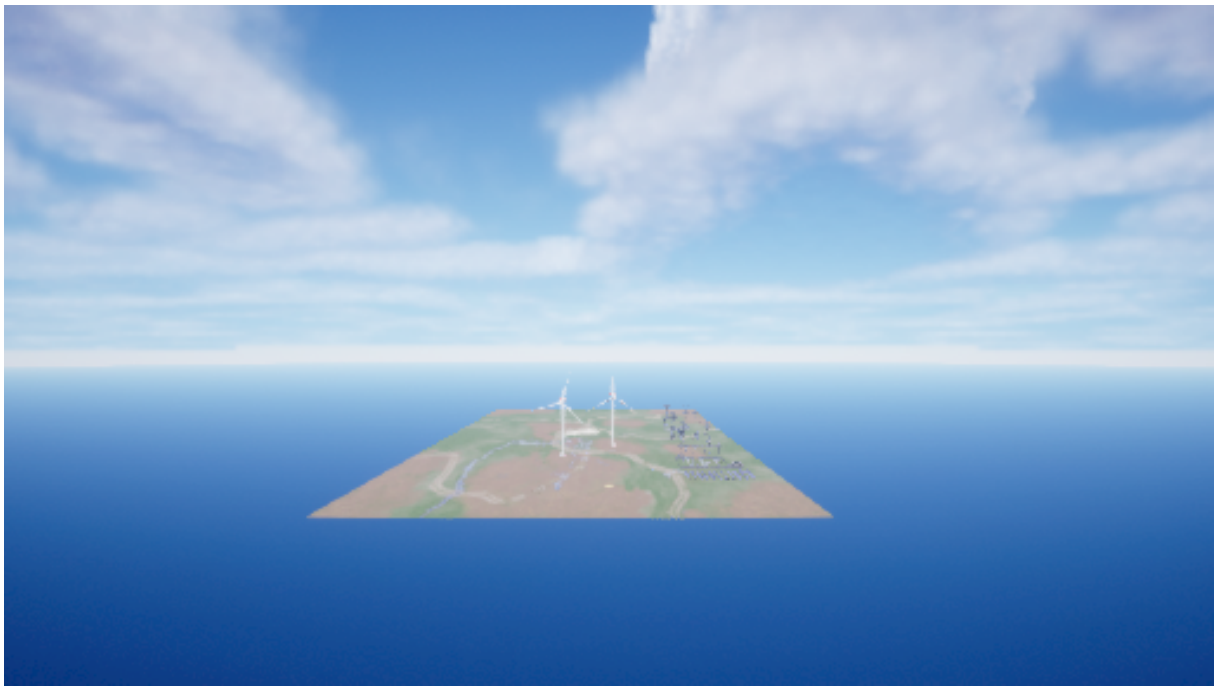


Figure 3.1: 1 of 100 captured Images.

Table 3.1: Targets and their pixel coordinates in image 1 [Fig. 3.1]

Target number	PixelX	PixelY
1	236	190
2	190	205
3	349	215
4	291	171
5	257	186
6	309	193
7	274	196
8	246	189
9	208	171
10	128	216

Data: 1000

In this work, machine learning approached in static and video target geolocation. UAV data: The public, UAV-borne dataset for target geolocation is mainly visible data, and the image size is 512 × 288. There are only one multiple source data containing ground targets dataset.

3.4 Challenges

Target geolocation and tracking tasks in the UAV remote sensing video face many challenges, such as image degradation, uneven target intensity, small object size, and background complexity.

- ***Image degradation*** **[4].** The load that a mini-UAV platform carries is strictly limited in terms of weight, volume, and power. Rapid movement changes in the external environment (such as light, cloud, fog, rain, etc.) cause aerial images to be fuzzy and noisy, which inevitably leads to image degradation . High-speed flight or camera rotation also increases the complexity of object detection. Thus, it is necessary to carry out image pre-processing, such as noise reduction, camera distortion correction, etc., to ensure the effectiveness of the model.

- ***Uneven target intensity.*** The image acquisition equipment of a UAV typically uses a large aperture, fixed focal, and wide-angle lens. In addition, flexible camera movement results in an uneven density of captured objects. Some of them are densely arranged and overlap many times, so that it is easy to repeat detection. This could be infered from the image dataset target pixels with its corresponding pixel locations. Some are sparse and unevenly distributed, so that it is prone to missed detection.

- ***Target size.*** UAV remote sensing images can be acquired at different altitudes, yielding photographs containing any size of ground targets. This challenges the classical DL-based method. Ground objects in UAV remote sensing are primarily shown as images with an area smaller than 32 × 32 pixels.

- ***Real-time.*** Target geolocation or tracking in a video obtained by a drone needs to quickly and accurately locate moving ground objects, so real-time processing performance is highly essential.

In table 3.2 the horizontal line separates the inputs and outputs of the full geolocation system. Altitude, Latitude, Longitude are of the drone. PixelX, PixelY are the pixel location of the target in the captured image. Ylatitude, Ylongitude are the predicted geolocation of the target.

Kernel (Covariance) Function: In supervised learning, it is expected that the points with similar predictor values x_i, naturally have close response (target) values y_i.

Table 3.2: Small sample of acumulated data

Altitude	Latitude	Longitude	PixelX	PixelY	Ylatitude	Ylongitude
222.5351715	47.63749253	-122.1401649	309	193	47.64210198	-122.1387175
224.434082	47.63749253	-122.1401649	309	193	47.64210198	-122.1387175
224.4347076	47.63749253	-122.1401649	309	194	47.64210198	-122.1387175
224.7390747	47.63749253	-122.1401649	309	194	47.64210198	-122.1387175
225.4135895	47.63749253	-122.1401649	309	195	47.64210198	-122.1387175
226.0838318	47.63749253	-122.1401649	309	195	47.64210198	-122.1387175
226.480072	47.63749253	-122.1401649	309	195	47.64210198	-122.1387175
227.0464478	47.63749253	-122.1401649	309	196	47.64210198	-122.1387175
227.4352722	47.63749253	-122.1401649	309	196	47.64210198	-122.1387175
227.8648376	47.63749253	-122.1401649	309	196	47.64210198	-122.1387175
228.4628143	47.63749253	-122.1401649	309	196	47.64210198	-122.1387175
228.9729156	47.63749253	-122.1401649	309	196	47.64210198	-122.1387175
229.4390106	47.63749253	-122.1401649	309	197	47.64210198	-122.1387175
229.9884491	47.63749253	-122.1401649	309	197	47.64210198	-122.1387175
230.4102631	47.63749253	-122.1401649	309	197	47.64210198	-122.1387175

In Gaussian processes, the covariance function expresses this similarity[5]. It specifies the covariance between the two latent variables $f(x_i)$ and $f(x_j)$, where both x_i and x_j are $d \times 1$ vectors. In other words, it determines how the response at one point x_i is affected by responses at other points x_j, $i \neq j$, $i = 1, 2, ..., n$. The covariance function $k(x_i, x_j)$ can be defined by various kernel functions. It can be parameterized in terms of the kernel parameters in vector θ. Hence, it is possible to express the covariance function as $k(x_i, x_j|\theta)$.

For many standard kernel functions, the kernel parameters are based on the signal standard deviation σ_f and the characteristic length scale σ_l. The characteristic length scales briefly define how far apart the input values x_i can be for the response values to become uncorrelated. Both σ_f and σ_l need to be greater than 0, and this can be enforced by the unconstrained parametrization vector θ, such that $\theta_1 = \log \sigma_l$, $\theta_2 = \log \sigma_f$.

1. **squaredexponential (default):**, this is one of the most commonly used covariance functions and is the default option for fitrgp. The squared exponential kernel function is defined as

$$k(x_i, x_k|\theta) = \sigma_f^2 \exp\left(-\frac{1}{2}\frac{(x_i - x_j)^T(x_i - x_j)}{\sigma_l^2}\right)$$

   where σ_l is the characteristic length scale, and σ_f is the signal standard deviation.

2. **ardsquaredexponential:**, this covariance function is a squared exponential kernel with a separate length scale per predictor.

$$k(x_i, x_k|\theta) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{m=1}^{d}\frac{(x_{im} - x_{jm})^2}{\sigma_m^2}\right)$$

where σ_m is length scale for each predictor m, $m = 1, 2, ..., d$ implement automatic relevance determination (ARD)[6]. The unconstrained parametrization θ in this case is

$$\theta_m = \log \sigma_m for m = 1,\ 2, ..., d \text{ and}$$
$$\theta_{d+1} = \log \sigma_f.$$

FitMethod: Method to estimate parameters of GPR model.

1. **"none"**, No estimation. Use the initial parameter values as the known parameter values.

2. **"sr"**, subset of regressors approximation is a sparse method for parameter estimation. consists of replacing the kernel function $k(x, x_r|\theta)$ in the exact GPR method by its approximation $\hat{k}_{SR}(x, x_r|\theta, \mathcal{A})$, given the active set $\mathcal{A} \subset \mathcal{N} = 1, 2, ..., n$. For the exact GPR model, the expected prediction in GPR depends on the set of $\mathcal{N}$ functions $\mathcal{S}_\mathcal{N} = k(x, x_i|\theta)$, $i = 1, 2, ..., n$, where $N = 1, 2, ..., n$ is the set of indices of all observations, and n is the total number of observations. The idea is to approximate the span of these functions by a smaller set of functions, $\mathcal{S}_\mathcal{A}$, where $\mathcal{A} \subset \mathcal{N} = 1, 2, ..., n$ is the subset of indices of points selected to be in the active set. Consider $\mathcal{S}_\mathcal{A} = k(x, x_j|\theta)$, $j \in \theta$. The aim is to approximate the elements of as linear combinations of the elements of $\mathcal{S}_\mathcal{N}$.

Suppose the approximation to $k(x, x_r|\theta)$ using the functions in $\mathcal{S}_\mathcal{A}$ is as follows:

$$\hat{k}(x, x_r|\theta) = \sum_{j \in \mathcal{A}} c_{jr} k(x, x_j|\theta),$$

where $c_{jr} \in \mathbf{R}$ are the coefficients of the linear combination for approximating $k(x, x_r|\theta)$. Suppose C is the matrix that contains all the coefficients c_{jr}. Then, C, is a $|\mathcal{A}| \times n$ matrix such that $C(j, r) = c_{jr}$. The software finds the best approximation to the elements of $\mathcal{S}_\mathcal{N}$ using the active set $\mathcal{A} \subset \mathcal{N} = 1, 2, ..., n$ by minimizing the error function

$$E(\mathcal{A}, C) = \sum_{r=1}^{n} ||k(x, x_r|\theta) - \hat{k}(x, x_r|\theta)||^2_\mathcal{H}$$

where $\mathcal{H}$ is the Reproducing Kernel Hilbert Spaces (RKHS) associated with the kernel function k [5], [7].

PredictMethod: Method used to make predictions from a Gaussian process model given the parameters, specified as one of the following values.

1. **"exact" :** An instance of response y from a Gaussian process regression (GPR) model can be modeled as $P(y_i|f(x_i), x_i) \sim N(y_i|h(x_i)^T\beta + f(x_i), \sigma^2)$ Hence, making predictions for new data from a GPR model requires:

- Knowledge of the coefficient vector, β, of fixed basis functions.
- Ability to evaluate the covariance function $k(x, x'|\theta)$ for arbitrary x and x', given the kernel parameters or hyperparameters, θ.
- Knowledge of the noise variance σ^2 that appears in the density $P(y_i|f(x_i), x_i)$.

2. **"fic" :** fully independent conditional approximation [8] method for prediction. It is a way of systematically approximating the true GPR kernel function in a way that avoids the predictive variance problem of the SR approximation while still maintaining a valid Gaussian process. One can specify the FIC method for parameter estimation by using the 'FitMethod','fic' name-value pair argument in the call to fitrgp. For prediction using FIC, one can use the 'PredictMethod','fic' name-value pair argument in the call to fitrgp.

The FIC approximation to $k(x_i, x_j|\theta)$ for active set $\mathcal{A} \subset \mathcal{N} = \{1, 2, ..., n\}$ is given by:

$$\hat{k}_{FIC}(x_i, x_j|\theta, \mathcal{A}) = \hat{k}_{SR}(x_i, x_j|\theta, \mathcal{A}) + \delta_{ij}(k(x_i, x_j\theta) - \hat{k}_{SR}(x_i, x_j|\theta, \mathcal{A})),$$

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

That is, one needs first to estimate β, θ, and σ^2 from the data (X, y).

That is, the FIC approximation is equal to the SR approximation if $i \neq j$. For $i = j$, the software uses the exact kernel value rather than an approximation. Define an $n \times n$ diagonal matrix $\Omega(X|\theta, \mathcal{A})$ as follows:

$$[\Omega(X|\theta, \mathcal{A})]_{ij} = \delta_{ij}(k(x_i, x_j|\theta) - \hat{k}_{SR}(x_i, x_j|\theta, \mathcal{A}))$$

$$= \begin{cases} k(x_i, x_j|\theta) - \hat{k}_{SR}(x_i, x_j|\theta, \mathcal{A}) & \text{if} i = j, \\ 0 & \text{if} i \neq j. \end{cases}$$

Standardize: false or 0 (default), standardize the predictors in the training data. If one sets Standardize=1, then the software centers and scales each column of the predictor data by the column mean and standard deviation. The software does not standardize the data contained in the dummy variable columns generated for categorical predictors.

Sigma: std(y)/$\sqrt{2}$ (default), positive scalar value, initial value for noise standard deviation of the Gaussian process model. The training function parameterizes the noise standard deviation as the sum of SigmaLowerBound and e^{η}, where η is an unconstrained value. Therefore, Sigma must be larger than SigmaLowerBound by a small tolerance so that the function can initialize *eta* to a finite value. Otherwise, the function resets Sigma to a compatible value.

The tolerance is $1e-3$ when ConstantSigma is false (default) and 1e-6 otherwise. When ConstantSigma is true, the training function does not optimize the value of Sigma,

but instead uses the initial value throughout its computations. If the tolerance is not small enough relative to the scale of the response variable, one can scale up the response variable so that the tolerance value can be considered small for the response variable.

Chapter 4

Results

4.0.1 Cross validation

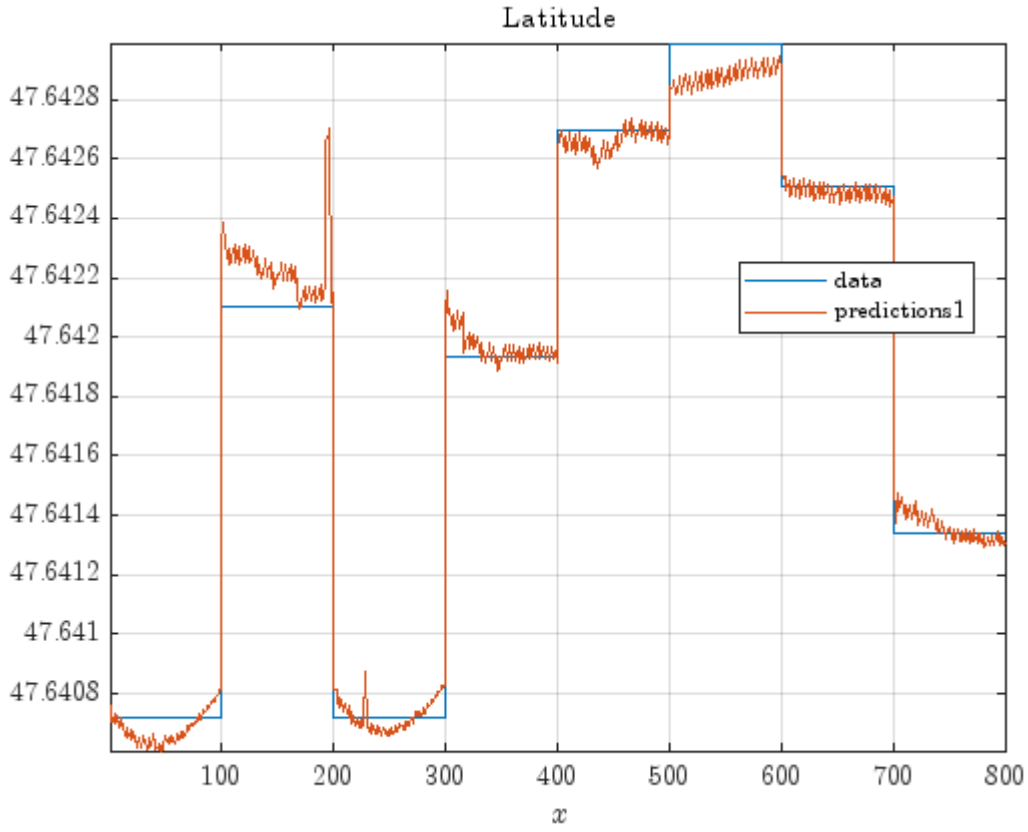


Figure 4.1: Testing with training data.

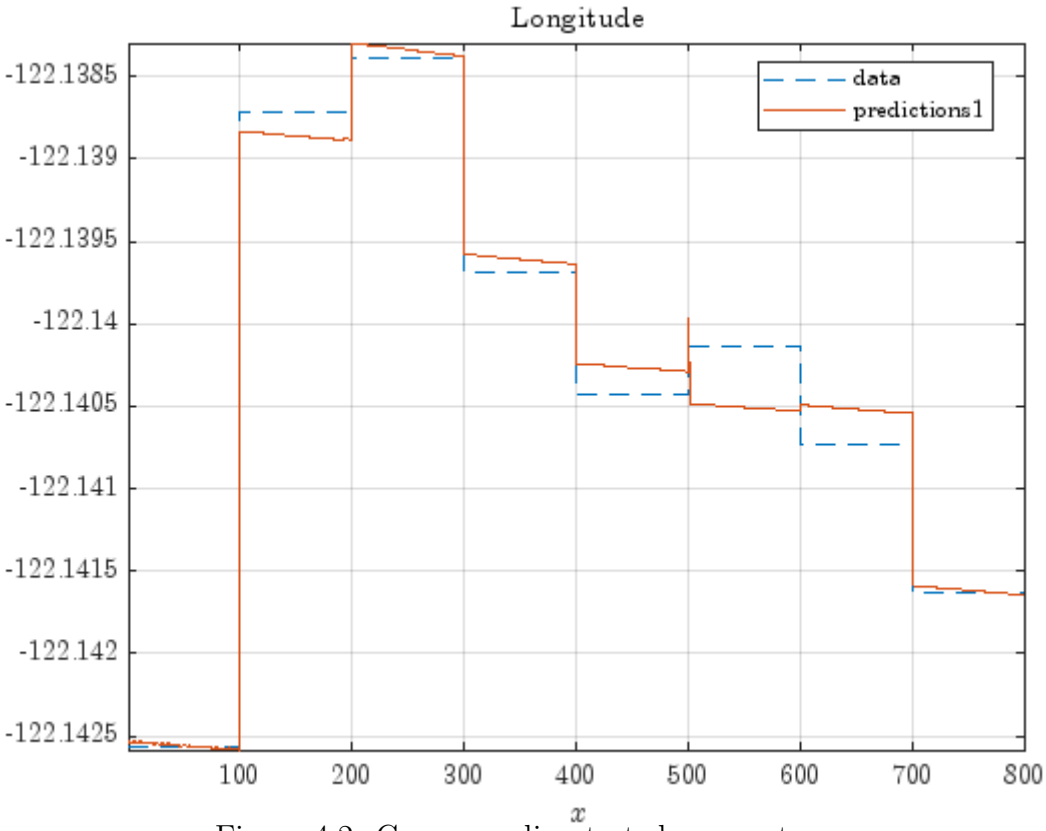


Figure 4.2: Corresponding tested parameter.

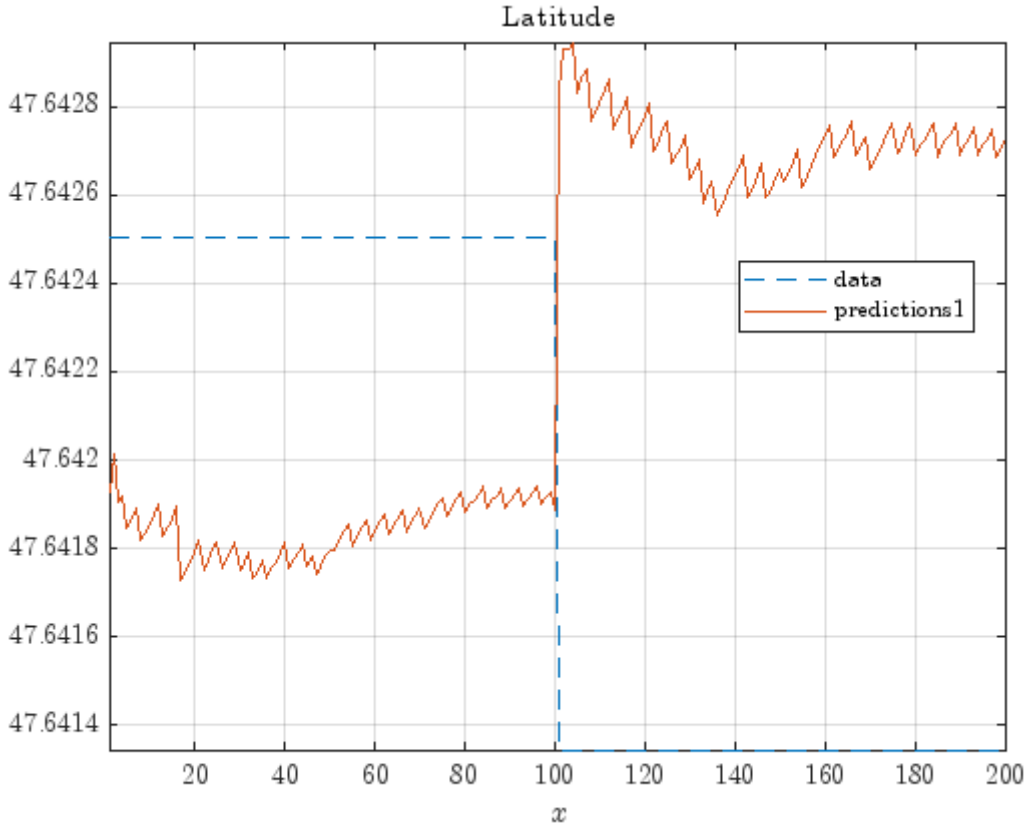


Figure 4.3: Cross validation: 1 of 5 parameter test.

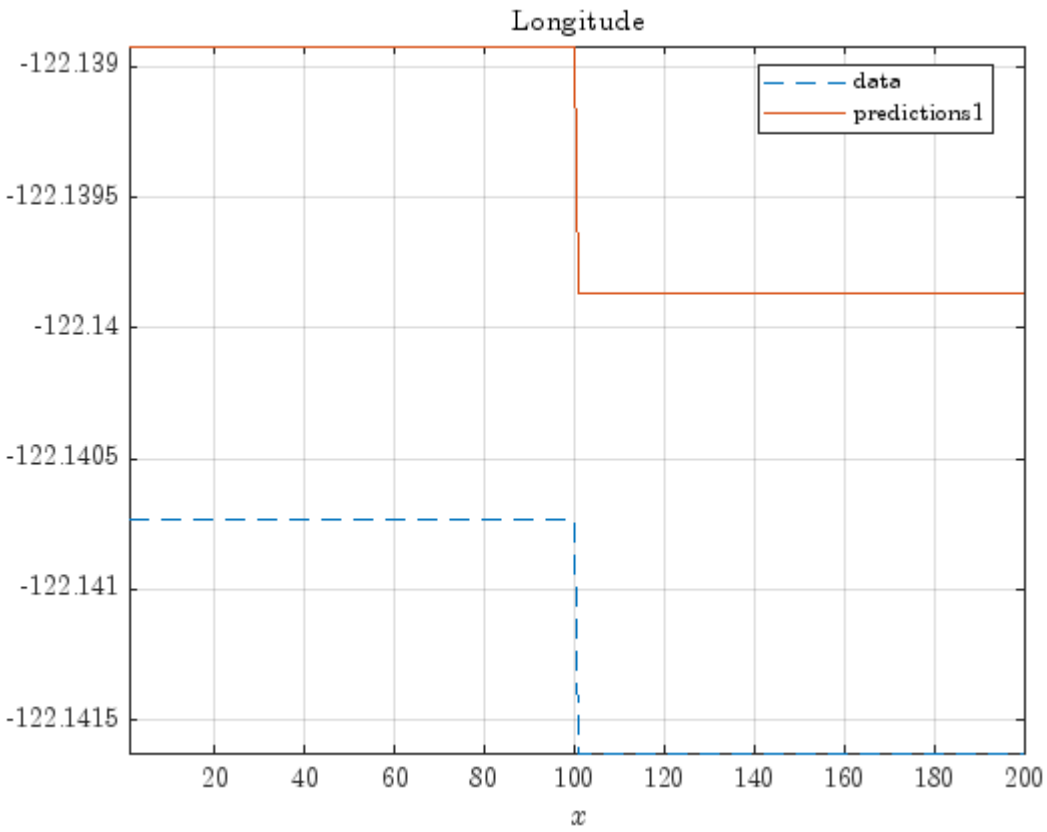
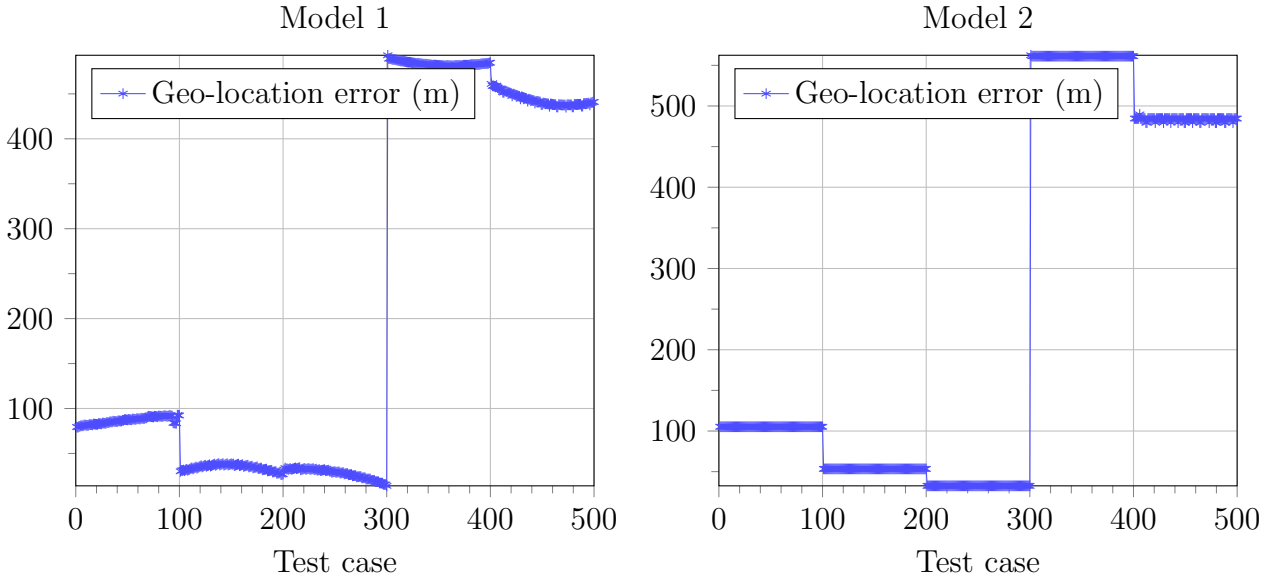


Figure 4.4: Corresponding Cross validation: 2:10 data set for testing.

4.0.2 500 Training, 500 testing



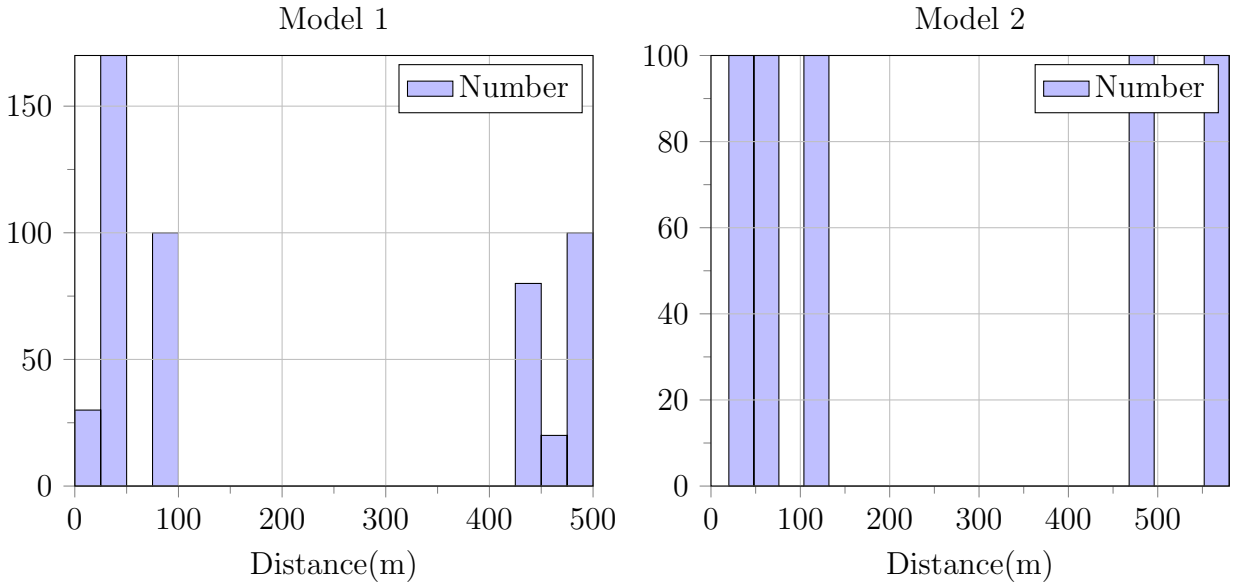


Table 4.1: Cross validation result

	Minimum	Maximum
Distance(m)	547.4	682.9
Position	200	200

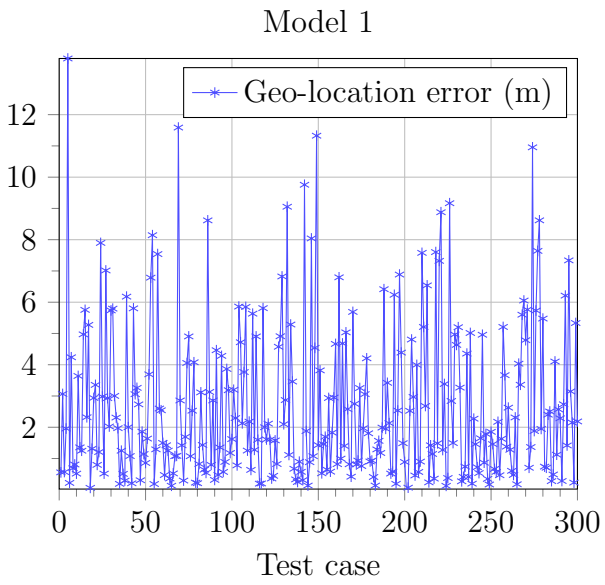
4.0.3 Without cross validation

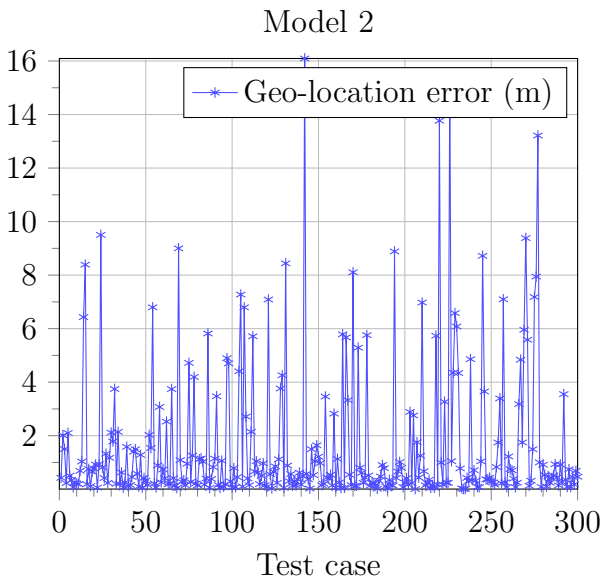
Table 4.2: Training and testing

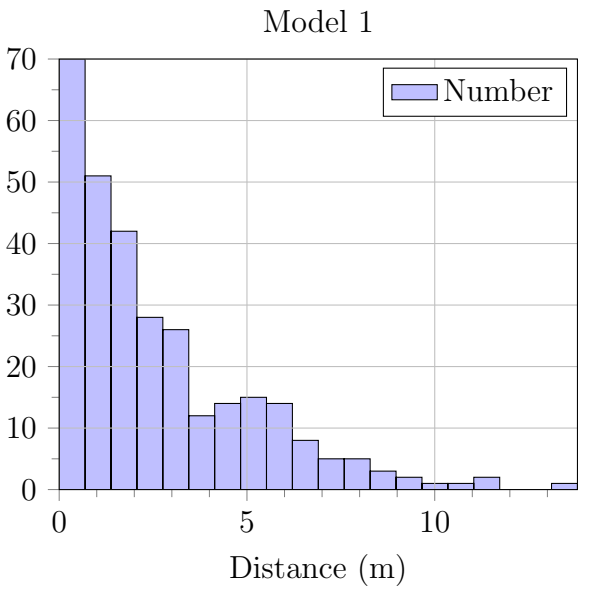
	Training	Testing
Latitude	700×6 table	300×6 table
Longitude	700×6 table	300×6 table

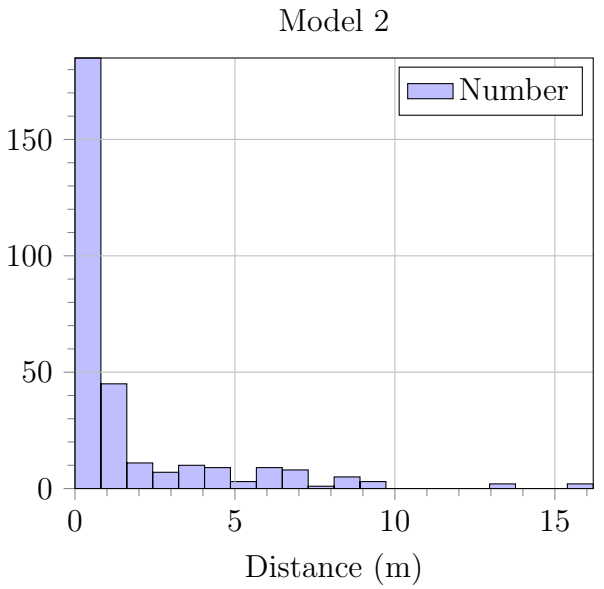
Table 4.3: Models with properties as per MATLAB's fitrgp() function

Properties	Model 1	Model 2
KernelFunction	-	ardsquaredexponential
FitMethod	-	sr
PredictMethod	-	fic
Standardize	-	1
Sigma	0.002	0.002

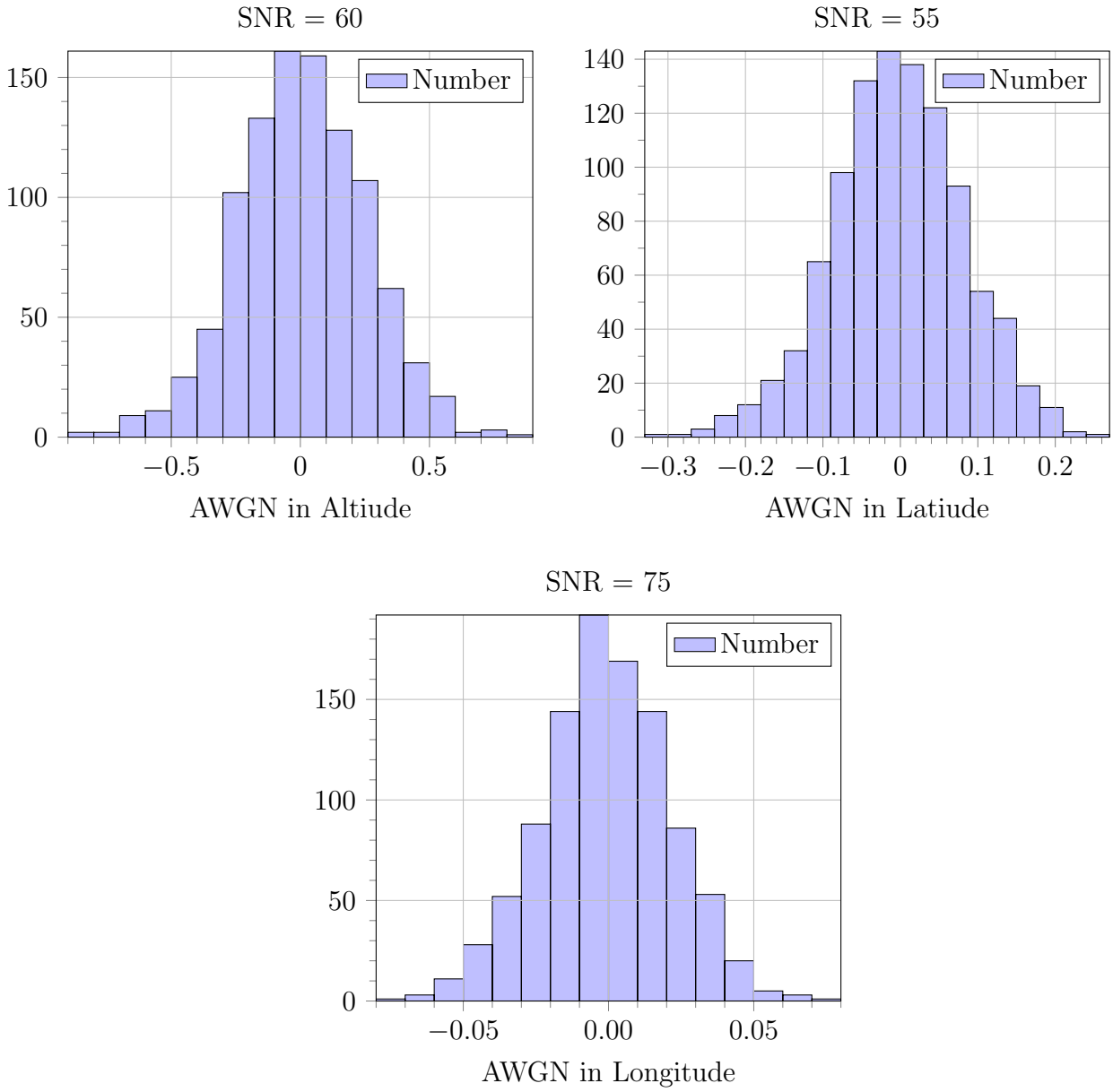

Model 1
Geo-location error (m)
12
10
8
6
4
2
0
50
100
150
200
250
300
Test case

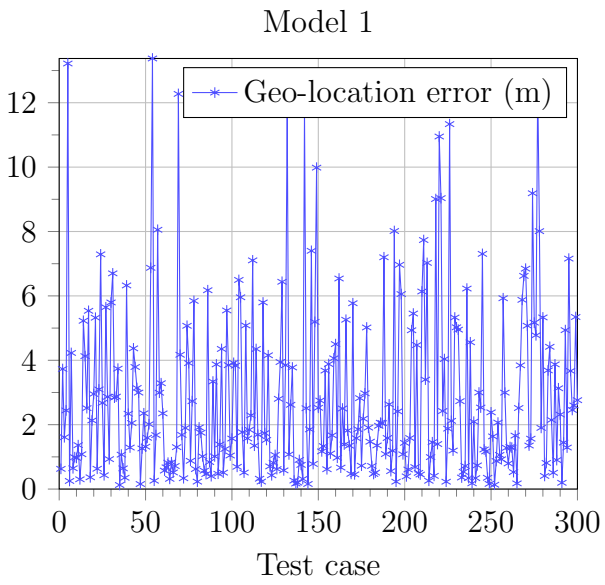

Model 2
Geo-location error (m)
16
14
12
10
8
6
4
2
0
50
100
150
200
250
300
Test case

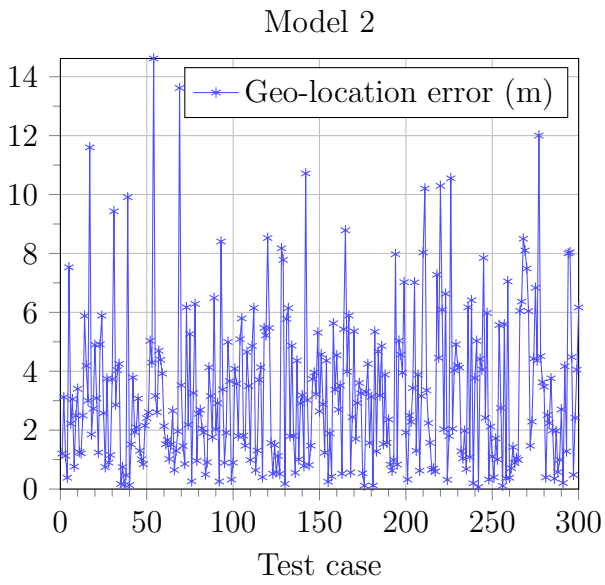

Model 1
Number
70
60
50
40
30
20
10
0
0
5
10
Distance (m)

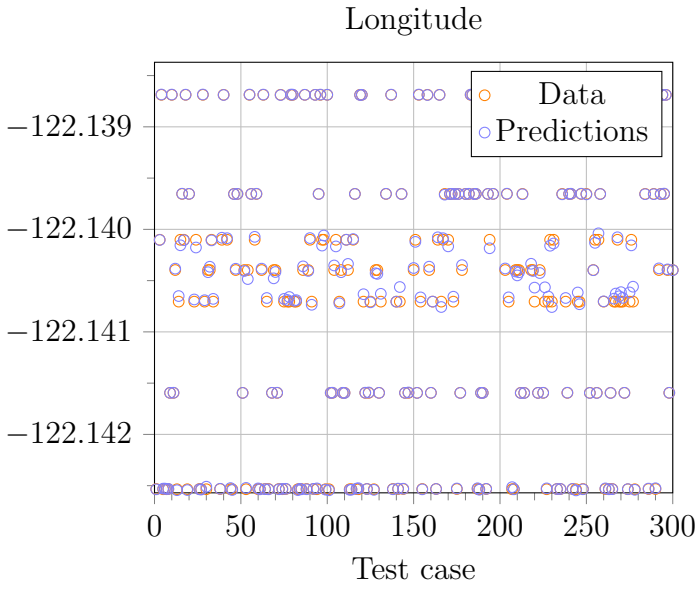

Model 2
Number
150
100
50
0
0
5
10
15
Distance (m)

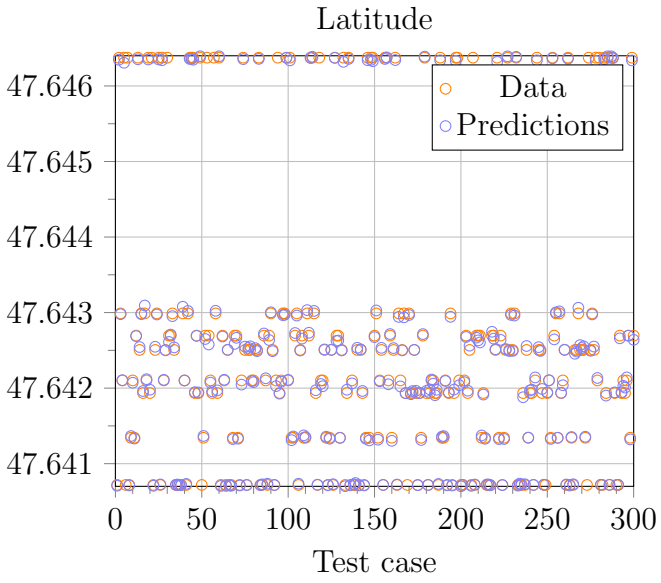
4.0.4 Without cross validation but adding noise

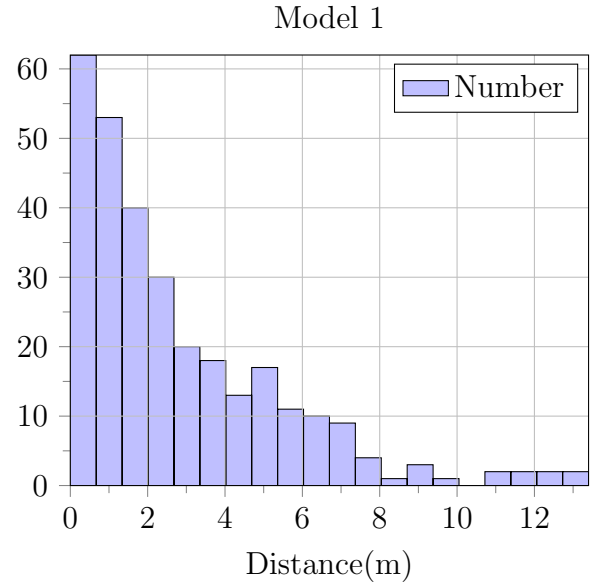


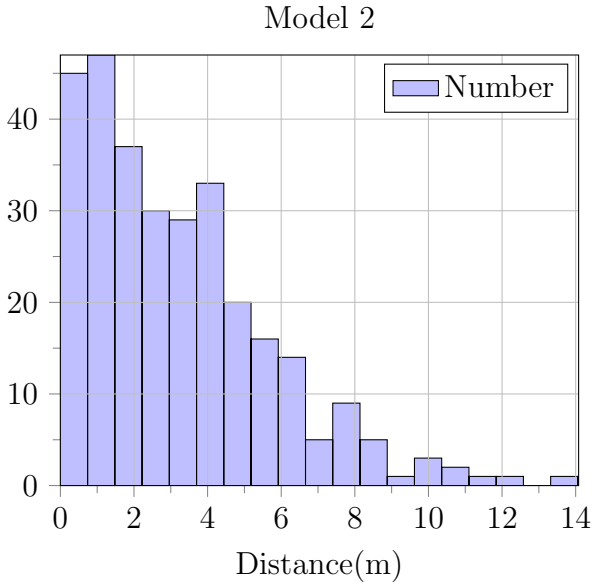

Model 1
Geo-location error (m)
0
2
4
6
8
10
12
0
50
100
150
200
250
300
Test case


Model 2
Geo-location error (m)
0
2
4
6
8
10
12
14
0
50
100
150
200
250
300
Test case


Longitude
Data
Predictions
−122.139
−122.140
−122.141
−122.142
0
50
100
150
200
250
300
Test case


Latitude
Data
Predictions
47.646
47.645
47.644
47.643
47.642
47.641
0
50
100
150
200
250
300
Test case


Model 1
Number
0
10
20
30
40
50
60
0
2
4
6
8
10
12
Distance(m)


Model 2
Number
0
10
20
30
40
0
2
4
6
8
10
12
14
Distance(m)

Distance(m)	Model 1	Model 2
Maximum	13.3737	14.6186
Minimum	0.1261	0.0669

Future Work

Some more sets of images will be collected and correspondingly the n target locations will be acquired for training and testing. The images can be passed through a convolutional neural networks for detection of object detection such as litter on beaches or for coordinate location of objects. Moving vehicle detecting, tracking, and geolocating based on a monocular camera, a GPS receiver, and inertial measurement units (IMUs) sensors will be used. Vehicle detection and efficiency for small object detection in complex scenes would be investigated. COSYS-AIRSIM [9] may be used for more images.

Then, a visual tracking method based on filters, and a geolocation method to calculate the GPS coordinates of the moving vehicle may be implemented. Flight control method in terms of the previous image processing results would be introduced to lead the UAV that is following the moving vehicle. The framework should automatically supervise on target vehicles in real-world experiments, which would suggests its potential applications in urban traffic, logistics, and security.

References

[1] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[2] *jMAVSim*. URL: https://github.com/DrTon/jMAVSim.

[3] S. Shah, D. Dey, C. Lovett and A. Kapoor. "AirSim: high-fidelity visual and physical simulation for autonomous vehicles". In: *IEEE Transactions on Geoscience and Remote Sensing* (2017), p. 14.

[4] Xin Wu et al. "Deep Learning for Unmanned Aerial Vehicle-Based Object Detection and Tracking: A survey". In: *IEEE Geoscience and Remote Sensing Magazine* 10.1 (Mar. 2022), pp. 91–124. ISSN: 2373-7468. DOI: 10.1109/mgrs.2021.3115137. URL: http://dx.doi.org/10.1109/MGRS.2021.3115137.

[5] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press. Cambridge, Massachusetts, 2006.

[6] R.M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer New York, 2012. ISBN: 9781461207450. URL: https://books.google.co.in/books?id=LHHrBwAAQBAJ.

[7] Alex Smola and Bernhard Schölkopf. "Sparse Greedy Matrix Approximation for Machine Learning". In: Jan. 2000, pp. 911–918.

[8] Joaquin Quiñonero Candela and Carl Edward Rasmussen. "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *J. Mach. Learn. Res.* 6 (2005), pp. 1939–1959. URL: https://api.semanticscholar.org/CorpusID:16005390.

[9] Wouter Jansen et al. "COSYS-AIRSIM: A Real-Time Simulation Framework Expanded for Complex Industrial Applications". In: *2023 Annual Modeling and Simulation Conference (ANNSIM)*. 2023, pp. 37–48.