Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2022-06-27

Efficient and Adaptive Decentralized Sparse Gaussian Process Regression for Environmental Sampling Using Autonomous Vehicles

Tanner A. Norton
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

 Part of the Physical Sciences and Mathematics Commons

BYU ScholarsArchive Citation

Norton, Tanner A., "Efficient and Adaptive Decentralized Sparse Gaussian Process Regression for Environmental Sampling Using Autonomous Vehicles" (2022). *Theses and Dissertations*. 9609.
https://scholarsarchive.byu.edu/etd/9609

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.

Efficient and Adaptive Decentralized Sparse Gaussian Process Regression for Environmental Sampling Using Autonomous Vehicles

Tanner A. Norton

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Michael A. Goodrich, Chair
Cameron K. Peterson, Adviser
Josh Mangelson

Department of Computer Science

Brigham Young University

Copyright © 2022 Tanner A. Norton

All Rights Reserved

ABSTRACT

Efficient and Adaptive Decentralized Sparse Gaussian Process Regression for Environmental Sampling Using Autonomous Vehicles

Tanner A. Norton
Department of Computer Science, BYU
Master of Science

In this thesis, I present a decentralized sparse Gaussian process regression (DSGPR) model with event-triggered, adaptive inducing points. This DSGPR model brings the advantages of sparse Gaussian process regression to a decentralized implementation. Being decentralized and sparse provides advantages that are ideal for multi-agent systems (MASs) performing environmental modeling. In this case, MASs need to model large amounts of information while having potential intermittent communication connections. Additionally, the model needs to correctly perform uncertainty propagation between autonomous agents and ensure high accuracy on the prediction.

For the model to meet these requirements, a bounded and efficient real-time sparse Gaussian process regression (SGPR) model is needed. I improve real-time SGPR models in these regards by introducing an adaptation of the mean shift and fixed-width clustering algorithms called radial clustering. Radial clustering enables real-time SGPR models to have an adaptive number of inducing points through an efficient inducing point selection process. I show how this clustering approach scales better than other seminal Gaussian process regression (GPR) and SGPR models for real-time purposes while attaining similar prediction accuracy and uncertainty reduction performance.

Furthermore, this thesis addresses common issues inherent in decentralized frameworks such as high computation costs, inter-agent message bandwidth restrictions, and data fusion integrity. These challenges are addressed in part through performing maximum consensus between local agent models which enables the MAS to gain the advantages of decentralization while keeping data fusion integrity. The inter-agent communication restrictions are addressed through the contribution of two message passing heuristics called the covariance reduction heuristic and the Bhattacharyya distance heuristic. These heuristics enable user to reduce message passing frequency and message size through the Bhattacharyya distance and properties of spatial kernels.

The entire DSGPR framework is evaluated on multiple simulated random vector fields. The results show that this framework effectively estimates vector fields using multiple autonomous agents. This vector field is assumed to be a wind field; however, this framework may be applied to the estimation of other scalar or vector fields (e.g., fluids, magnetic fields, electricity, etc.).

Keywords: Sparse Gaussian process regression, clustering, event-triggered, decentralized, sensor fusion, uncertainty propagation, inducing points

ACKNOWLEDGMENTS

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry/University Cooperative Research Center (I/UCRC) under NSF award No. IIP-1650547 along with significant contributions from C-UAS industry members. Great appreciation goes towards the members of this thesis committee (Dr. Goodrich, Dr. Peterson, and Dr. Mangelson) who have freely given advice and direction regarding this research. I would also like to thank Dr. Matthew Heaton and Dr. Philip White from BYU's Department of Statistics for their added input and advice.

Table of Contents

List of Figures

Chapter 1

Introduction

Multi-agent systems (MASs) are extremely valuable due to the large number of benefits they offer compared to a single agent system. MASs can increase areas of operation, provide time-sensitive information, and produce high-fidelity information. A group of coordinating agents can quickly explore large environments and use their measurements to rapidly create a model of important parameters within the operating region. However, coordinating and controlling a large number of agents in a MAS comes with complex challenges such as high computation costs, inter-agent message bandwidth restrictions, and data fusion integrity. I address these issues for coordinated autonomous agents with a novel decentralized sparse Gaussian process regression (DSGPR) framework with event-triggered, adaptive inducing points.

A MAS equipped with this DSGPR framework has a wide range of real-world applications for estimating various scalar or vector fields. Some exciting applications include measuring and tracking smoke plumes, radiation, and oceanic scalar fields (such as salinity and water temperature), or estimating wind fields to help with optimal vehicle control [8, 20, 27]. DSGPR provides a framework for cooperating agents to fuse information gained from environmental measurements. In this thesis, we assume that the environmental parameters being measured are a three-dimensional wind field, but this approach can be generalized to any scalar or vector field.

Figure 1.1 illustrates a simulation of two agents (red circles) modeling a two dimensional wind field. The blue arrows denote the agent's current estimation of the wind field at predetermined test points. The gold arrows show the unknown ground truth wind field values. As seen in the figure, the aerial agents perform an effective environmental modeling of the wind field. Note that we depicted a two dimensional scenario due to the difficulty of clearly illustrating three dimensional vector fields. All the algorithms and approaches presented in this thesis are capable of operating in three dimensions.

1.1 Related Works

Environmental modeling has been studied previously by many other groups; this research varies between centralized, distributed, and decentralized methods. Decentralized estimation allows many agents to fuse measurements to generate individual local representations of the global scalar or vector field. Decentralized approaches require agents to do individual computation and also perform inter-agent communication. Decentralized methods do not require a central server as all agents individually update their representation of the global model. This makes decentralized algorithms robust to single agent failures [1, 4, 26]. Centralized estimation requires agents to send their information (e.g. measurements, state values) to a central server

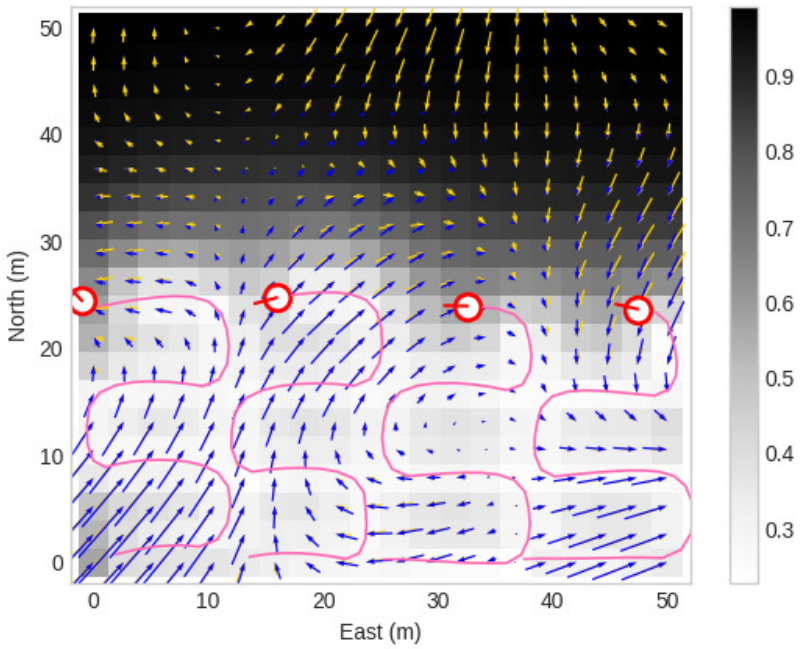

Figure 1.1: This figure depicts four agents as they cooperatively estimate an environmental field. The red circles and lines denote the agent's current position and heading respectively. The blue arrows show the agent's prediction of the wind-field. While the gold arrows are the ground truth wind field. Pink lines show the path taken by each agent. The gray/black color bar/heat-map shows the global variance of the wind field. Darker color denoting higher uncertainty/variance, while lighter color denotes lower uncertainty/variance.

that computes the global model. While distributed estimation splits data sets into subsets that are sent to multiple agents to be processed. The agents then send a estimation of their subset of data to a central server where the summaries are combined [9, 22].

In this work, I focus on the development of a real-time decentralized multi-agent framework using sparse Gaussian process regression . Other approaches in both decentralized and distributed scenarios have provided the foundational knowledge that has led to the contributions of this thesis. These contributing papers are discussed next.

A distributed regression system is proposed in [37] where agents send messages in small neighborhood groups (agents within message passing range) to share information regarding a desired approximation problem. The system is fully distributed and convergence is guaranteed under the correct circumstances. They focus on their system being consistent while confident, where some data fusion approaches become overly conservative in their estimations. Their implementations only deal with a parametric Kalman filter, while in this thesis, I create a decentralized semi-parametric information fusion model.

Other research has investigated decentralized GPRs. The work from [4] outlines a decentralized Gaussian process approach for mobility-on-demand systems. To combine information, they assume a common set of inducing points (or support set) that are shared

among all agents. This system tracks and predicts traffic flow based on mobile sensing units that travel different roads given a certain boundary. Each agent in the algorithm produces summary statistics for a common support set. Then the agents share there summary statistics and create a global model by combing all agents local models.

Their work is extended by [26] to have separate support sets for sub-regions of the boundary space. However, this method falls short if there is insufficient data in a sub-region. DSGPR does not assume a common set of inducing points (support set) ensuring the agents can explore new areas of the boundary space according to the needs/direction of a path planning algorithm or user.

The authors from [1] present a decentralized Gaussian process fusion approach for MAS information sharing and prediction. Instead of sharing measurements, agents share a local GPR model with other agents. Artificial measurements are then drawn from those local models and fed into a global model. The authors admit that this can lead to inaccurate artificial measurements being used in the global model. Furthermore, the authors in [21] create a decentralized GP fusion method that allows agents to fuse other agent models into a global model. Thus allowing them to mitigate the propagation of rumors throughout (redundant measurement information) their decentralized system. However, their approach does not mention any sparse GPs which would decrease the scalability of their system.

A highly scalable decentralized approach called COOL is presented in [17]. Their work focuses on scaling to thousands of agents and enabling each agent to have their own hyper-parameters, which can be tuned online. However, their approach does not account for a varying number of inducing points.

The research in [39] focuses on adaptive model prediction using local model formation that adapts to new streaming data. While their local model formation is adaptive, those local SGPR models do not contain adaptive inducing points.

Because GPRs are inherently computationally inefficient, many researchers have developed SGPR algorithms that use pseudo measurements to reduce the computational complexity of a GP, while retaining much of its accuracy [28, 30, 33–35].

1.2 Contribution

In this thesis, I extend the work of prior researchers in providing SGPR models with the ability to choose a varying number of inducing points depending on the current data distribution. Adaptive inducing points enable accurate data point summarization for scenarios where the data set changes over time. Adaptive inducing points can also allow SGPR models to be implemented in applications that must run in real-time. On top of the adaptive inducing points, this thesis presents a decentralized SGPR framework for scalar or vector field modeling. This framework also enables message sharing between multiple decentralized agents. The presented framework and message heuristics allow small to medium sized groupings of agents to be utilized in real-time modeling scenarios.

These contributions of this thesis to decentralized MAS estimation are as follows:

1. A clustering algorithm providing an adaptive number of inducing points and their locations for decentralized SGPR.

2. A guaranteed two and three dimensional upper bound for those adaptive inducing points.
3. An effective event-triggering algorithm for updating inducing point values and locations.
4. A decentralized SGPR framework equipped with message passing heuristics that control message size and message passing frequency.

1.3 Background

This section provides the background information on GPR and SGPR models necessary to understand my contributions. I start by describing an exact GPR and then describe a common SGPR model that was developed in [29, 35].

1.3.1 Exact Gaussian Process Regression

Gaussian process regression (GPR) is a powerful non-parametric data driven model. GPRs approximate the latent ground truth function using data and distance properties inherent in the specified kernel function. GPs are a collection of random variables which each have a Gaussian distribution, and also which are jointly Gaussian.

Let the data be measured (training) points denoted as $\boldsymbol{x} = [x_1, ..., x_N]$ where $\boldsymbol{x}$ represents a vector of N training point locations. Let $\boldsymbol{x_*}$ be the test points, or the points at which prediction will occur. The true function values for $\boldsymbol{x}$ and $\boldsymbol{x_*}$ are denoted as f_* and f and their priors can be described jointly as

$$\begin{bmatrix} f_* \\ f \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{**} & K_{*f} \\ K_{f*} & K_{ff} \end{bmatrix} \right), \tag{1.1}$$

where the subscripts on the Ks denote whether the kernel was performed on the training or test set (the subscripts do not represent an index). It is common for a zero mean to be assumed on the prior. Using a regression model $y_i = f(x_i) + v$ with $i \in [1, N]$ and $v \sim \mathcal{N}(0, \sigma^2)$ as measurement noise, the GPR is characterized by

$$f(\boldsymbol{x}) \sim GP(m(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x'})), \tag{1.2}$$

with $m(\boldsymbol{x})$ and $K(\boldsymbol{x}, \boldsymbol{x'})$ being the mean and covariance functions, respectively.

The mean and covariance define a Gaussian distribution of all possible functions that describe the true underlying function f_* given the training points and hyper parameters, i.e., $P(f_*|f, \boldsymbol{x}, \theta)$ where θ is the hyper-parameters. This conditioning occurs because it is assumed that there is a dependence structure between f_* and f; the dependence structure used in GPs is called the kernel function. A kernel function is used to generate the covariance matrix and account for spatial similarities of test and training points. The specific kernel function used in this thesis is the Laplacian kernel (part of the radial basis function (RBF) family of kernels),

$$K_{ij} = k(x_i, x_j) = \sigma_f^2 e^{-\lambda \|x_i - x_j\|}, \tag{1.3}$$

where the hyper-parameter σ_f^2 is the variance of the signal and λ is the length-scale.

To predict over the x_* values, the predictive posterior for f_* must be found i.e., $P(f_*|f, \boldsymbol{x}, \theta)$. This multivariate conditioning results in a mean of

$$f_*(\boldsymbol{x_*}) = m(\boldsymbol{x_*}) + K_{*f}(K_{ff} + {\sigma_n}^2 I_N)^{-1} K_{f*}(\boldsymbol{y} - m(\boldsymbol{x})), \tag{1.4}$$

where $m(\boldsymbol{x_*})$ is the prior mean function of the desired test points (zero mean), I_N is the $(N \times N)$ identity matrix, $\boldsymbol{y}$ is the vector of observed values obtained at each position $\boldsymbol{x}$. The ${\sigma_n}^2$ term is the observation noise and $m(\boldsymbol{x})$ is the mean function for the prior of f (which is also assumed to be zero mean). The term $K_{ff} = k(x_i, x_j)$, $\forall i, j \in \boldsymbol{x}$ and $K_{*f} = k(x_i, x_j)$, $\forall i \in \boldsymbol{x_*}$ and $\forall j \in \boldsymbol{x}$. Note that K_{f*} is the transpose of K_{*f}. The result $f_*(\boldsymbol{x_*})$ is a vector with size $(N_* \times 1)$ containing the mean scalar field approximation at each given test point $\boldsymbol{x_*}$.

The covariance matrix for the predictive posterior for f_* is

$$cov(f_*) = K_{**} - K_{*f}(K_{ff} + {\sigma_n}^2 I_N)^{-1} K_{f*}, \tag{1.5}$$

where $K_{**} = k(x_i, x_j), \forall i, j \in \boldsymbol{x_*}$.

The two main drawbacks for using exact GPRs are the high computation and storage costs. Exact GPR has $O(N^3)$ computation complexity, and $O(N^2)$ space complexity. Sparse approaches have been developed to decrease the space and computational complexity of GPRs.

1.3.2 Sparse Gaussian Process Regression

High computational costs of exact GPs have lead to the development of sparse GPRs (SGPR). SGPRs rely on selecting a set of summary points that approximately describe the whole data set. These summary data points are called inducing points. To reduce the computational complexity of an exact GPR, M (the number of inducing points) must be much smaller than N (the number of training points). This reduces the computational complexity from $O(N^3)$ to $O(NM^2)$. The main difference between SGPR approaches is in how the inducing points are selected [28, 30, 33–35]. The authors in [29, 30, 35] select a representative set of inducing points; these inducing points can be described by a Gaussian process. The mean and covariance of this inducing point Gaussian process is given by

$$\boldsymbol{m} = {\sigma_n}^{-2} K_{uu} \Sigma K_{uf} \boldsymbol{y} \tag{1.6}$$

and

$$\Lambda = K_{uu} \Sigma K_{uu}, \tag{1.7}$$

where $\Sigma = (K_{uu} + {\sigma_n}^{-2} K_{uf} K_{fu})^{-1}$.

The u subscript represents the vector of inducing points and the f subscript represents vector of data or training points. The matrix K_{uu} is the output of the kernel function $K_{uu} = k(u_i, u_j), \forall i, j \in \boldsymbol{u}$ where $\boldsymbol{u}$ is the vector of inducing point locations. The matrix K_{uf} is the output of the kernel function $K_{uf} = k(u_i, x_j)$, $\forall\, i \in \boldsymbol{u}$, and $\forall j \in \boldsymbol{x}$ where $\boldsymbol{x}$ is a vector of measured (training) point locations. Using the distribution of the inducing points, the mean and covariance functions of the predictive posterior are

$$f_*(\boldsymbol{x_*}) = K_{*u} K_{uu}^{-1} \boldsymbol{m} \tag{1.8}$$

and

$$cov(f_*) = K_{**} - K_{*u}K_{uu}^{-1}K_{u*} + K_{*u}K_{uu}^{-1}\Lambda K_{uu}^{-1}K_{u*}. \tag{1.9}$$

These equations result in a computational complexity of $O(NM^2)$ instead of $O(N^3)$ allowing SGPRs to be used on large data sets. However, most implementations of SGPR do not allow for a varying number of inducing points as new data arrives.

1.4 Thesis Outline

The remainder of this thesis will describe my contributions in depth. Chapter 2 presents and outlines the radial clustering algorithm that enables adaptive inducing points. In this chapter, I provide a guaranteed upper bound on the number of inducing points that will be required in a given cubic boundary space. Chapter 3 introduces the DSGPR model and message passing protocols and heuristics. In Chapter 4, I present simulation results using the DSGPR model for a MAS. I analyze and discuss the DSGPR framework and its attributes across a variety of parameters, including the number of agents, message passing ranges, and heuristic parameters. I also compare the DSGPR framework to other seminal GPR models to show the advantages and drawbacks of the usage of each model.

Chapter 2

Radial Clustering

In this chapter, I present an efficient algorithm capable of adapting the number of inducing points in a SGPR. This algorithm is called radial clustering (RC) and uses a principled approach to find a representative set of inducing points. I also provide a proven upper bound on the number of inducing points that will be created for a square or cubic boundary space. This bound is necessary to set upper limits on the memory and message size used when operating with cooperating agents. RC uses a kernel function to decide when to insert another inducing point. While the following explanation uses the Laplacian kernel, the concept is applicable to other spatial RBF kernels.

Most SGPR models assume a static number of inducing points [3, 4, 17, 35]. However, [11] also provide an SGPR with an adaptive number of inducing points. Their approach takes advantage of the already computed K_{uf} kernel matrix, which contains information regarding the spatial relationship between the inducing points and the measured points, to determine when a new inducing point should be added. If a measured point's similarity (i.e. the corresponding entry in the kernel matrix K_{uf}) becomes less than a chosen threshold (meaning it is far away from all other inducing points), they choose this isolated point to be another inducing point. Their method is simple and fast. However, this approach allows an edge case, when a chosen inducing point cannot be changed or adjusted once selected. This becomes an issue when a large number of points are measured at the edge of an inducing point's threshold; the model's ability to correctly predict at test point locations nearby this area is hindered. The fixed inducing point no longer summarizes the data as well as a centroid point, which could decrease the model's accuracy. In this chapter, I show that RC is not hindered by this same edge case scenario, due to its ability to actively re-position old inducing points as new data becomes available.

The remainder of this chapter will proceed as follows. Section 2.1 describes the radial clustering algorithm in detail. The upper bound for the radial clustering algorithm is derived in Section 2.2. Finally, Section 2.3 presents an event condition that triggers radial clustering in a real-time implementation.

2.1 Radial Clustering Formulation

The goal of the RC algorithm is to provide a SGPR model with the ability to adapt the number of inducing points it is using in real-time. In real-time prediction, agents will receive a stream of data over time where it is difficult to predict the exact number of inducing points needed to best summarize the underlying data while not using an unnecessarily large number of inducing points. Thus, agents need to adapt the number and location of the inducing points according to the arriving data. To achieve this streaming data adaptability, RC can cluster

the existing data using the kernel function already in use with the SGPR. The centroids of these clusters becomes the new inducing points, and through the event triggering mentioned in Section 2.3 these clusters can be updated according to freshly arriving data. These clusters are determined through the same kernel used in the GPR model.

To create these clusters, I use the kernel to create groupings of similar data measurements. Using the kernel, RC groups measured points into clusters when groups of points have a spatial similarity defined by κ. The value of κ is chosen by the user to determine at what similarity (or distance) clustering should occur at between measurements. Let $0 < \kappa \leq \sigma_f^2$ be a constant output of the of the Laplacian kernel, and define $r = \|x_i - x_j\|$. Equation (1.3) can be solved for r yielding $r = log(\kappa/\sigma_f^2)/-\lambda$. Essentially, r now represents the distance between two points where their similarity has a value κ. With this rearrangement, the r distance can now be thought of as a radius where every point inside the radius has a similarity score greater than that of κ. Algorithm 1 shows how r is used in the radial clustering (RC) approach.

Algorithm 1: Radial Clustering

```
Data: D: Measured Data, r: Cluster Radius
Result: Centroid Points
for measurement in D do
    for point in clusters do
        if L2 distance(measurement - point) ≤ r then
            append measurement to point's cluster;
    if measurement not in clusters then
        add new cluster with measurement as point;
return Compute centroids(clusters);
```

As shown in Algorithm 1, the distance between a measured point and each current cluster point is calculated. If the distance between those two points is less than r, then the measured point is appended to the cluster associated with the cluster point. A cluster point is created if a measured point does not fall within the radius of any existing cluster; that measured point will then become a cluster point. There are no initial clusters are the beginning of the algorithm. Once all points have been assigned a cluster, the centroids of each cluster are computed. Because of the DSGPR framework has each agent creating their own isolated local model, the computed centroids become the inducing points for that local SGPR. However, clustering does not occur on the global DSGPR model outlined in Section 3.2. The advantage of the RC approach on local models is that the measured points are dynamically represented as the data set changes over time. The number of clusters is dependent on the given radius r and the distribution of the measurements.

Figure 2.1 shows how RC performs given a run of two agents randomly moving through a two dimensional field. Each agent will cluster its own measured points individually. Both paths are shown to represent how RC occurs in a decentralized setting. RC effectively reduces a total of 102 measured points into 38 clusters resulting in 38 inducing points totaled between both agents. The star markers denote the cluster centroids which are then chosen to be the

inducing points. Measured points in the same cluster are identified by the same color. As shown in the figure, RC performs well at summarizing the data into a small representative number of clusters without having to know the total number of clusters beforehand.

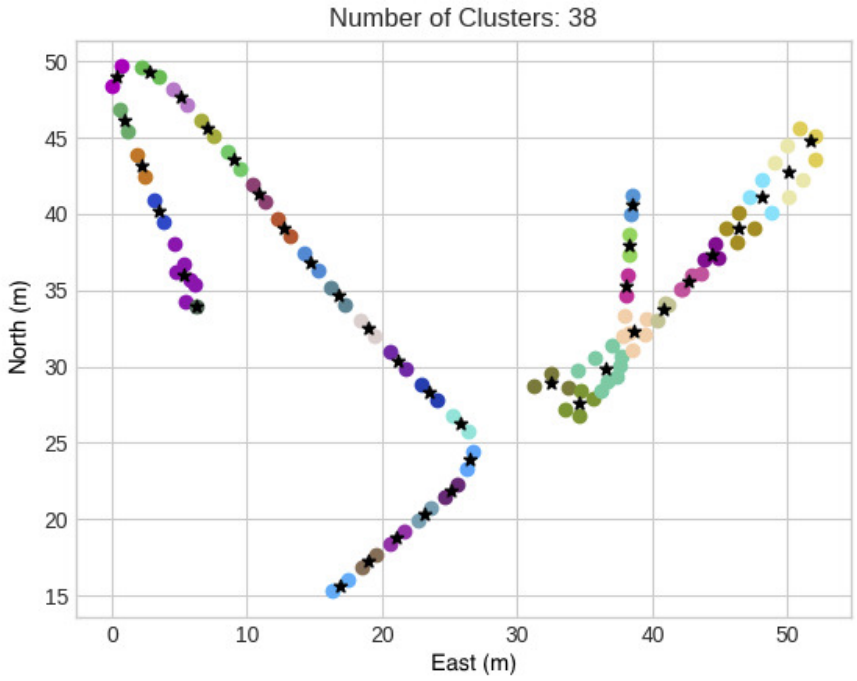

Figure 2.1: Example of two agent's paths clustering by radial clustering. The star marker denotes the cluster centroid which is selected as an inducing point.

This notable advantage of knowing clusters beforehand comes from properties of the mean-shift [5] and fixed-width clustering algorithms. Radial clustering (RC) is combination of the mean-shift and fixed width clustering [2] algorithms. Mean-shift uses a user chosen kernel to find regions with dense data and then iteratively shifts the points towards the those regions of highest density. Once iteration stops, clusters are assigned. RC does not iterate to change the clusters; this would be an expensive operation in real-time. Since clustering will occur more than once in most real-time scenarios, spending large amounts of computation time to find the optimal clusters has diminishing returns. During the clustering process, RC does not adjust the mean of the cluster as new points are added. This is different from the fixed-width clustering approach which does adjust the mean once a new point is added. RC is also different from fixed-width in that I am using a kernel to determine the fixed-width that will be used in the clustering process. Therefore, RC is a combination of the fixed-width and mean-shift clustering algorithms.

The fixed width clustering has $O(n^2)$ for its computational complexity, and the mean-shift approach is $O(T \cdot n^2)$ where T is the number of iterations. The computational complexity of RC is $O(n^2)$. However, since the mean-shift approach is $O(n^2)$ for each iteration, RC is faster by a factor of T. Furthermore, when thinking of my use case of agents measuring the scalar or vector field of a boundary space, I can show that the computational complexity for

my case is $O(u_b n)$ where u_b is the upper bound of the number of possible clusters that can be formed in the boundary space and n is the number of measured points. The viability of RC as a real-time solution for finding effective inducing points is shown in Section 4.3.2.

Those familiar with other SGPR inducing point optimization methods might suggest using the cluster centers as seeds to further optimize the inducing point locations. However, even with good seed values, the computational cost/time is typically too large to optimize inducing points in real-time systems (especially when the number of inducing points is high). Section 4.2.3 expressly shows a comparison of DSGPR to a common SGPR inducing point optimization approach. The cost of optimizing the inducing points is much greater than the cost to re-cluster, and there isn't a significant difference in prediction accuracy.

2.2 Radial Clustering Upper Bound

To find RC's computational complexity $O(u_b n)$ from Section 2.1, we must calculate u_b which is the maximum number of inducing points possible for an agent given a fixed boundary space, which I will refer to in this section as the cubic container. Knowing this upper bound also allows users and system designers to account for how large the message sizes between cooperating agents can become if every agent reaches this maximum. I provide this guaranteed upper bound of the RC inducing point numbers in a fixed cubic container for both 2D and 3D.

I find this upper bound using existing theory developed in the field of sphere packing. Articles [6, 7, 10, 12–15, 24, 36, 38] provide an in-depth discussion on sphere packing. For this chapter, each inducing point will be thought of as a sphere. I do this because each inducing point has a radius in which another inducing point center will not be placed. This spherical representation of inducing points leads us to the topic of fitting/packing spheres into cubes. I will briefly introduce the sphere packing topic so it is clear how I am taking the proof that comes with packing spheres into cubes to finding the RC upper bound.

Section 2.2.1 outlines the proof from [38] for finding the upper bound of packing spheres in a cube. Then, in Section 2.2.2 I modify the proof from Section 2.2.1 to create a guaranteed upper bound on the number of RC inducing points for three dimensions. In Section 2.2.3, an inducing point upper bound is derived for 2D. And finally, in Section A, a process is presented that finds an RC upper bound for 2D that is not guaranteed, but is a less-conservative approximation than what is presented in Section 2.2.3.

2.2.1 Optimal Sphere Packing Density for Cubic Containers

In this subsection, I provide a high-level overview of the sphere-packing research. This is essential context needed to understand the RC inducing point upper bound.

The optimal sphere packing problem is described as follows:

> *Find the maximum amount of congruent spheres that can be packed (non-overlapping) in a cubic container where (i) each sphere center must be at least* $2r$ *units away from every other sphere, where* r *is each sphere's radius, and (ii) no sphere can extend beyond the cubic container boundary.*

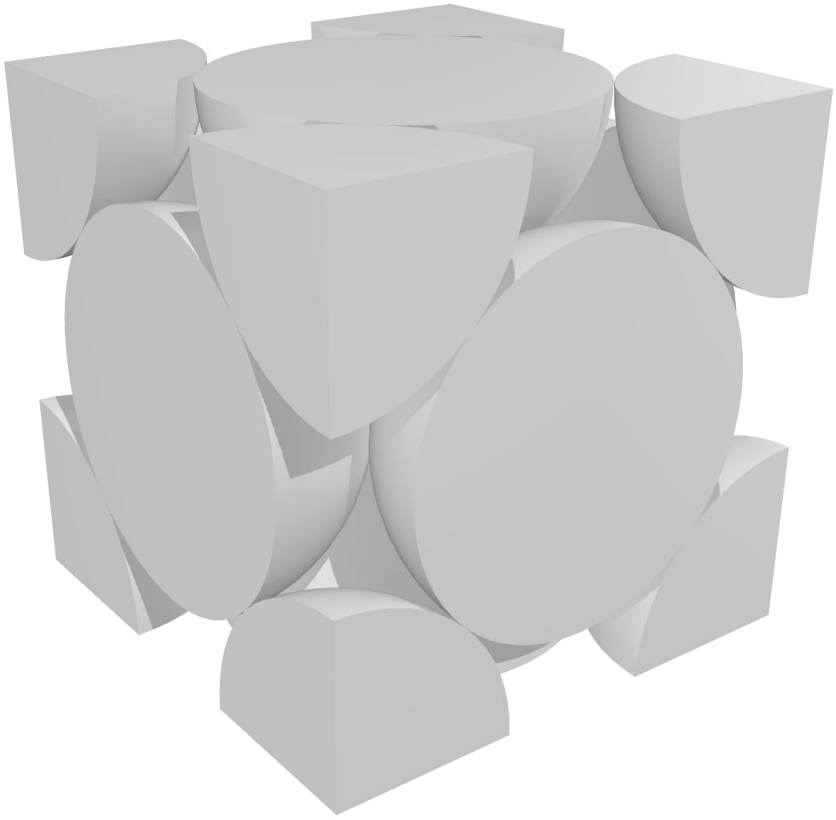

Figure 2.2: Base cube optimal sphere packing

A provably optimal approach for packing spheres in a cubic container is summarized in [38]. This approach finds a base cube of optimally packed spheres (shown in Figure 2.2) and finds the number of base cubes that fit inside the larger cubic container. The total number of spheres that fit in the cubic container is then determined using the base cube's "density". Where the packing density is defined to be the total volume of spheres that fit inside the volume of the cubic container.

Note that the base cube (in Figure 2.2) consists of six non-overlapping hemispheres placed at each face of the cube, and eight eighth spheres are placed at each of the eight corners. The length of each side of the cube is described as $2\sqrt{2}r$, and thus the volume of the base cube is $V_b = (2\sqrt{2}r)^3$. The well-known volume of an individual sphere is $V_i = \frac{4}{3}\pi r^3$. The total volume of spheres that pack the base cube can be computed by summing the volume of each partial sphere shown in Figure 2.2. Thus the total volume of spheres, $V_s = V_i(6 \cdot \frac{1}{2} + 8 \cdot \frac{1}{8}) = \frac{16\pi}{3}r^3$.

The packing density is defined to be the ratio between the volume of spheres in the cube with the volume of the cube, given as $P_d = V_s/V_b = \pi/\sqrt{18}$. We use packing density of the base cube to determine the maximum number of spheres that can be packed into a larger cubic container. A cubic container with side length b has a volume $V_c = b^3$ and will fit

$$T_S = \frac{V_c \cdot P_d}{V_i} \tag{2.1}$$

total number of spheres.

I use a similar idea of using a base cube to find a guaranteed upper bound for the maximum number of inducing points chosen in the RC algorithm in two and three dimensional spaces.

2.2.2 Three Dimensional Radial Clustering Upper Bound

Here, I provide the process for determining the guaranteed upper bound for the max number of clusters possible for RC in a cubic container. Similar to sphere packing, the problem of finding the upper bound for radial clustering can be described with the problem statement:

> *Find the maximum number of congruent spheres that can fit into a cubic container where (i) each sphere center must be at least a distance r away from every other sphere center, where r is the radius of each sphere, (ii) spheres are allowed to overlap each other and the boundary, and (iii) no sphere center may be placed outside the cubic container.*

Following the methodology described in Section 2.2.1, the total number of spheres stuffed into a cubic container can be determined by first creating a base cube of stuffed spheres (the term stuffed is used to reference the fact that the spheres in radial clustering can overlap each other's radii). Figure 2.3 shows the optimal configuration of spheres being stuffed into a base cube. The pattern shown in Figure 2.3 is continued with two more layers of spheres (total of three layers), yielding a total of 27 spheres that are partially or completely contained within the base cube. Similar to sphere packing, the goal is to calculate the total number of all spheres that have their centers inside the base cube. Simply stated, the summed volume of all spheres that are involved with the base cube will be used to compute the the total number of spheres that can fit into the cubic container. To find the total number of spheres, the cubic container will then be optimally filled with base cubes and the total sphere number that corresponds to those base cubes is used to find the total number of spheres inside the cubic container.

The stuffed base cube has a side length of $2r$ where r is each congruent circle radius. Each base cube has complete and partial spheres stuffed inside it. The interior sum of spheres is used when fitting all the base cubes inside the larger cubic container. The exterior summed number of spheres must be found so that the spheres that will extend passed the edge of the cubic container are accounted for. Combining all the interior and exterior sphere numbers related to the many base cubes inside the cubic container leads to the upper bound of all possible spheres that can possibly be fit inside the cubic container.

Interior Sphere Volume of the Base Cube

The interior of the base cube contains many overlapping partial spheres and a complete sphere; summing all those partial spheres and complete spheres gives the total number of spheres for the base cube interior. I will walk through how many spheres exist for each of the three layers shown in Figure 2.3 that strictly falls within the boundary of the base cube.

The bottom layer of spheres (where only 3 of the 9 spheres colored blue are shown in Figure 2.3) contains four corner spheres, four edge spheres, and one face sphere. The

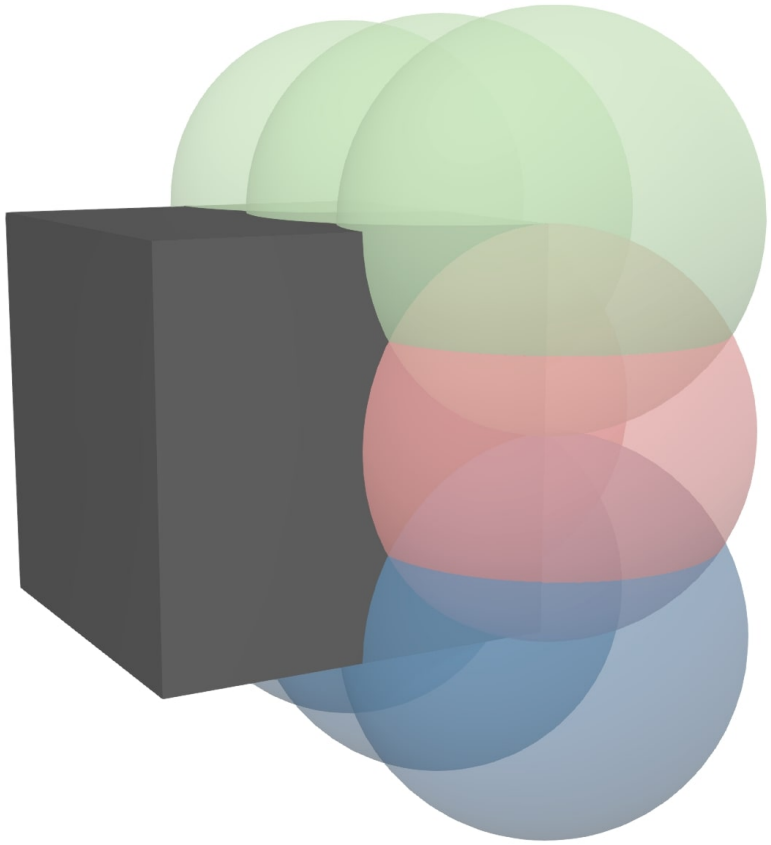

Figure 2.3: Base cube optimal sphere stuffing

corner spheres each take up an eighth of a sphere inside the cube. The four edge spheres each take up a fourth of a sphere inside the cube. The one face sphere has a volume of half a sphere in the base cube boundary. Thus the bottom layer has two complete spheres, $4(\frac{1}{8}) + 4(\frac{1}{4}) + 1(\frac{1}{2}) = 2$, that fall within the base cube boundary.

The middle layer (colored red spheres in Figure 2.3) has four edge spheres, four face spheres, and a single center sphere. The four edge spheres each have a fourth of a sphere residing in the base cube. The four face cubes each have a half sphere in the base cube. The center sphere fits completely in the base cube; therefore, its entire volume is counted for this layer. The number of spheres for the middle layer residing in the cube is $4(\frac{1}{4})+4(\frac{1}{2})+1(1) = 4$.

The top layer (green spheres in Figure 2.3)is exactly the same as the bottom layer. Thus bringing the total number of spheres including all the portions of spheres for the interior of the base cube to a count of $C_i := 2 + 4 + 2 = 8$.

Exterior Sphere Volume of the Base Cube

To find the exterior number of spheres, the volume associated with the base cube's six faces, twelve edges, and eight corners must be taken into account. To find the number of spheres that exist on the outside of the base cube, we must count how many total spheres (in terms of combined volume) exist on the face, edge, and corner of the base cube.

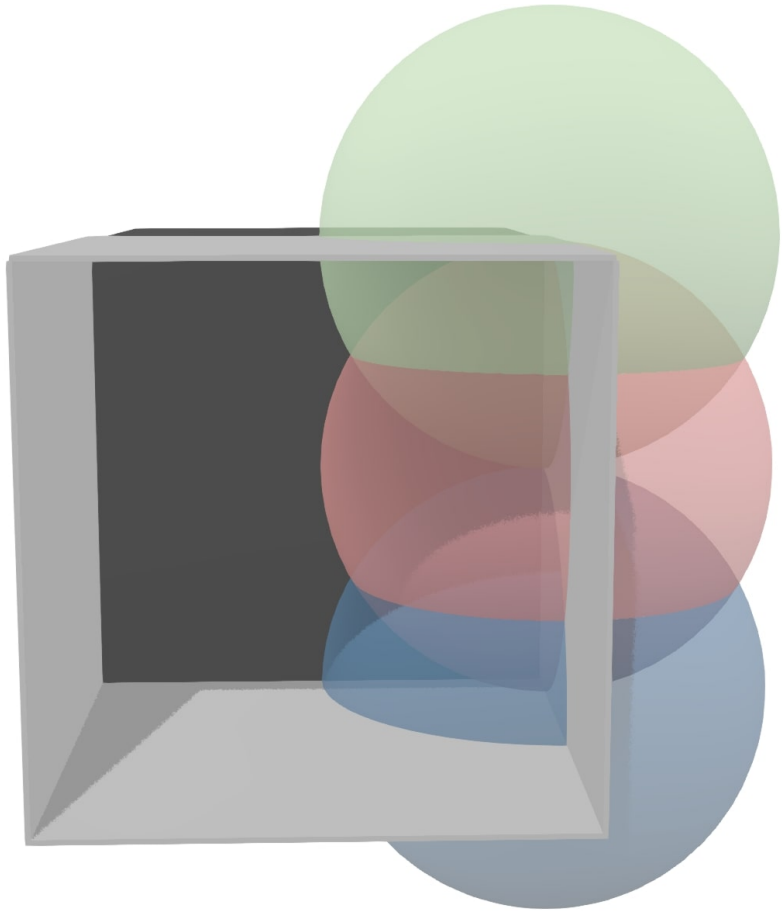

Figure 2.4: Illustration of some of the partial spheres that reside directly over the face of the base cube.

For a single face of the exterior of the base cube, we only count spheres that are directly over the face, or the volume that is contained by the white extending bounds as seen in Figure 2.4. To count up the spheres on the face, we start with the four spheres that lie at the corners of the face. Each of these four spheres have an eighth of a sphere volume residing directly above the face of the base cube. There are four spheres that reside on the edges of the face. Only a fourth of a sphere volume for these spheres lie directly over a base cube face. Lastly, there is a single sphere that lies in the middle or face of the base cube with half of a sphere volume that extends into the exterior region. In total, a single face of the base cube has a combined number of two spheres, $C_f \coloneqq 4(\frac{1}{8}) + 4(\frac{1}{4}) + 1(\frac{1}{2}) = 2$.

To count the number of spheres that exist on the exterior of an edge of the base cube, there are three spheres that must be accounted for. These three spheres are pictured by the extending white boundary in Figure 2.5. Note that this area will not overlap with areas extending from any face of the cube. The two corner spheres each have an eighth of a sphere that is directly over the edge of the base cube, while the middle edge sphere has a fourth of a sphere over the base cube edge. The number of spheres that reside on an edge of the base cube is $C_e \coloneqq 2(\frac{1}{8}) + 1(\frac{1}{4}) = 0.5$.

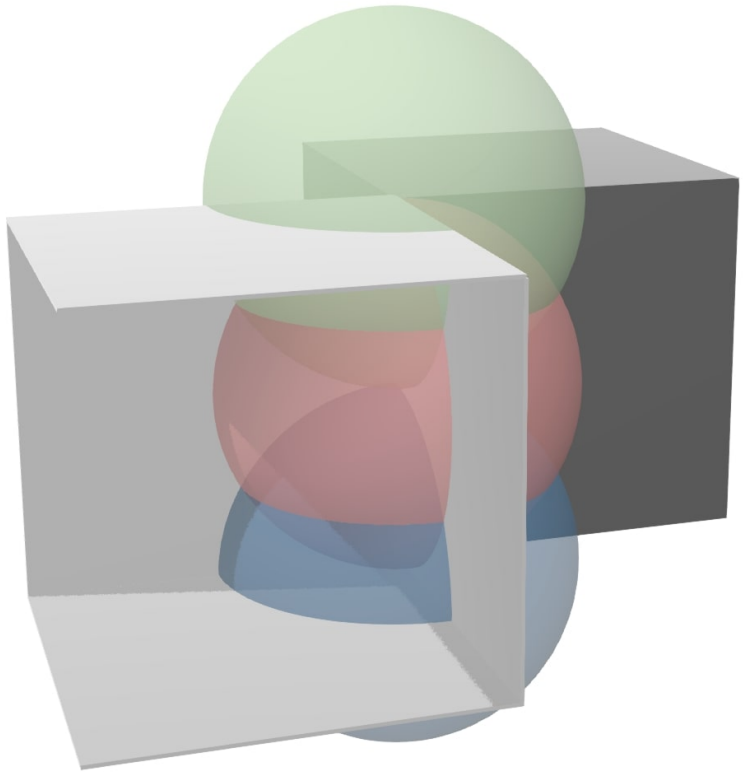

Figure 2.5: Illustration of some of the partial spheres that reside directly over the edge of the base cube.

Lastly, the number of spheres that are on the exterior of the base cube corners must be counted. For a single corner, there is only one sphere that must be addressed. That single sphere has an eighth of a sphere that lies directly over the corner of a base cube such that $C_c \coloneqq 1/8$.

These three exterior base cube counts of the faces, edges, and corners (C_f, C_e, C_c) can be multiplied by their respective occurrences and summed up to find the total number of spheres that exist on the exterior of a single base cube. This enables the calculation of the number of spheres that will exist once many base cubes are placed inside the cubic container.

Sphere Stuffing Equation

Given the interior number of spheres C_i and the exterior number of spheres, C_f, C_e, and C_c, we can use volumetric attributes of the base cube to create an equation for the upper bound of total spheres stuffed into a larger cubic container. The upper bound for the number of spheres that can be stuffed into a large cubic container given by the already defined base cube is,

$$A_S = 1\left(\frac{b}{2r}\right)^3 \cdot C_i + 6\left(\frac{b}{2r}\right)^2 \cdot C_f + 12\left(\frac{b}{2r}\right)^1 \cdot C_e + 8\left(\frac{b}{2r}\right)^0 \cdot C_c, \tag{2.2}$$

where b is the side length of the large cubic container. The $b/2r$ term determines how many base cubes fit in each dimension of the cubic container, and the following C coefficient shows how many spheres to count for a single base cube for that dimension of the cubic container.

Equation (2.2) is able to count the total number of spheres that fit in a cubic container according to the predefined base cube. The meaning of the terms in the equation are as follows. The $1\left(\frac{b}{2r}\right)^3 \cdot C_i$ term counts how many base cubes fit within the cubic container and then multiplies by C_i which is the total number of spheres that are associated with the interior of a base cube. The $6\left(\frac{b}{2r}\right)^2 \cdot C_f$ term counts the number of base cubes that are fit against the plane of each face of the cubic container. The C_f coefficient then accounts for the number of spheres for each face of those base cubes. The $12\left(\frac{b}{2r}\right)^1 \cdot C_e$ counts the cubes at each of the cubic container's edges. Multiplying by C_e then counts up the spheres associated with the edge of each of those base cubes. Finally, the $8\left(\frac{b}{2r}\right)^0 \cdot C_c$ counts how many spheres must be counted at the four corners of the cubic container. Summing up the resulting number of spheres from these terms gives A_S. This term is the maximum number of spheres that can possibly be stuffed into a larger cubic container with side length b.

Knowing A_S enables the use of RC in a boundary space (cubic container) where the upper bound of the maximum number of possible clusters is known. The use case in this thesis is for multiple agents measuring from a cubic boundary space. Knowing the size of the boundary space and the kernel being used is required to compute this upper bound. Note that this is an upper bound and not an exact computation of the number of inducing points that will be placed in the space. Since some may use RC for other use cases besides scenarios in three dimensions, I provide a two dimensional bound as well as an unproven approximation of the exact number of circles that can fit into a square.

2.2.3 Two Dimensional Radial Clustering Upper Bound

Now I present the guaranteed upper bound for the max number of clusters possible for radial clustering in two dimensions. The maximum number of inducing points that can be selected in a two dimensional square boundary space is a simplification of the 3D Equation (2.2). Switching from volume to surface area, the total surface area associated with each portion of the square must be included (i.e. face, each edge, and each corner). The portion of each circle that overlaps over the single face can summed up and denoted by $S_f = 4(\frac{1}{4}) + 4(\frac{1}{2}) + 1(1) = 4$. The overlapping surface area for an edge can be defined as $S_e = 2(\frac{1}{4}) + 1(\frac{1}{2}) = 1$, and each corner $S_c = 1(\frac{1}{4})$. Counting base squares in the larger square gives the overall equation,

$$A_C = 1\left(\frac{b}{2r}\right)^2 \cdot S_f + 4\left(\frac{b}{2r}\right)^1 \cdot S_e + 4\left(\frac{b}{2r}\right)^0 \cdot S_c, \tag{2.3}$$

where A_C denotes all the circles stuffed into any large square with side length b and circle radii r.

Equations (2.3) and (2.2) give an upper bound. Both equations will give the **exact** number of spheres or circles that can be stuffed into the larger container when the radius r of each circle is a factor of the side length b. However, these equations provide a conservative estimate on number of spheres/circles in the cubic container when r is not a factor of b. While difficult to prove exact closed-form solutions for 2D and 3D, I provide an equation

for the 2D case which provides an approximation for a less conservative number of circles that can be stuffed into any sized square. While not theoretically guaranteed, it works well in practice and can be checked in conjunction with the aforementioned upper bound. This equation is outlined in the appendix Section A.

2.3 Event-Triggered Sparse Message Passing

Adaptive inducing point selection through RC and event triggering is pivotal for the decentralized multi-agent system to dynamically search the boundary space and communicate accurate and efficient inducing points between multiple agents. It is desirable to only re-cluster the measured points when the measurements begin to change significantly (have a high kernel dissimilarity from other measurements). If an agent is re-sampling from an area that is already adequately described by an inducing point, it is redundant and inefficient to re-cluster. Therefore, I implement an event-trigger where agents perform RC when measured points are no longer being correctly summarized by the current number of inducing points and their locations. Similar to [11] but independently derived, I check the most recent measured point's column in the K_{uf} matrix to determine if re-clustering should occur. These values show how spatially correlated the most recent measured point is to all existing inducing points. If the maximum value of this vector (i.e. the spatial correlation value of the measured point and the inducing point closest to it) is smaller than a user defined similarity threshold ρ, then re-clustering is triggered. This event-trigger ensures all new measured points are summarized by an inducing point with a similarity of at least ρ. It also allows inducing points to shift towards better summary locations given the new distribution of measured data. The event-trigger ensures no time is wasted on unnecessary clustering, which is vital for real-time systems.

Using the RC algorithm allows real-time SGPR systems (single agent or multi-agent) the ability to adapt the number of inducing points as new data is received. This approach is robust to edge case sampling cases and is outlier resistant (unlike [11]) because if new measurements would be better described by a different inducing point, RC has the ability to re-cluster. This re-cluster ability allows for the agent to have the best inducing point distribution according to the current data. The work done in [11] does not get updated with the current measurement information.

Chapter 3

Decentralized Framework

This chapter outlines the full decentralized sparse Gaussian process regression (DSGPR) framework that enables multiple agents to perform environmental modeling of a square or cubic boundary space. There are three typical architectural approaches to working with multi-agent systems: centralized, distributed, and decentralized. I implement a decentralized GP framework due to advantages in system robustness under degraded communications or node failure. To fully understand the advantages and disadvantages of a decentralized approach, I summarize the different architectural approaches below.

Centralized systems have a main server/computer that receives all measurements (information) from each individual agent. Once it has received those measurements the central computer will complete all the computations and decision making, eventually relaying commands back to each agent. When completing tasks, such as environmental modeling, the centralized computer provides the best model, solution, and decisions since it receives and uses all the gathered information. In this scenario, each autonomous agent essentially becomes a mobile sensor. Centralized systems can fail when the central computer is down, or when there are message passing bottlenecks.

Distributed frameworks such as [9, 18, 22, 40] are formulated such that each agent computes a piece of the model taking computational load off of a central computer. This is advantageous when the agents must solve computationally complex tasks. Agents send the solved piece of the overall problem back to the central computer. Similar to the centralized case, the central computer will assemble all the distributed pieces back together for decision making and modeling. All agents still receive commands from the central computer. One significant issue with distributed frameworks is the possibility of a single agent causing a system wide failure. Distributed systems are also prone to central computer failure, similar to the centralized case. And there can be bottleneck issues when high loads information must be communicated with the central computer.

Decentralized frameworks enable each agent to perform sensor measurements, model computations, and decision making without having to rely on other agents or centralized computers [9, 22]. Agents communicate their individual information (measurements, results, or decisions) with each other directly. Decentralized systems are robust to single agent failures (including a central computer failure) and can potentially mitigate communication bottlenecks. This is especially advantageous when dealing with autonomous mobile agents (such as unmanned aerial vehicles) when communication may be unreliable and individual agents have high probabilities of failure.

While there are some clear benefits and advantages to choosing decentralized systems, there are also unique challenges that arise. Decentralized agents are subject to some common data fusion problems such as the whispering hall problem (i.e. remnants of past measurements

being perpetuated when no agent has actually measured it recently), double counting, and accuracy degeneracy. There are also memory and computational constraints unique to decentralized systems that must be addressed.

DSGPR mitigates these decentralized challenges in a way that provides better computational efficiency and inducing point flexibility as compared to other decentralized GP models. As noted in Section 1.1, the work from [1] avoids sensor fusion problems by passing local models and creating artificial measurements from those models. The creation of artificial measurements reduces accuracy while increasing computational complexity (note the discussion on computational complexity in Section 3.3.1). In comparison to [1], which samples a large number of measurements, DSGPR decreases the computational complexity. The work done in [39] creates a decentralized system using online local models based on the prior work of [3]. They use a product of experts model between local online SGPR models to predict over a space. This means that the density functions of each local model are multiplied together for prediction. However, the product of experts do not compute a global covariance between the local agents. Agents can use global covariances for further use in other objectives (such as informative path planning, etc.). DSGPR provides a global covariance that can be used for secondary tasks, such as path planning. The work done in [17] merges multiple local models using an information version of their local covariance. They then add and subtract information matrices to avoid the mentioned data fusion issues of double counting etc. While efficient at scaling, their decentralized approach requires a set or fixed number of inducing points for their local models. My DSGPR framework mitigates this issue by using RC as explained in Chapter 2.

In this thesis, I also chose to implement a decentralized system due to the aforementioned advantages of robustness and I have developed a method that mitigates the challenges with data fusion and efficient message passing. This DSGPR framework is outlined in Section 3.1 while the equation for computing the global model is derived in Section 3.2. I show how the challenges of data fusion are mitigated by the structure of the message passing in the DSGPR framework and how the message passing memory cost can be reduced through a heuristic in Section3.3.

3.1 Decentralized Framework: Outline

The DSGPR framework, as depicted in Figure 3.1, consists of three main pieces: local SGPR models, message passing, and a global model. Each piece is vital in making DSGPR a viable real-time option for a MAS performing scalar or vector field estimation.

The local portion of the DSGPR framework is set up to address the whispering hall and double counting data fusion challenges. This is accomplished by having each agent having their own local model according to their own measurement data, which is always isolated from other agent's data. In this way, no local model can be corrupted by other agents because no other agent information is ever considered when an agent is computing its local SGPR model (shown as the green diamond in Figure 3.1).

The agent's local model is sent to neighboring agents using the message passing protocols outlined in Section 3.3 (seen as the yellow boxes in Figure 3.1). DSGPR passes messages between agents in the same neighborhood and between multiple neighborhoods over time (a group of agents that are within message passing range are considered a neighborhood).

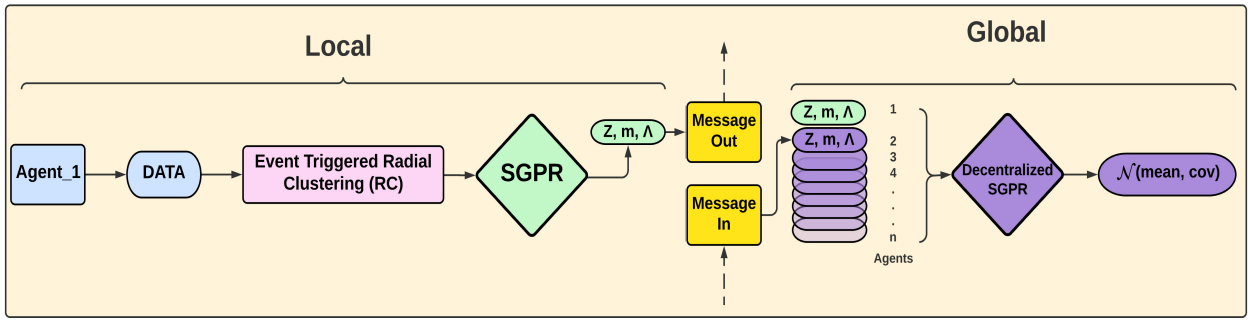

Figure 3.1: Overview of the information flow for a single agent creating first a local GP representation and then, by combining information from neighboring agents, a global representation.

As explained in Section 3.3, agents will relay each other's local models until all agents in the system have the most up to date version of all local models. This message relay is implemented using a max consensus on the timestamps of each agent's local model. This process is described in detail in Section 3.3.2. The advantage of this message passing setup is that agents are still able to make informative decisions when single (or multiple) agent failures occur.

When agents have received other local models, through message passing, they can create accurate global models of the entire boundary space. The global model computation, shown as the purple diamond in Figure 3.1, is derived in Section 3.2.

3.2 Decentralized Sparse Gaussian Process Regression

Each agent's local model is defined by the sparse Gaussian process regression models outlined in [35]. The local models can be defined by the location of each inducing point in two or three dimensional space described by vector $\boldsymbol{z}$, the mean value of the desired scalar or vector prediction for each inducing point represented as vector $\boldsymbol{m}$, and the covariance of those mean values denoted as matrix Λ.

The equations for combining all local models into a single global model were derived by PhD candidate Grant Stagg in 2021 and the work is currently unpublished. Only Section 3.2 was heavily borrowed from the work done by Grant Stagg and is included in this thesis for completeness.

To derive the DSGPR model, we start with the distribution of each agent's inducing points $q_i = p(f_{u_i}|y_i) \sim \mathcal{N}(m_i, \Lambda_i)$. The equation for the predictive posterior of the decentralized SGPR mirrors the predictive posterior for SGPR and is

$$p(f_*|\overline{\mathbf{y}}) = \int \cdots \int p(f_*|f_{u_1},\ldots,f_{u_n},\overline{\mathbf{y}})p(f_{u_1},\ldots,f_{u_n},\overline{\mathbf{y}})df_{u_1}\cdots df_{u_n}, \tag{3.1}$$

where $\overline{\mathbf{y}}$ is the vector of prior measurements and f_{u_i} is the underlying function values for the i^{th} agent's inducing points. We start by deriving the prior for the predictive posterior of f_* which is $p(f_*|f_{u_1},\ldots,f_{u_n},\overline{\mathbf{y}})$. As with other GPRs, we use the kernel function to calculate the covariance of the prior and assume zero mean. Differing from other GPR approaches, we

stack all agents' inducing points into combined variables that now describe the entire system. Each agent's inducing point locations are stacked into a single variable $X_{\overline{\mathbf{u}}}$. The unknown ground truth of each inducing point is denoted as $f_{\overline{\mathbf{u}}}$. The local agents' predicted mean value at each inducing point is described by stacking all the local $\boldsymbol{m}$ values into $\overline{\mathbf{m}}$. Both $f_{\overline{\mathbf{u}}}$ and $\overline{\mathbf{m}}$ are defined as

$$f_{\overline{\mathbf{u}}} = \begin{bmatrix} f_{u_1} \\ f_{u_2} \\ \vdots \\ f_{u_n} \end{bmatrix} \text{ and } \overline{\mathbf{m}} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix}. \tag{3.2}$$

Using these stacked inducing points from all agents, the prior for f_* can be written as $p(f_*|f_{\overline{\mathbf{u}}}, \overline{\mathbf{y}})$ because f_{u_i} is a Gaussian distribution and the joint distribution of all f_{u_i} is also Gaussian. Define the covariance matrix between test points and inducing points to be

$$K_{*\overline{\mathbf{u}}} = \begin{bmatrix} K_{*u_1} & K_{*u_2} & \dots & K_{*u_n} \end{bmatrix}, \tag{3.3}$$

where $K_{\overline{\mathbf{u}}*} = K_{*\overline{\mathbf{u}}}^\top$ and K_{*u_i} is the kernel function between the i^{th} agent's inducing points and the test points $k(X_x, X_{u_i})$. And defining the covariance matrix between inducing points to be

$$K_{\overline{\mathbf{u}}\overline{\mathbf{u}}} = \begin{bmatrix} K_{u_1u_1} & K_{u_1u_2} & \dots & K_{u_1u_n} \\ K_{u_2u_1} & K_{u_2u_2} & \dots & K_{u_2u_n} \\ \vdots & \vdots & \dots & \vdots \\ K_{u_nu_1} & K_{u_nu_2} & \dots & K_{u_nu_n} \end{bmatrix}, \tag{3.4}$$

where $K_{u_iu_j}$ is the kernel function between the i^{th} and the j^{th} agent's inducing points $k(X_{u_i}, X_{u_j})$. We then calculate the prior using the conditional multivariate Gaussian distribution,

$$\begin{bmatrix} f_* \\ f_{\overline{\mathbf{u}}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{**} & K_{*\overline{\mathbf{u}}} \\ K_{\overline{\mathbf{u}}*} & K_{\overline{\mathbf{u}}\overline{\mathbf{u}}} \end{bmatrix} \right), \tag{3.5}$$

yielding

$$p(f_*|f_{\overline{\mathbf{u}}}) \sim \mathcal{N}(K_{*\overline{\mathbf{u}}}K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}f_{\overline{\mathbf{u}}}, K_{**} - K_{*\overline{\mathbf{u}}}K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}K_{\overline{\mathbf{u}}*}), \tag{3.6}$$

where K_{**} is the covariance matrix from Equation (1.5). Equation (3.6) gives us the prior distribution of function values at the test point locations conditioned on the prior distribution of function values at the inducing point locations. Because f_* and $f_{\overline{\mathbf{u}}}$ are both multivariate Gaussian distributions, we can write Equation (3.6) such that we will be able to predict the mean values of f_* as a linear transformation from $f_{\overline{\mathbf{u}}}$ through the equation

$$f_* = K_{*\overline{\mathbf{u}}}K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}f_{\overline{\mathbf{u}}} + \eta \tag{3.7}$$

where $\eta \sim \mathcal{N}(0, K_{**} - K_{*\overline{\mathbf{u}}}K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}K_{\overline{\mathbf{u}}*})$.

Because we are combining multiple agent's inducing points, we must account for the covariance already existing in each agent's local model f_{u_i}. To account for these covariances,

we use a temporary matrix $B = K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}$ and partition it into submatricies

$$B_{\overline{\mathbf{u}}\overline{\mathbf{u}}} = \begin{bmatrix} B_{u_1u_1} & B_{u_1u_2} & \dots & B_{u_1u_n} \\ B_{u_2u_1} & B_{u_2u_2} & \dots & B_{u_2u_n} \\ \vdots & \vdots & \cdots & \vdots \\ B_{u_nu_1} & B_{u_nu_2} & \dots & B_{u_nu_n} \end{bmatrix}. \tag{3.8}$$

Note that $B_{u_iu_j} \neq K_{u_iu_j}^{-1}$. Rather, $B_{u_iu_j}$ is a submatrix of the precomputed $K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1}$ that corresponds to the location of $K_{u_iu_j}$ in $K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}$. By expanding Equation (3.7) with Equation (3.8) one can see that f_* is a sum of linear transformations from $u_1, u_2, \dots, u_n$. We can then create a transformation matrix,

$$C_i = \sum_{j=1}^{n} K_{*u_j} B_{u_ju_i}, \tag{3.9}$$

that represents the linear transformation from agent i's inducing points to f_*. We write the linear transformation as $f_* = C_1 f_{u_1} + C_2 f_{u_2} + \cdots + C_n f_{u_n} + \eta$. Because we have information gained from our agent's measured points, we have a local posterior distribution for each f_{u_i} such that $q_i = p(f_{u_i}|y_i) \sim \mathcal{N}(m_i, \Lambda_i)$. If we assume each agents' inducing points are independent, the predictive posterior is a Gaussian distribution takes the form

$$p(f_*|f_{\overline{\mathbf{u}}}, \overline{\mathbf{y}}) \sim \mathcal{N}(K_{*\overline{\mathbf{u}}} K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1} \overline{\mathbf{m}}, \Sigma_{f_*}), \tag{3.10}$$

with

$$\Sigma_{f_*} = \sum_{i=1}^{N} C_i \Lambda_i C_i^\top + K_{**} - K_{*\overline{\mathbf{u}}} K_{\overline{\mathbf{u}}\overline{\mathbf{u}}}^{-1} K_{\overline{\mathbf{u}}*}. \tag{3.11}$$

From Equation (3.10), the prior does not depend on $f_{u_1}, \dots, f_{u_n}$, so we can remove it from the integration in equation (3.1). This gives the new predictive posterior,

$$p(f_*|\overline{\mathbf{y}}) = p(f_*|f_{\overline{\mathbf{u}}}, \overline{\mathbf{y}}) \int \cdots \int q(f_{u_1}, \dots, f_{u_n}|\overline{\mathbf{y}}) df_{u_1} \cdots df_{u_n}. \tag{3.12}$$

Due to the law of total probability the integral term becomes 1 and the predictive posterior is $p(f_*|f_{\overline{\mathbf{u}}}, \overline{\mathbf{y}})$ who's equation is given in (3.10).

The derivation of the predictive posterior assumes the distribution of inducing points for each agent are independent. Because of the exponential nature of the kernel function, each agent's distributions can never be truly independent of another agent's inducing points. However, if an agent's inducing points are spaced far enough apart from another agent's points, the distributions can be approximately independent.

3.3 Message Passing

Cooperating agents must be able to efficiently and effectively communicate their local model's information with each other in order to compute their global models. I present the details on how message passing occurs in the DSGPR framework. First, Section 3.3.1 discusses the memory considerations inherent with peer to peer message passing. The actual protocols for

each agent in the DSGPR framework is outlined in Section 3.3.2. Lastly, two heuristics are introduced that provide efficient message passing. Section 3.3.3 discusses a covariance matrix reduction heuristic to reduce the size of each message being sent. Section 3.3.4 presents a heuristic using the Bhattacharyya distance that controls the frequency of messages being passed. A discussion on how a user can manipulate each heuristic occurs at the end of this chapter (subsection in Section 3.3.4). Each of these pieces enables the DSGPR framework to be decentralized while maintaining viability for its use in real-world scenarios.

Passing local models between agents comes with two memory considerations that must be addressed. These are the cost of storing models in memory (actual hardware on the drones) and the total cost of sending messages between agents (network bandwidth), both of which are limited. A discussion regarding the size of the models as sampling continues in time will be insightful for the justification of choosing my message passing protocols.

3.3.1 Message Passing Memory Considerations

Reducing message sizes is a challenge when completing cooperative estimation with GPs. The work done by [1, 17, 40], all make the choice to send representations of their prediction models instead of sending the exact measurements, stating that it is more memory efficient. Instead, they send the covariance matrix of the inducing points (or the covariance of the sampled measurements). On the other hand, the authors in [21] send the measurements and have a message passing complexity of $O(n)$. While there are pros and cons to sending exact measurements, sending the model representation also has cons that are not highlighted in the prior research. The covariance scales at a rate of $O(n^2)$. If the inducing points are allowed to increase in total number, then the covariance matrix describing the inducing points can become too large, causing issues for message passing in a real-time system.

A quick example illustrates the drawback of sharing the covariance matrix of induced points among cooperating agents. Assume an environmental sampling scenario where A agents are operating within message passing range (a single neighborhood), and each agent is sampling a scalar field. Each agent's local model will consist of the $\boldsymbol{z}$, $\boldsymbol{m}$, and Λ values mentioned in Section 3.2. If m is the agent's individual number of inducing points. The total memory (MB) it takes to describe a local model is given by the equation,

$$T_{MB} = \frac{(3m + m + \frac{1}{2}(m^2 + m))}{1024^2}, \tag{3.13}$$

where $3m$ represents all floats it takes to describe $\boldsymbol{z}$ in three dimensional space, m represents the predicted scalar mean values for each inducing point, and $(m^2 + m)/2$ is the most efficient way to collapse a covariance matrix (taking advantage of symmetry). Given an example where $m = 10,000$, then $T_{MB} = 47.7$ MB. That is the total memory (MB) for a single agent. If an agent is required to send all the local models it knows about, then it would be the sum of each agent's T_{MB}. This growth scales with the number of agents A, which clearly isn't sustainable for message passing within the network of a real-time system with time steps in the seconds range. However, time steps that are minutes long would make these message passing memory issues negligible.

Note that even when the number of inducing points is small, the strain on the message passing network is not negligible if there are desires to scale to tens of agents or especially hundreds of agents, i.e. if each agent's $m = 1,000$ and $T_{MB} = 0.48$ MB, when $A = 100$ then the total memory of all combined models would be 48 MB. The worst case scenario would be all agents sending their models at the same time, in this case, even for a small m value (e.g., $m = 100$), the required bandwidth to handle $A > 1000$ would surpass most cellular data rates [25] and would require high data rates for any WiFi connections (i.e., ≈ 400 Mbps).

The aforementioned papers make the assumption that efficient message passing can occur when passing SGPR model parameters ($\boldsymbol{z}$, $\boldsymbol{m}$, Λ), but they do not mention the scaling rate of Λ. Note that most of these implementations do not have adaptive inducing points which is what gives the undesirable squared scaling of Λ in this case; however, these papers still don't account for statically choosing a number for m among their local models which still becomes expensive when trying to represent a large space. Compared to passing SGPR model parameters, passing exact measurements scales linearly per agent, i.e. $(3w + w)/1024^2$ where w is the total number of measurements for a vehicle in 3D space. Passing measurements only becomes inefficient compared to sending a model when the SGPR model has reached its upper bound u_b as derived in Section 2.2.2. Note however, that the number of measurements would be quite high before this occurs even for a small value of m. An example goes as such, given a small upper bound for m of 100 (i.e., both m and u_b are 100), sending measurements is less memory intensive until the number of measurements passes 1363. Calculated by $w = (3m + m + (m^2 + m)/2)/4$. Note that this scenario only occurs if all the measurements had to be sent in one fell swoop. The most likely scenario is that agents would have the advantage of only needing to send a measurement once if they were in message passing range of other agents.

This analysis begs the question of why not always use the measurement passing approach instead of sending models? The main reason is due to the high computational cost required to compute models based on a large number of measurements instead of computing based on local models. The computational complexity on the global model calculation of the DSGPR framework presented in this work is $O(M^3)$ where M is all combined inducing points from all agents. Assuming an approach where all measurements are passed between all agents, then the vanilla exact GPR used in that case would have a computational complexity of $O(N^3)$ where N is the combination of all agent measurements. Another approach to only using measurements would then be to cluster those N measurements for each agent using RC from Section 2. The RC algorithm has a computational complexity of $O(N^2)$ so the combined clustering and SGPR model computational complexity for measurements for each agent would be $O(N^2 + NM^2)$ while each agent in the model case has a total computational complexity (including clustering) of $O(n^2 + M^3)$ where n is the measurements for an individual agent and $M \ll N$. Also, the overall number of messages could potentially be greater for sending individual measurements between agents than the number of messages sent when only sending local models (this concept will be domain dependent).

With these pros and cons in mind, I provide a framework for an efficient approach for passing local models between agents. This message passing framework has three main components as follows:

1. Message passing protocols that enable agents to perform maximum consensus on the local models of the system (Section 3.3.2).
2. Agents send messages of reduced size through the covariance reduction heuristic (Section 3.3.3).
3. Agents send fewer messages overall using the Bhattacharyya distance heuristic (Section 3.3.4).

The first item ensures that agents are robust to switching between neighborhoods where certain agents may be out of communication range for long periods of time. All agents are also guaranteed to have the most up to date system information (once maximum consensus has been performed). The last two components provide the ability for users to reduce message passing frequency and message size according to their network bandwidth needs (i.e., when message passing could be at its peak).

3.3.2 Message Passing Protocols

The message passing protocols are query based instead of broadcasting dependent. When an agent's neighbor has an updated or new model, it will query its neighbor for this model instead of agents broadcasting their own updated models to all neighbors. The following protocols determine when the agent should query a neighbor agent for a local model and how this query process is implemented.

The message passing process is described by three steps as seen in Figure 3.2. Each time step every agent will broadcast a ping. Agents can also selectively send query messages or local model information if it was queried from a peer agent.

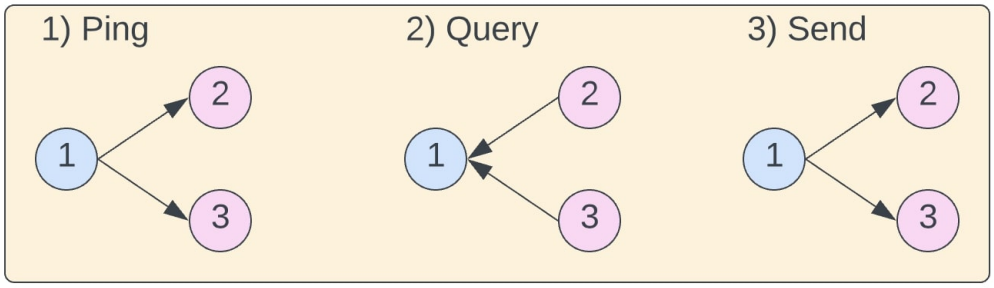

Figure 3.2: Message passing process where agent 1 will ping all neighbors in their neighborhood, then each agent that received a ping will potentially query agent 1 for any agent's updated local model, and finally agent 1 will send a message containing the queried model to agents 2 and 3.

First: Ping

Each agent will broadcast a ping to any agent that is within communication range. The ping message contains the agent's current local model timestamp, a list of all peer agent local models it has in memory (each agent is given a unique ID), and the timestamps indicating when the local models were updated.

Second: Query

When an agent receives a ping message, it will determine if there are newer local models that it has not yet received. If so, it will query its neighbor (who has the updated model) to request the information. As a part of the query, the requesting agent will send the most recent timestamp it has for the model being queried such that the agent who is receiving the query can send a reduced Λ matrix according to the heuristic described in Section 3.3.3.

Third: Send

Any agent that has received a query will send the requested model. An agent may send any local model it has stored (not just its own local model).

Note that steps one and two will in general only take on average 100 ms total for an average 4G network (due to the small ping/query message sizes), while step three's time will vary depending on message size [25]. Because steps one and two are so quick, it is reasonable to count steps one, two, and three as one time step.

To reduce the amount of messages passed through the network, each agent only updates its local model timestamp when one of the following conditions are met:

- re-clustering occurs,
- the Bhattacharyya distance change threshold is surpassed (as explained in Section 3.3.4), or
- a first time agent connection occurs.

When any of these conditions are met, the agent will update the timestamp of its model to the current time. This will trigger all agents within its message passing range to query for the updated model. The updated model will then be passed around until max consensus of all model timestamps has been reached. When agents are not within message passing range (the same neighborhood), there is a chance that formerly isolated agents will meet for the first time. I refer to this first time meeting as a first connection. When this occurs, the protocol is to send that formerly isolated agent the most up to date models available. This includes the self model. This will then trigger any currently connected agents to query for the newly updated model.

These message passing protocols can be applied to a drone system that is on its own local area network (LAN), equipped with individual cellular networks, or with direct WiFi capabilities where data rates can reasonably handle 100 - 200 Mbps. Any issues related to network congestion caused by agents sending messages at the exact same time is left as a networking problem and is not addressed in this work.

These protocols allow the agents to be fully decentralized. Here fully decentralized refers to the idea that agents can receive all the local models in the system under any communication network topology given that the topology forms a connected graph over time. Furthermore, each agent is given the ability to act completely autonomously without the need for a centralized node that provides direction or information. The main component of the message passing framework that enables this decentralization is the maximum consensus on the timestamp of the local models.

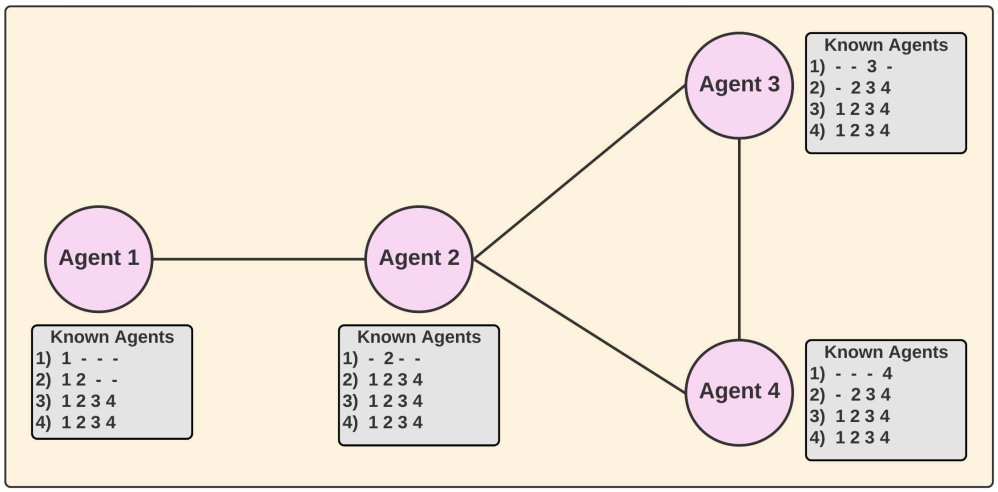

Figure 3.3: Decentralized agent graph. The edges connecting agent nodes denote that two agents are within the same neighborhood (message passing range).

The maximum consensus algorithm implemented in this framework allows agents to receive the local information of other agents despite being outside of their message sharing neighborhood. Being outside of communication range is a common issue when multiple agents gather measurements over long range distances. Consensus is reached in the network when each node (agent) has received the maximum model timestamp (and it's corresponding model) for each agent, even when they are out of message sharing range. Figure 3.3 shows a simple example of this at time step 4. Each node in the graph is an agent, the edges connecting the nodes denote the ability to pass messages between agents. The grey boxes show which models each corresponding agent has stored in memory at different time steps. Agent one is not within range of agents three and four. However, agent one is connected to agent two. Agent two becomes the relay between the agents that are not within message passing range. Over the course of four time steps, each agent's model is propagated throughout the agent graph.

Successful consensus can only be achieved when each node in the graph is reachable from every other node, i.e. the network topology graph, which is directed, must be connected.Therefore, an assumption of connectivity is required for the network to achieve maximum consensus. However, even when agents are not connected for a time, once they return to within communication range, they will be updated with all the modified/new models of their peer agents. Then their global predictions will be as accurate as the other agents of the system. Therefore, connectivity is only required when the agent needs to provide the most accurate global prediction possible.

3.3.3 Covariance Matrix Reduction

The viability of this entire framework for use on smaller bandwidth networks may be dependent on the size of the messages needing to be passed. Covariance matrix reduction is a heuristic designed to alleviate the most network intensive (in terms of message size) element when passing the agent's local model, the covariance of the inducing points Λ. I developed an

adaptive method that reduces the number of elements shared from the Λ matrix based upon user/domain knowledge of a reasonable kernel length scale (distance) where points are essentially independent of each other. Note that I assume memory on-board the autonomous agent will be relatively cheap, i.e. less consideration is given for the cost to store information in memory. The focus of this method is on reducing the amount of information that must be shared across a network in real-time.

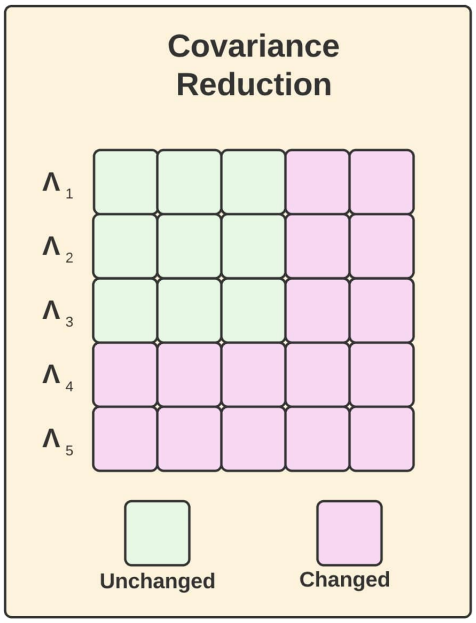

Figure 3.4: Sketched representation of covariance reduction of the Λ matrix.

A common approach to sending covariance matrices over a network is to send the diagonal and upper triangular elements of the matrix, since it is symmetric. This yields a memory cost of $(n^2 + n)/2$ for a $n \times n$ covariance matrix. I further limit memory cost by only sending values of Λ that have changed significantly from the last time Λ was sent. This occurs because agents send updated versions of their own model when their model has changed significantly due to their continual measurements (the heuristic for sending an updated local model is explained in Section 3.3.4).

Equation (1.7) shows that Λ is defined using a kernel on the inducing points and measured points. Under the assumption that the kernel being used is a type of stationary distance kernel, there exists a distance where the kernel similarity score (weight) of new inducing and measured points have negligible effects on the previous covariance values. When this is true, there is no need to re-send the original/unchanged values of Λ (assuming the neighbor agent has received the prior Λ matrix). This significantly reduces the amount of memory that is sent between neighboring agents.

Each agent determines how much of Λ to send based on two pieces of information, the querying agent's last received Λ and the user defined low significance factor, L_{SF}. The formulation and logic for the heuristic reducing Λ can be seen in the following toy example.

I illustrate how much message passing memory can be saved using a simplified example where there are two measured points (denoted by the small green dots in Figure 3.5) and

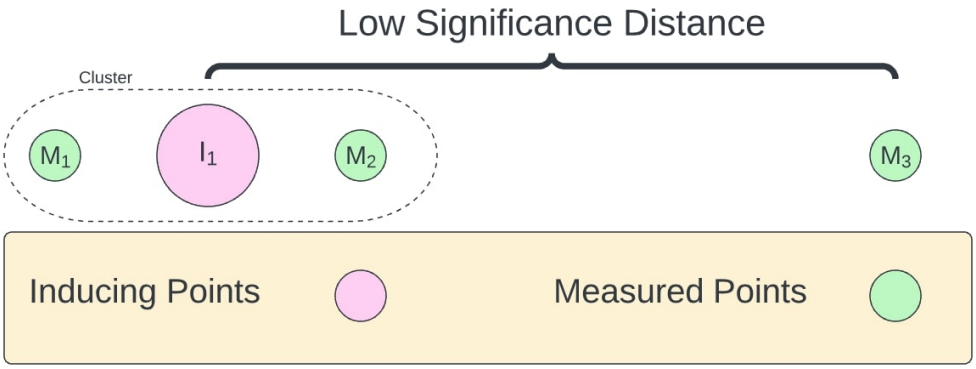

Figure 3.5: Example showing the scenario in which Λ doesn't change when new measurements are far enough away.

an inducing point (shown as the larger pink dot in Figure 3.5) that is representative of the measured points M_1 and M_2. We create an example that shows when an agent measures a new point (M_3) that is "far" enough away from a known cluster of prior measured points (M_1, M_2) and their inducing point I_1, then M_3 won't affect the Λ values of I_1. This is due to exponential nature of RBF kernels. As the distance increases, the similarity between those two points decreases which lowers the weight that the two points have on each other. We can then take a user defined distance at which two points become essentially independent. The user chooses this distance by choosing a low similarity threshold that can be used to compute a distance between points as was done with κ in Section 2.1. Using this general computed distance between points, the toy example shows the change in Λ covariance values related the inducing point I_1 caused by the new measurement M_3. Define λ_1 as the first covariance entry (a scalar value) of I_1 in Λ before M_3 is measured, and define λ_2 as the first covariance value of I_1 in Λ after M_3 is measured. Then, $D_C = |\lambda_1 - \lambda_2|$, where D_C is the absolute value of difference. This shows the amount of impact M_3 has on I_1. The difference changed value D_C can then be used in my covariance reduction heuristic to help reduce the number of Λ rows sent in a message.

The D_C value is used for comparison in the following covariance reduction heuristic. It is a heuristic in that it is not exact, but gives a user adjustable approach for determining unneeded information in the Λ covariance matrix. The value D_C gets checked anytime an agent is queried. Assume Agent 1 has a model from Agent x, where the model is represented as B_x. B_x is made up of mean values (m) and covariance values Λ as described in Section 3.2 where x denotes whose model it is. The inducing point covariance matrix in this example is denoted as Λ_{B_x} with time step t, thus written as $\Lambda_{B_{x_t}}$. Assume Agent 2 has a more up to date local model of Agent x, with the $\Lambda_{B_{x_{(t+1)}}}$ matrix. Therefore, Agent 1 queries Agent 2 to send it this more up to date version of model B_x. In the query, Agent 1 notifies Agent 2 that it has already received the B_{x_t} version of the model. Agent 2 can then look at all the time stamped versions of the model K it has saved (note that each agent will store all versions of each model it encounters, relying on the assumption that on-board memory is relatively cheap). In this example, Agent 2 has two versions, B_{x_t} and $B_{x_{(t+1)}}$. Agent 2 then performs the covariance matrix reduction heuristic between $\Lambda_{B_{x_t}}$ and $\Lambda_{B_{x_{(t+1)}}}$. Any row of

L_{SF}	**0.05**	**0.10**	**0.15**	**0.20**	**0.25**
Total Sizes in MB	50.04	29.23	19.81	15.18	12.48
Total Messages	17964	11688	8508	6712	5512

Table 3.1: Covariance reduction and Bhattacharyya heuristic effectiveness. Message sizes and message frequency decrease as the similarity score threshold defined by the user increases. Five separate tests run with five fully connected agents.

$\Lambda_{B_{x_{(t+1)}}}$ where each value in the row has changed less than D_C from the previous $\Lambda_{B_{x_t}}$ matrix will not be sent. Agent 2 sends this reduced matrix, and Agent 1 uses the old values from $\Lambda_{B_{x_t}}$ and the received $\Lambda_{B_{x_{(t+1)}}}$ covariance values to create a $\Lambda'_{B_{x_{(t+1)}}}$ matrix so it can have the most up to date information for Agent x.

The effect of this covariance reduction heuristic on message memory size is seen in the five test results shown in Table 3.1. This table illustrates that the total number of messages and message size throughout a 1000 time step simulation decreases as L_{SF} increases.

3.3.4 Message Frequency Heuristic: Bhattacharyya Distance

As mentioned in Section 3.3.2, an adaptable message passing frequency is another strength of this decentralized system. Messages are communicated when an agent's local model has changed significantly, as determined by this Bhattacharyya distance heuristic. Using the toy problem from Section 3.3.3, the user can define a low significance factor L_{SF} to control how often messages are sent.

In the first step (ping) of the message protocol, each agent will find the Bhattacharyya distance between their most recently computed local model ($\boldsymbol{m_r}$ and Λ_r) with the last broadcasted model ($\boldsymbol{m_b}$ and Λ_b). To find the distance between these two models, I use the well-known equation for Bhattacharyya distance [19]

$$D_B = \frac{1}{8}(\boldsymbol{m_r} - \boldsymbol{m_b})^T P^{-1}(\boldsymbol{m_r} - \boldsymbol{m_b}) + \frac{1}{2} ln \left(\frac{det(P)}{\sqrt{det(\Lambda_r) det(\Lambda_b)}} \right), \tag{3.14}$$

where

$$P = \frac{\Lambda_r + \Lambda_b}{2} \tag{3.15}$$

is a new covariance matrix being the average of Λ_r and Λ_b. With D_B computed, I can then compare it to the pre-computed distance between the two distributions described in the toy example in Section 3.3.3. Referencing the toy example again, there exist two distributions to compare: the distribution that existed with M_1, M_2, and I_1, and the second distribution that added M_3. The Bhattacharyya distance between these two toy distributions is denoted as D_T (note that D_T is computed once before the agents start measuring). The distance D_T shows when a new measurement is no longer close enough to be described well by old data (because the data is too far away for there to be any dependence between them). When $D_B > D_T$, it is reasonable to determine that the old model ($\boldsymbol{m_b}$ and Λ_b) no longer describes the new model ($\boldsymbol{m_r}$ and Λ_r) well enough for prediction, which then triggers the agent to

inform other agents that its model is ready to be queried; i.e., other agents now know that an updated model is available. Thus the agent will update the timestamp for its own model (alerting neighbors in the ping step) which will then trigger maximum consensus to occur for all neighboring agents and thus the whole system.

The effect of this Bhattacharyya distance heuristic on the total messages passed is seen in a simple experiment conducted where the L_{SF} value is increased as seen in Table 3.1. As L_{SF} is increased, the number of total messages decreases as expected.

Impact of L_{SF}

The advantage of using these heuristics is that it allows the user to determine how aggressive they are in their memory saving approaches. Increasing the value of L_{SF} is used to reduce the Λ covariance matrix that needs to be sent in a message described in Section 3.3.3 and, and increasing L_{SF} reduces how often new messages need to be sent as described in Section 3.3.4. The higher L_{SF} gets, the fewer numbers of Λ entries get sent, according to the covariance matrix reduction heuristic, and the less often agents will update the timestamp on their own models, which will cause less messages to be sent overall. Thus allowing lower data rate message passing such that DSGPR can be viable for decentralized modeling. However, these heuristics allow for user flexibility if their network bandwidths can handle large amounts of data, then users can lower L_{SF} and higher numbers of the Λ covariance rows will be sent between agents, as well as an increased rate at which each agent will update their model's timestamp. All of this is determined by the user defining what significance levels (L_{SF}), which may be chosen to best fit the application scenario and hardware being utilized. Note that the usage of L_{SF} is completely kernel dependent. Domain knowledge is required to make the appropriate judgements on which kernel is to be used and what should be chosen for the length scale.

Chapter 4

Results and Analysis

Extensive testing shows the viability of the DSGPR framework presented in the above chapters. This chapter shows the strengths and weaknesses of this approach and provides insight into the specific use cases where this framework is best suited. In this chapter, we first describe the simulation setup. Second, DSGPR is compared to exact GPR and variational free energy sparse Gaussian process regression (VFE-SGPR), which is explained below. This comparison consists of showing how DSGPR is more computationally efficient in inducing point selection (Subsection 4.2.3) and in re-measuring areas of the boundary space (Subsection 4.2.2) while maintaining competitive predictive accuracy (Subsection 4.2.1). Third, the Monte Carlo (MC) runs are presented and analyzed showing viability of this system for some real-world applications. The DSGPR framework is shown to perform well at prediction and uncertainty reduction in Subsection 4.3.1. The computational cost of the entire framework is outlined in Subsection 4.3.2. And finally, Subsection 4.3.3 outlines how the message passing performs such that DSGPR can be considered for real-time systems.

4.1 Simulation Details

In these tests, I simulate agents traveling through two or three dimensional space as seen in Figure 4.1. Agents travel at a constant velocity, with the assumption that they maintain that constant velocity while travelling through any wind-field, i.e. the agents were not deterred or negatively affected by surrounding weather conditions. Agents are tasked with prediction at evenly spaced test points throughout a boundary space. All agents share the same test points for prediction, meaning that each agent is capable of predicting across the entire boundary space (note that agents are not required to have the same exact test points, but I choose this for simplicity). The wind-field was made to match the number of spatial dimensions (2D or 3D) for each specific run. Note that the DSGPR framework is not limited to predicting 3D vector fields, the number of dimensions for prediction can vary from a scalar field to any number of dimensions. Figure 4.1 also shows how the agents have a sphere around them denoting their message passing range. In many large, real-world applications, agents will not always be within communication range of each other (the magenta line connecting two agents shows that they are within in communication range). This was also accounted for in these simulations and is shown in the figure.

Tests ran for a pre-determined, specified number of time steps, where time steps can represents a single second, or be adjusted to varying use cases. As will be seen in the results, the framework is able to handle real-time use cases where a time step is set as $1 - 2$ seconds. Agents measure the wind-field each time step at its current location. Each wind-field measurement has added sensor noise of 0.4 units added to it. The ground truth wind-field

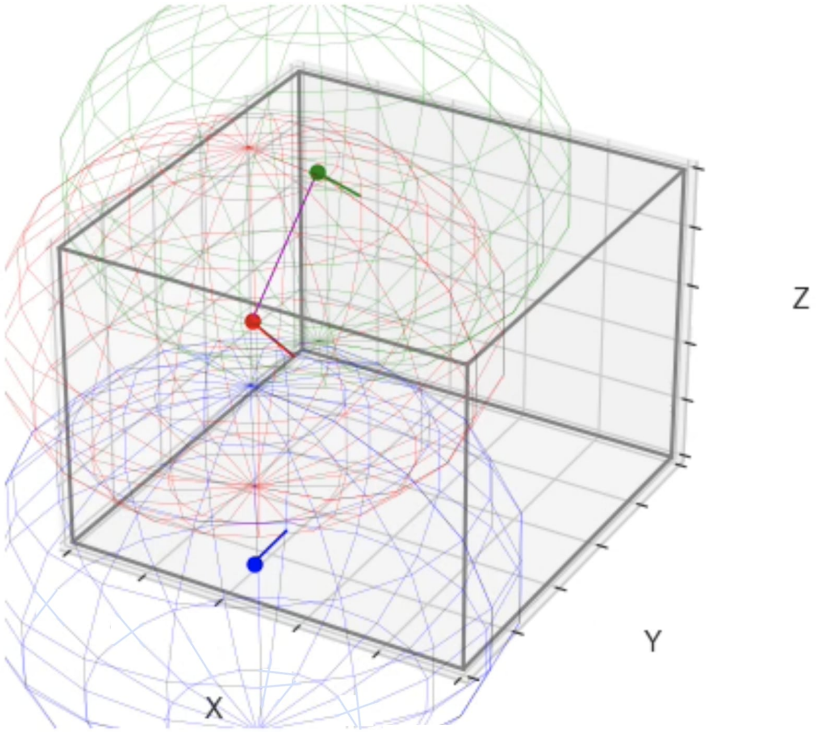

Figure 4.1: Illustration of simulating multiple agents measuring a cubic wind-field and passing messages according to pre-determined message passing ranges.

was simulated using python package gstools [23]. Gstool's core tool is the spatial random field generation. Their process of generation is based on the work done in [16] where a randomisation method is presented for generating random fields. This generation occurs through a specified covariance model or semi-variogram. The wind-field covariance model chosen for these simulations is Gaussian with a variance of 25 and a length scale of 38.

Each agent is initialized in the boundary space at a random starting position. And the agents travel within the bounding area by following randomly generated way-points. Note that uncertainty and MSE results would be improved with intelligent path planners, this is left as a future research direction. The two RBF kernel hyper-parameters presented in Equation (1.3) (σ_f and λ), were tuned offline and set to be fixed for all agent's global and local models. Decentralized online tuning is addressed in other works [3, 17, 39] and is left as a future research avenue for this particular framework.

4.2 DSGPR Model Comparison

The DSGPR framework is compared to an exact GPR framework and a variational free energy (VFE) sparse Gaussian process regression framework (VFE-SGPR) [35]. VFE-SGPR is a popular and seminal SGPR approach where instead of making a sparse approximation on the likelihood like [32] and [31], the approximation occurs on the approximate posterior. VFE-SGPR and exact GPR are used to serve as a comparison to the DSGPR approach, allowing for analysis of the model's strengths and weaknesses.

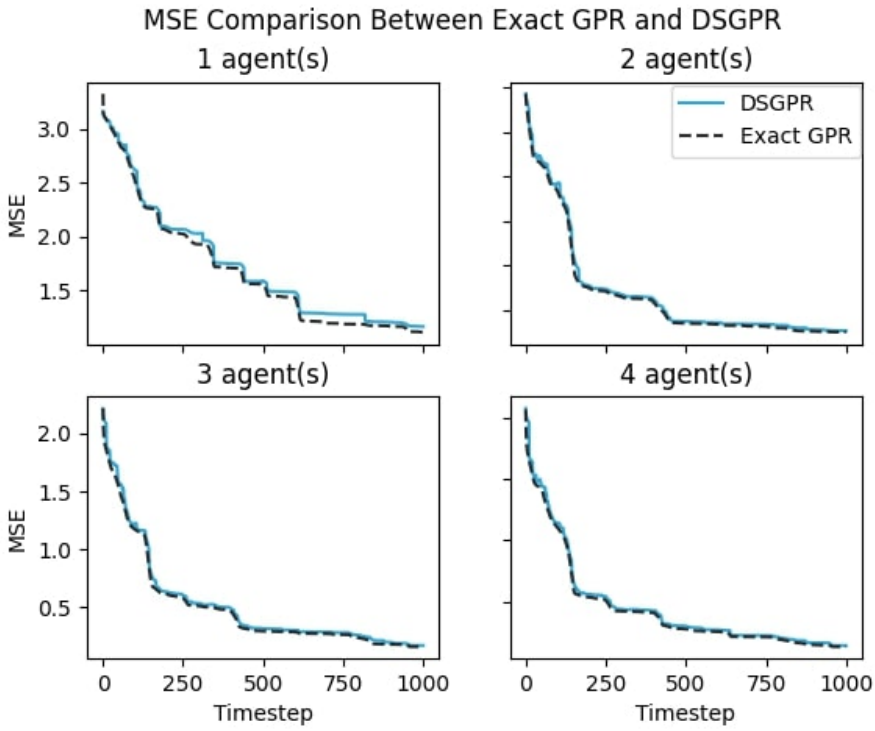

Figure 4.2: MSE comparison between Exact GPR and DSGPR for 1, 2, 3, and 4 agents. DSGPR set with same parameters as MC runs described in Table 4.1. All agents are fully connected for the duration of this test (i.e., message passing density of 1.0, refer to Figure 4.7).

Subsections 4.2.1, 4.2.2, and 4.2.3 show test results regarding the DSGPR framework compared to centralized exact GPR and VFE-SGPR. Subsection 4.2.1 shows how exact GPR compares to varying number of agents in terms of MSE. Subsection 4.2.2 shows how the uncertainty (global trace) decreases over time and how DSGPR is more efficient at decreasing uncertainty. Finally, subsection 4.2.3 shows the advantages of using RC for inducing point selection; the computational cost and MSE are compared between the three models where MSE is scored against the ground truth values of the vector field. These results help show the advantages and trade-offs of DSGPR against these other well-known algorithms.

4.2.1 Exact GPR MSE Comparison

First, I compare an exact GPR model to my DSGPR framework in terms of accuracy. An exact GPR model will always perform better in terms of MSE when compared to sparse models. This is due to sparse models giving up some information (i.e., all the measurement data) for a smaller representation and computation cost. This brings up the question, how much better does exact GPR perform compared to my DSGPR model? To answer this question, a small test was performed between exact GPR and DSGPR.

Given the $O(n^3)$ computational complexity of an exact GP, I do not include exact GP in the many MC runs. However, I show results for a single run with 1, 2, 3, and 4 agents in Figure 4.2, where all agents are fully connected in terms of message passing.

Figure 4.2 shows that as the number of agents increases, there is minimal to no change in the performance of MSE. In the figure, the exact GPR (shown by the dashed black line) performs only marginally better in terms of MSE when compared with DSGPR (shown by the blue line) for agent sizes up to 4. The results certify that this DSGPR framework is comparable to an exact GPR model in terms of MSE while being tractable in terms of computational feasibility for an increasing number of agents as shown in subsection 4.2.3.

4.2.2 Decreasing Uncertainty Efficiently

One key advantage of DSGPR that is seen when comparing to exact GPR and VFE-GPR is the re-sampling ability of this framework. Once an agent has covered a space, all the inducing points are set and don't need to be adjusted.

RC can create these inducing points quickly and effectively with a guaranteed upper bound on the number of inducing points. When coverage has occurred (i.e. no regularly new inducing points), the agents may be tasked with re-sampling from certain regions to reduce the uncertainty. To show this advantage of our framework's re-sampling ability, a test using all models (exact GPR, VFE-SGPR, and DSGPR) was run for 2000 time steps, where a single agent covers the boundary space (20 x 20 x 20) and then re-traces its path again. We can see that the agent is in an area that already has inducing point coverage from time steps 650 - 1250, 1250 - 1510, and 1510 - 2000 because there are no spikes in time from the cost to re-cluster (in other words it is re-measuring places it has already been at these time steps). The re-sampling comes with little computation cost as seen by the blue line in Figure 4.3. To accomplish the same re-sampling with an exact GPR (shown in the black dashed line), takes a higher computational cost. The VFE-SGPR model has low computational cost once it has the inducing points optimized, but getting to that point is more expensive than DSGPR as seen by the red spikes in Figure 4.3. DSGPR is the best choice in this instance, it reduces the uncertainty at a similar rate as an exact GPR model, while not accruing the same amount of computational cost.

This advantage of DSGPR comes into play when is is necessary to make predictions with low uncertainty. The use case may require a certain threshold of certainty regarding prediction and the agents running DSGPR would be able to provide that confidence with a lower computational strain as compared to the other models. As seen in Figure 4.4, the uncertainty (global trace) decreases consistently over time. There are jumps of uncertainty decrease when re-clustering occurs. Also note that even when re-clustering does not occur, the DSGPR model decreases uncertainty even as areas already visited are re-measured. Note that all three approaches decrease the trace when new measurements occur, but the DSGPR approach has a much smaller cost of computation for the same result. Also observe that DSGPR performs better in terms of uncertainty reduction compared to VFE-SGPR; one potential reason for this result is the adjustments made to the VFE-SGPR algorithm such that it can be compared in a real-time data streaming setting. The optimization of inducing points is implemented with early stopping to allow it to be more competitive in terms of computational time compared to DSGPR, this early stopping does not allow optimal inducing point placement which may have affected its performance in terms of uncertainty reduction negatively.

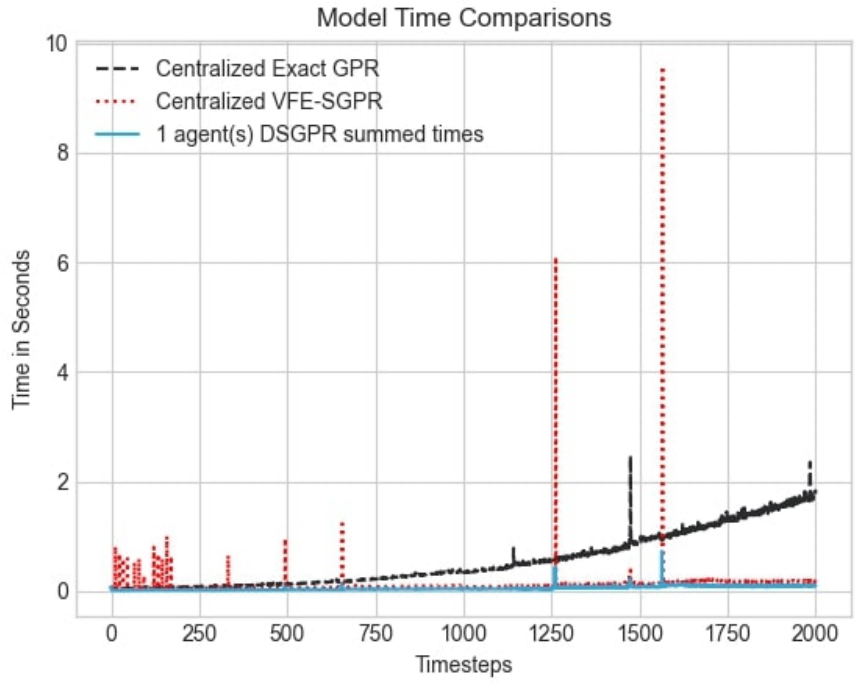

Figure 4.3: The total cost in terms of computation time of each model (exact GPR, VFE-SGPR, summed agents running DSGPR). Note the heavy cost of optimization for VFE-SGPR, and the cubic time of exact GPR.

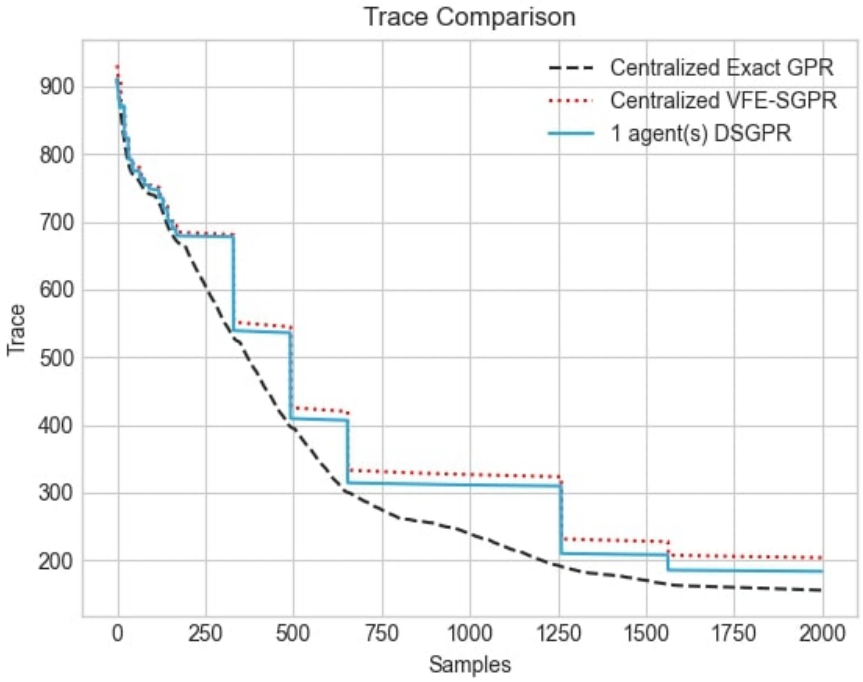

Figure 4.4: Notice how the trace of the global covariance matrix (uncertainty) decreases even when sampling of the same area occurs.

4.2.3 Inducing Point Selection Comparison

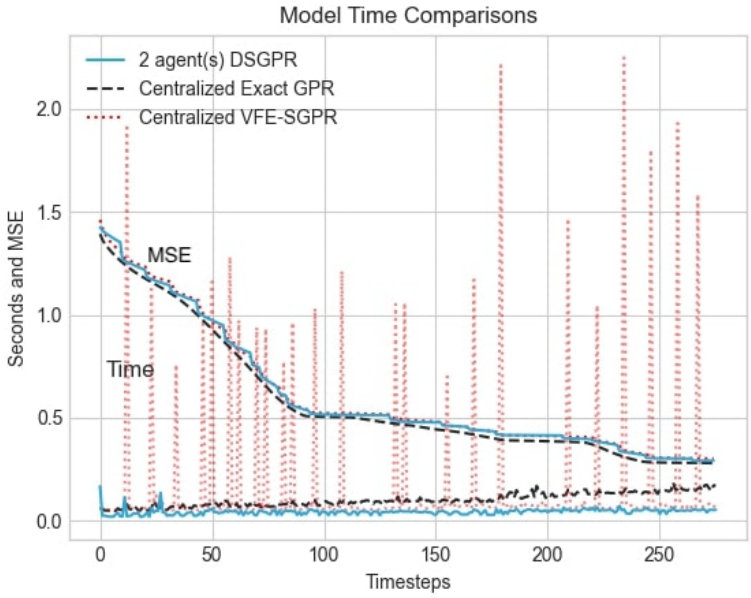

Figure 4.5: Time and MSE comparisons for DSGPR, centralized exact GPR, and centralized VFE-SGPR. DSGPR was run with two agents, their MSE is averaged and their times are summed.

Here, I explain how DSGPR has low computational cost in choosing inducing points compared to VFE-SGPR while maintaining competitive accuracy (MSE).

Section 2 describes how RC will adaptively determine the number of inducing points and select their positions. RC has a small computational cost compared to common SGPR inducing point optimizations. I show the optimization cost of radial clustering by comparing it with the optimization cost of the variational free energy SGPR (VFE-SGPR) approach developed in [35]. In this work, Titsias, et. al maximize the evidence lower bound (ELBO) to calculate optimal inducing point locations. Note that the ELBO is the lower bound for log-likelihood of the observed data given kernel hyper-parameters and inducing point locations.

When the evidence lower bound (ELBO) approach is used for inducing point location optimization, it becomes infeasible to re-optimize the inducing points. Figure 4.5 illustrates the results of a test where three GPR approaches are compared: a centralized exact GPR model (shown in dashed black), a centralized VFE-SGPR model (shown in dotted red), and two agents running RC with DSGPR (shown in blue). The figure shows that as the number of inducing points increase, the VFE-SGPR model requires large amounts of computation time to optimize the inducing point locations. To make the comparison fair, the VFE-SGPR model is re-optimized based on when each DSGPR agent re-clusters (this is an advantage that the standard VFE-SGPR model does not have). The original VFE-SGPR ELBO approach was implemented with early stopping gradient descent. Another advantage given to the VFE-SGPR approach is knowing how many inducing points are needed to represent the data.

Even with these advantages, the VFE-SGPR method is too slow to be a viable option for real-time inducing point optimization (in comparison to RC).

Figure 4.5 also shows the mean-squared-error (MSE) comparison for these models; note that all methods are competitive in terms of MSE (with centralized exact GPR being consistently the most accurate) with our decentralized approach being the best option in terms of time. Section 4.3.2 provides a more thorough testing/analysis of the DSGPR model computation times. However, it is evident from this example that RC provides fast inducing point optimization times with nearly identical resulting MSE values, as compared with exact GPR and VFE-SGPR models.

4.3 Monte Carlo Runs

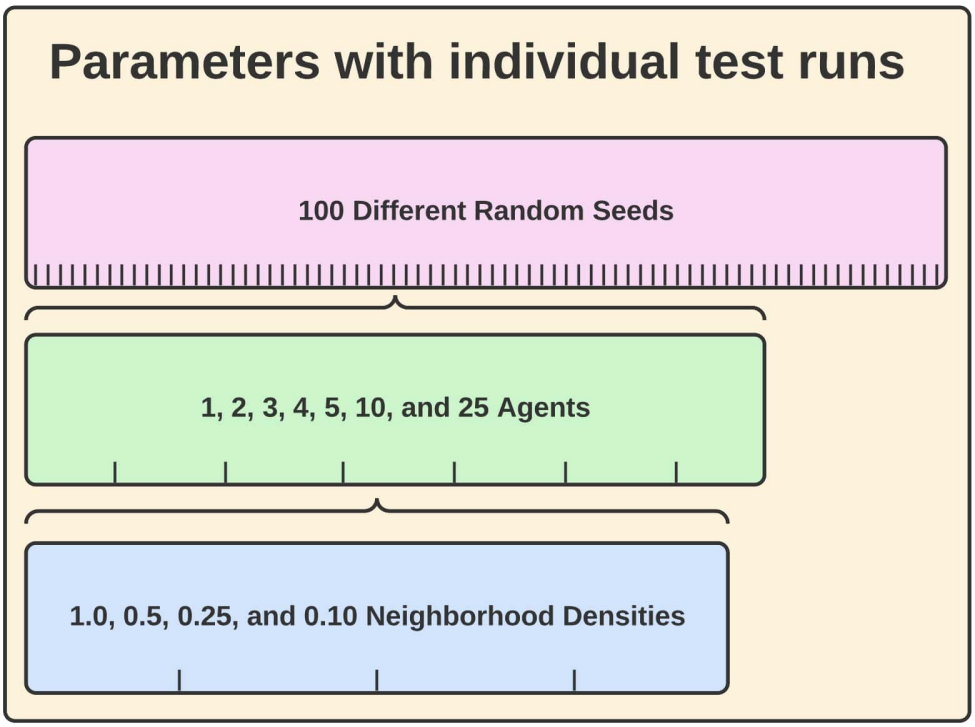

Figure 4.6: Breakdown of individual test runs given parameters of random seeds, number of agents, and neighborhood density.

I use Monte Carlo (MC) test runs to show that DSGPR is a viable real-time option for different numbers of agents. For each of 100 random seeds (pink box in Figure 4.6), runs are made with 1, 2, 3, 4, 5, 10, and 25 agents (green box in Figure 4.6). The random seeds create a new distinct wind-field, random agent starting positions, and random way-points for each agent. For each number of agents test, there are four neighborhood density tests that were run (blue box shown in Figure 4.6). The neighborhood densities represent the percentage of the boundary space that is covered by a single agent's message range. Since this can vary in a real-world scenario, I change the tests according to message passing densities of 1.0, 0.50, 0.25, and 0.10, where the density indicates the percentage of the boundary space

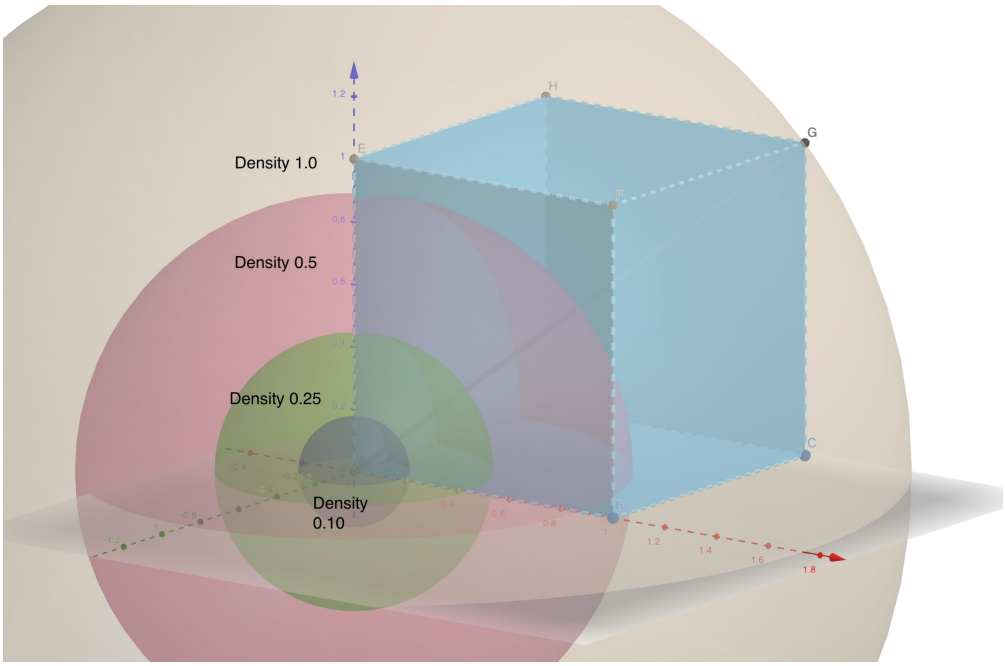

Figure 4.7: Message passing distances for each agent at the varying message passing densities

that is covered by the agent's spherical communication radius. This relationship is shown in Figure 4.7, where the spheres show the communication region for an agent positioned at $(0, 0, 0)$ and the blue box is the boundary space. Note that an agent with a density of 1.0 will have communication connectivity with all other agents no matter where they are in the boundary space (shown by the light pink sphere).

Parameter Name/Descr.	**Value**
Boundary Space Side Length	30
Test Points	1000
Kernel Length Scale	0.085
Kernel σ_f Term	1
High Kernel Significance Factor	0.8
Low Kernel Significance Factor	0.15
Re-Cluster Significance Factor	0.6
Wind Sensor Error	0.4
Kernel	Laplacian

Table 4.1: Name and values of the significant parameters used in simulation.

Each of these tests is run in a fixed cubic boundary space so as the number of agents increases, the reduction of MSE and uncertainty drops more quickly; a single agent can not traverse the cubic boundary space as quickly as a large number of agents.

The simulations all had some basic parameters that are noted in Table 4.1 for completeness and reproducibility. Notice how there are no units for these parameters, this is because the framework is scale invariant. It will be effective given any units and scale. The performance of DSGPR is more dependent on the sample rate than the units defining the size of the space. The wind sensor error is will also be in the units of whatever the units are of the magnitude of the wind measurements.

4.3.1 Mean-Squared Error (MSE) and Uncertainty

In this section, I show how the MSE error and the global trace (uncertainty) changes as a factor of the number of agents and communication densities.

The average MSE over all MC runs for different groups of agents is shown in Figure 4.8. As expected, the MSE consistently drops over time. Note that as the number of agents increases the MSE will decrease more rapidly. Given long enough simulation time, there will eventually be complete coverage of the bounding areas and the MSE for all numbers of agents would eventually decrease to similar MSE results.

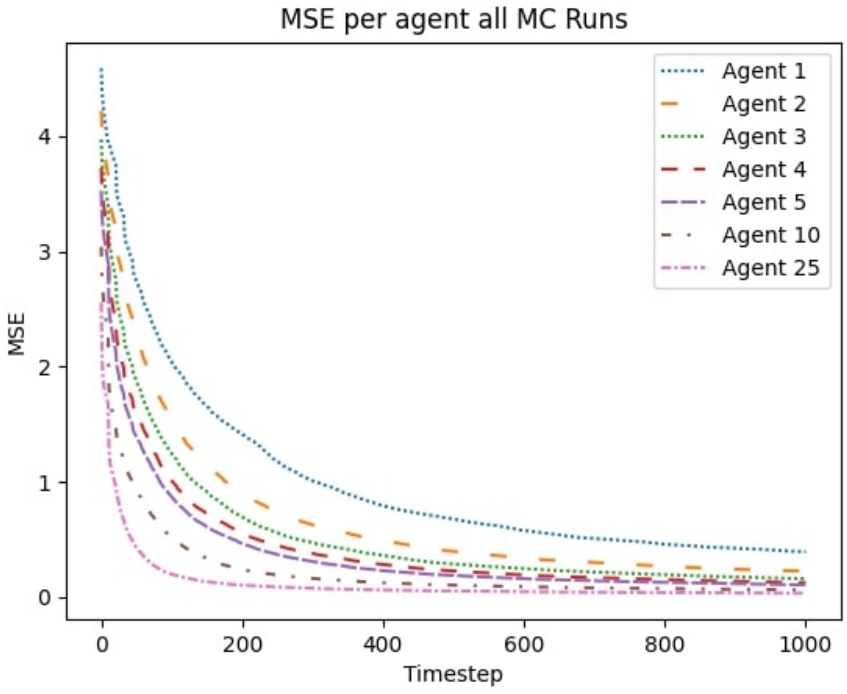

Figure 4.8: Average MSE across all MC runs and across each agent grouping. MSE at each timestep is plotted.

On top of MSE comparison for agent groupings, I use the message passing density illustrated in Figure 4.7 to explore how message passing affects the rate of MSE reduction.

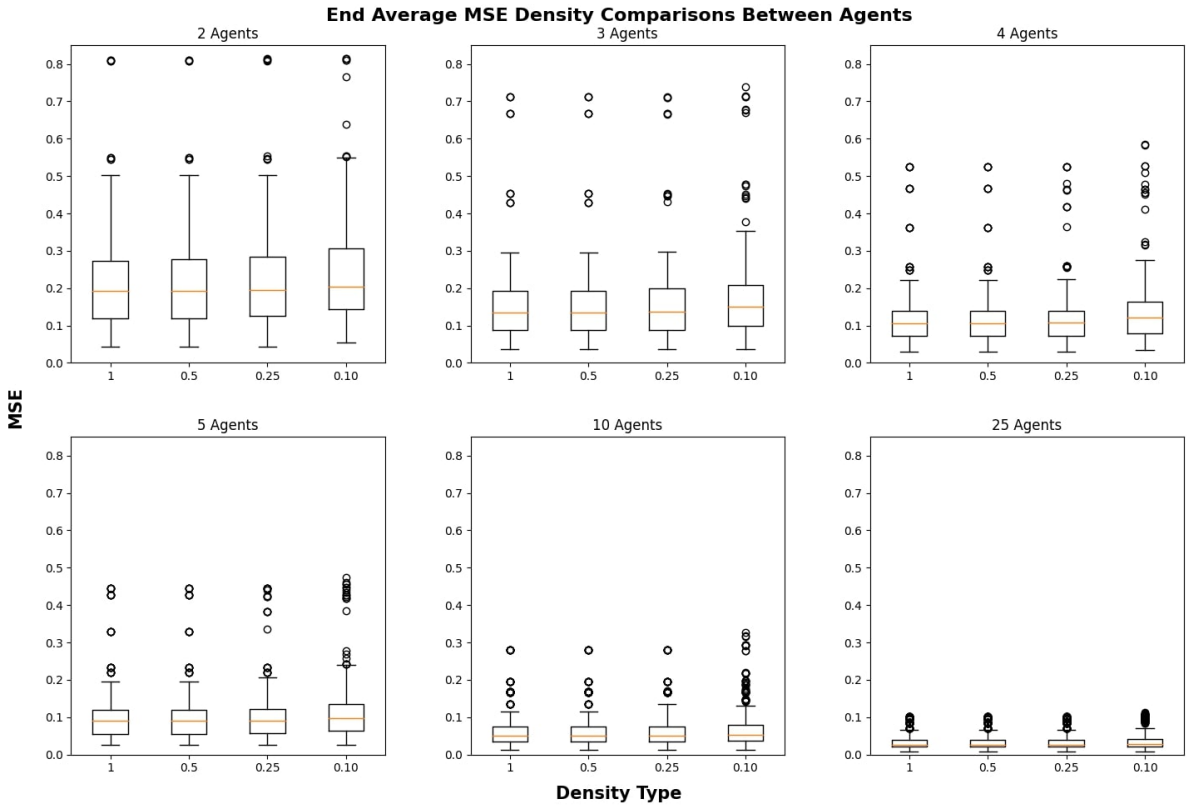

Figure 4.9: MSE box plot of each agent and their respective densities averaged across all MC runs. The plotted MSE value is the very last average agent MSE score of the run.

Figure 4.9 shows the MSE for each agent and density at the end of the 1000 time step simulations, averaged across all MC runs. The neighborhood densities are represented on the x-axis and decrease as we move right across the x-axis. The yellow line in the middle of each box is the median of the averaged MSE across all 100 runs. The bottom and top line of the box is the first and third quartile respectively. The distance between the first and third quartile is known as the inter-quartile range (IQR). Each whisker extends a distance of 1.5x the IQR. The unfilled dots extending passed the whiskers are outliers of the data.

This plot (Figure 4.9) only shows the MSE at the final time step. From it, we see that when agents are farther apart (or cannot pass messages according to neighborhood bounds i.e., message passing densities) each agent can have less global information (given a snapshot of the current model predictions). Note that accuracy degeneration is strongly evident in the 0.10 density case (i.e., when agents have an extremely limited message passing range); however, there is little MSE variation for the other densities. We can observe that when agents are not fully connected (in the 0.5, 0.25, and 0.10 density cases) the agent models are propagated throughout the network effectively enough that the drop in prediction accuracy is insignificant. This trait is extremely advantageous for agents surveying larger boundary spaces. Agents that can predict well even when they are not always within message passing range of their neighbors enables extended exploration of large boundary spaces. Also, as expected, the higher the number of agents (e.g., 5, 10, and 25 agents) in a fixed boundary

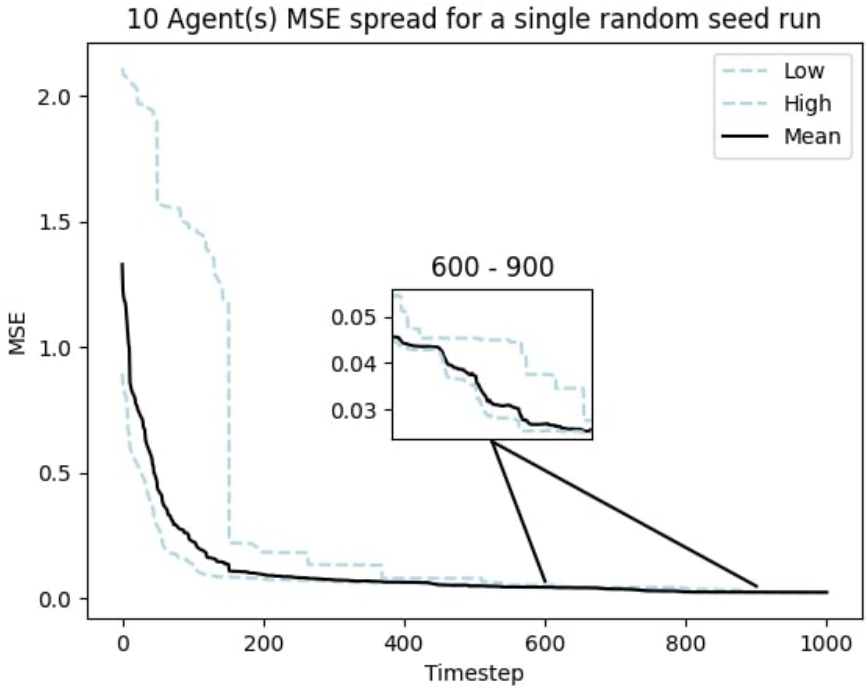

Figure 4.10: MSE low, high, and mean plotted for a 10 agent group from a single MC run. All message passing densities are considered in this plot. Low MSE variation is seen across all 10 agents.

space, the better the resulting prediction accuracy becomes, regardless of the message passing density.

After reviewing MSE variation given agent groupings and changing neighborhood densities, it is important to look at the spread of MSE within a single agent grouping. This is done to ensure that all agents perform adequately well and no discrepancy is being hidden by averaging across agent groups for individual runs. The spread is measured by plotting the low, high, and mean MSE for an agent group each time step in a randomly selected MC run. Figure 4.10 shows that while the 10 agents begin with high variation in their MSE scores, once message passing and exploration has occurred, the agent MSE variation is minuscule. The plot in Figure 4.10 is calculated across densities as well, showing that all agents perform within similar ranges for a given agent group even in varying message passing densities.

Along with analyzing prediction accuracy, determining whether or not there is confidence (low uncertainty) in those predictions is also important. Figure 4.11 depicts the trace of the global covariance matrix (uncertainty) averaged across each agent group for all MC runs (including the different density configurations). As expected, the trace decreases over time as new areas of the boundary space are explored as well as when areas are re-measured by the agents. The more agents exploring this fixed boundary space the quicker the uncertainty decreases. This is caused by having more agents able to explore more of the boundary space quicker than runs with less agents. This trace plot is similar to the average agent MSE plot in Figure 4.8 because both MSE and uncertainty will decrease as more diverse measurements

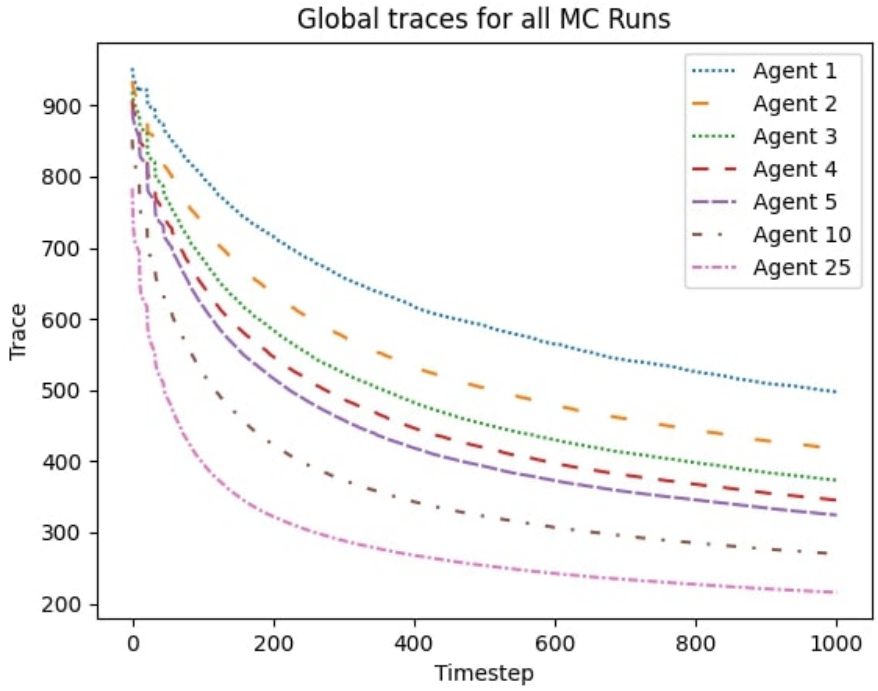

Figure 4.11: Global model covariance trace (uncertainty) results for all MC runs. The trace is taken from the global decentralized model provided by each agent and then averaged across all the agents participating in that run. Then it is averaged across all 100 random seed runs (including each density).

occur in the boundary space. Note that the trace (uncertainty) would decrease much more quickly and efficiently given an intelligent path planner that directs agents to high uncertainty regions rather than this current approach of having each agent choose random waypoints. For this work, it suffices that the overall global uncertainty is being reduced as agents explore the boundary space over time.

4.3.2 Computational Time

In this subsection, I examine the total computational cost for the model. To understand what components are the most costly, I show what the cost would be when not including the global decentralized computation and the computational strain for RC, message passing, and the global decentralized prediction portion of the framework. From this analysis, it is clear that the highest cost is the decentralized prediction computation of the global mean and covariance. Because the decentralized prediction is not necessarily needed each time step, this result provides more evidence for real-world viability of the DSGPR framework.

Exact GPR models have a computational complexity of $O(n^3)$, VFE-SGPR models have a computational complexity of $O(nm^2)$ where n is the number of data points and m is the number of inducing points. The DSGPR model presented in this work has a computational complexity of $O(\overline{m}^3)$ where $\overline{m}$ denotes the total number of inducing points for all agents.

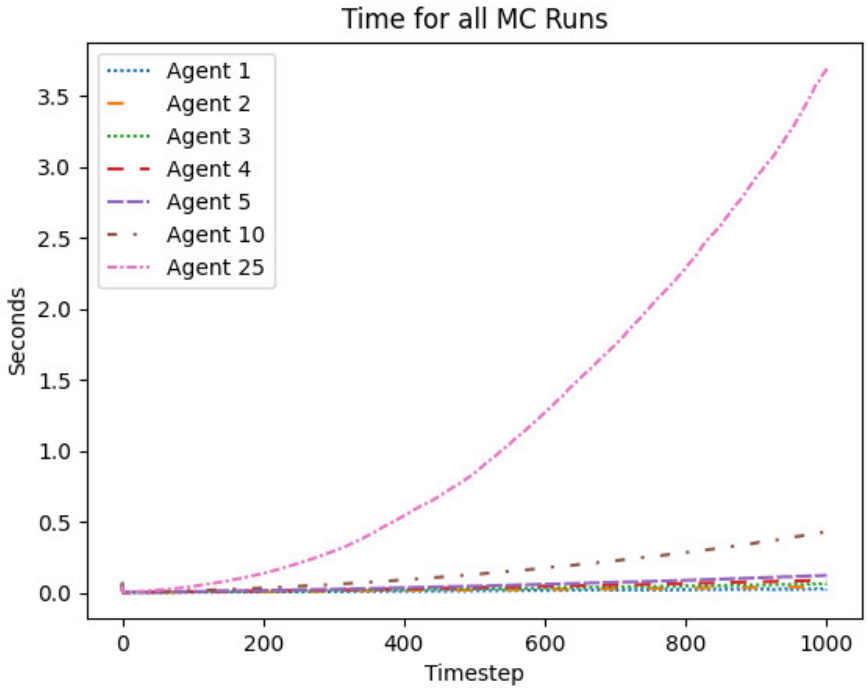

Figure 4.12: Total average DSGPR time results from all MC runs averaged across each agent.

Each individual agent has some m_i inducing points that are combined to create the global model. To determine the viability of this system for real-time performance, it is imperative to know how DSGPR performs in terms of computation time. For each MC run, the time to compute each major portion of the framework was tracked, allowing an analysis on the run time of the framework.

Figure 4.12 shows the average computation time for each agent averaged across all MC runs (including all density configurations) computing decentralized global model at each time step. Note that, as expected, the average times increase as the number of agents increases and the biggest jump in average times occurs between 10 to 25 agents. From this figure, it is evident that this model is not realistic for a large numbers of agents (25+) with sampling intervals of 1 second. However, it is viable in its current form (with a 1 second sampling rate) for real-time prediction with 1-10 agents.

Another consideration for why DSGPR can be utilized in real-time is because the global model is not always needed (i.e., it is not always required for decision making). If this is true, then it would only be computed as needed (e.g. when queried for by a user or needed by an agent). Figure 4.13 shows the computational time in seconds for running DSGPR without computing the decentralized global model. We can observe that the computational cost becomes more quadratic than cubic comparatively. The time includes the local model computation time, RC, event triggering, and message passing computations; the final global Gaussian process equations for the global mean and covariance (Equation (3.10)) are the only computations not included in this figure's plot. Note that all the global information is available and ready to be computed, but the global mean and covariance at the test points is

not computed. Depending on the problem to be solved, there are situations/use cases where the global mean and covariance would not need to be computed each time step. The rest of this section shows the computational cost of RC and message passing.

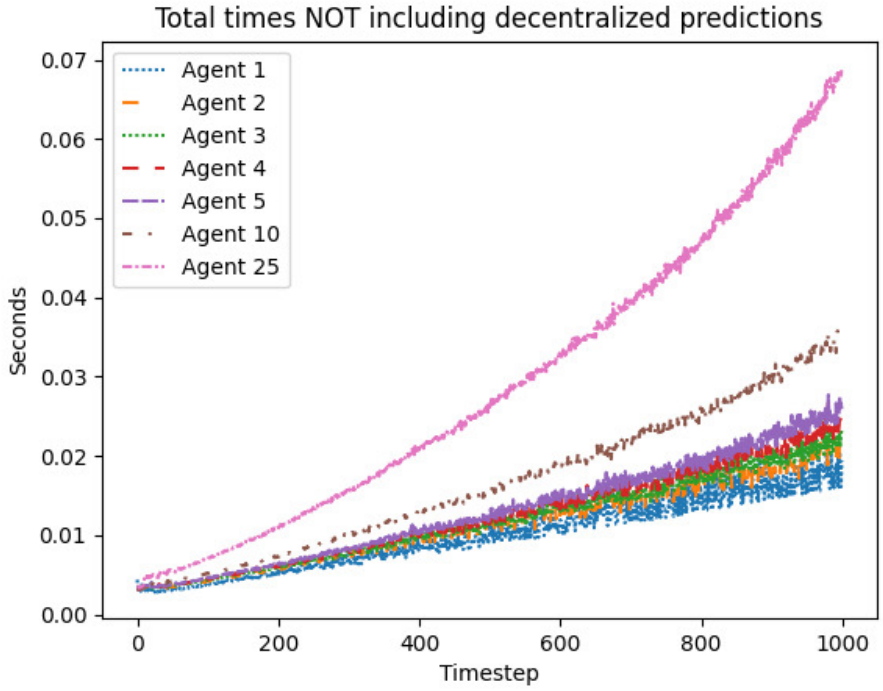

Figure 4.13: Total model times for all MC runs NOT including the global DSGPR model computation time.

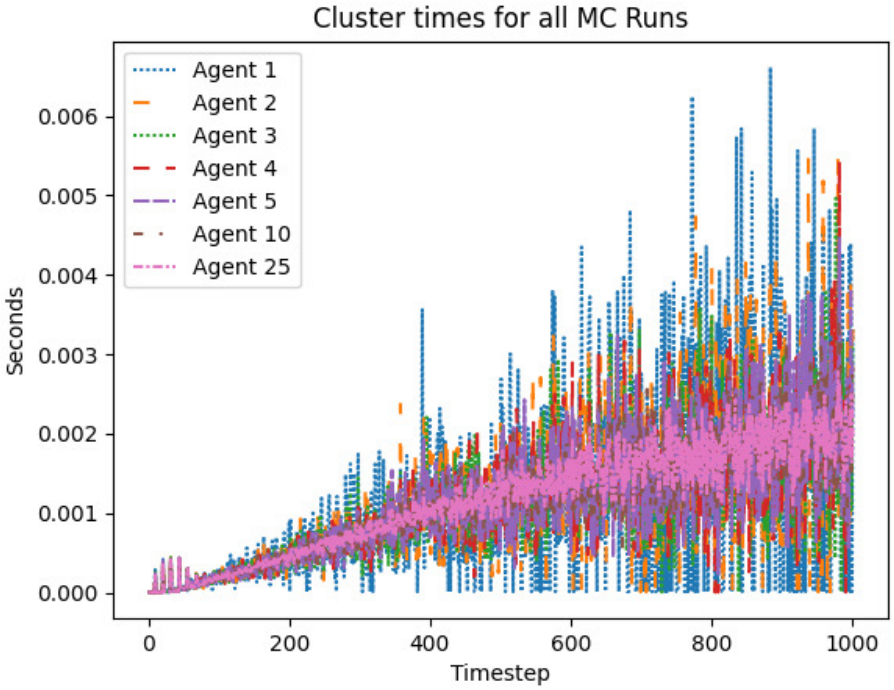

Figure 4.14: Average agent clustering times from all MC runs

The cost of RC for each agent is illustrated in Figure 4.14; we see that clustering only accounts for a small portion of the computation time. Average agent clustering scales according to $O(n^2)$, as mentioned in Section 2, because radial clustering is only dependent on each agent's local measurements. Since each agent is traveling at the same speed in these tests, they are gather close to the same number of measurements. Hence why all the RC computational times are extremely similar for all agent groupings and all message passing densities.

The message passing heuristics outlined in Sections 3.3.3 and 3.3.4 also have a computational cost that must be analyzed. Figure 4.15 shows the average agent's time to send messages averaged across all MC runs. This plot averages the time to send messages across all message passing densities.

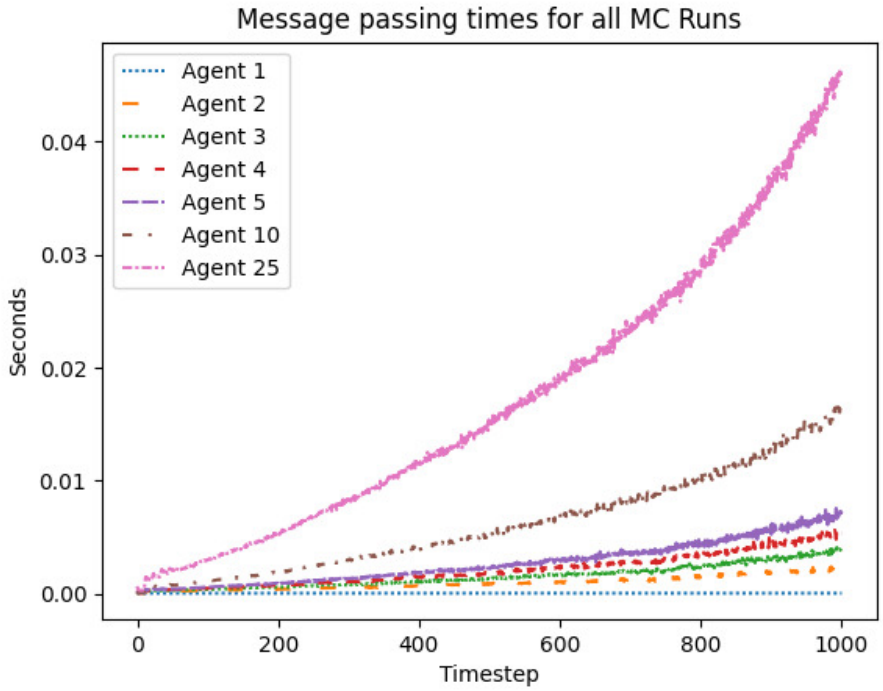

Figure 4.15: Average agent message passing times from all MC runs. This time includes all protocols mentioned in Section 3.3.2 and computation times of both heuristics from Sections 3.3.3 and 3.3.4.

It is clear that the bulk of the total time does not lie in clustering or message passing parts of the DSGPR framework. Most of computation resides inside the actual decentralized sparse Gaussian process regression calculations outlined in Section 3.2. Figure 4.16 shows how the DSGPR makes up almost all of the total computation time. This high computation cost is expected due to DSGPR calculations being $O(\overline{m}^3)$. As the number of agents increases, so does the total number of inducing points (denoted as $\overline{m}$). This high computation cost shows that if this calculation is not required each time step, it furthers the case for DSGPR being real-time viable. Future work could be done to run an optimization or radial clustering on top of all agent's inducing points to creating another layer of summarization thus reducing

the computation time of the final global DSGPR calculation. I leave that as a future avenue of research.

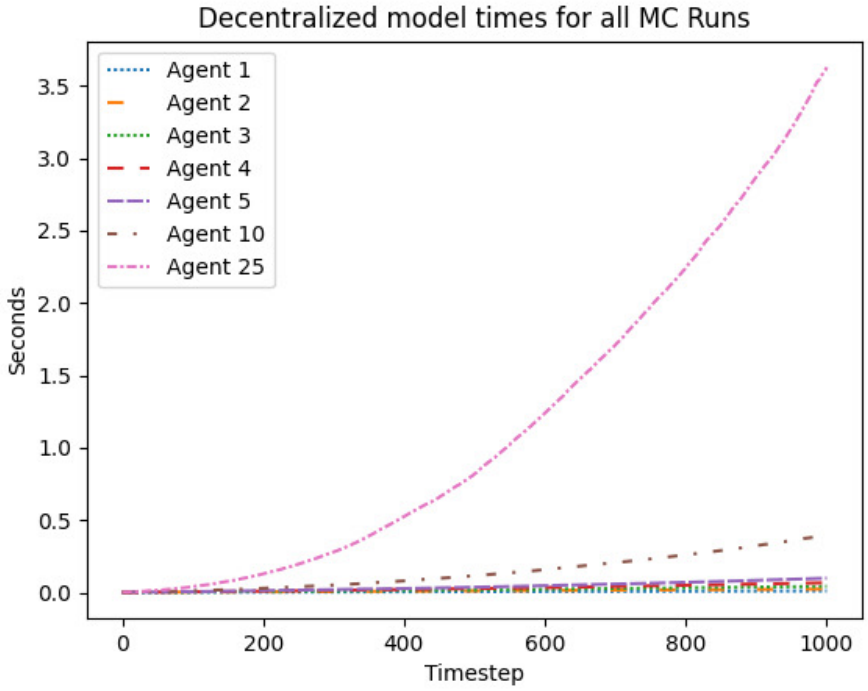

Figure 4.16: Average agent DSGPR computation times from all MC runs. This shows the cost of computing the global decentralized model. This global model will not be computed each time step in practice (unless the global model is required each time step for a different process).

4.3.3 Message Passing Memory Considerations

I present the performance of the message passing protocols and heuristics when run with varying number of agents and message passing densities. It is shown that the amount of memory sent between agents is small enough to be viable for real-world use given existing reasonable network data rates.

The message passing protocol presented in Section 3.3.2 can be analyzed through the MC tests. Because of necessary bandwidth restrictions, it is imperative that the data rate or total message size does not exceed reasonable message passing networks. Figure 4.17 shows the data rate in terms of Kilobytes per second (KBps) for all agent groups for the MC runs. The data rate is computed by summing up the total memory sent throughout all previous time steps and dividing it by the total number of time steps. Thus giving the rate at which data is transferred. As expected, the data rate increases as the number of agents increases.

Along with increasing number of agents, the data rate is also affected by the message passing density as described in Figure 4.7. We observe in Figure 4.19 that the density of 0.25 has the highest data rate out of all densities. This is due to the final protocol for

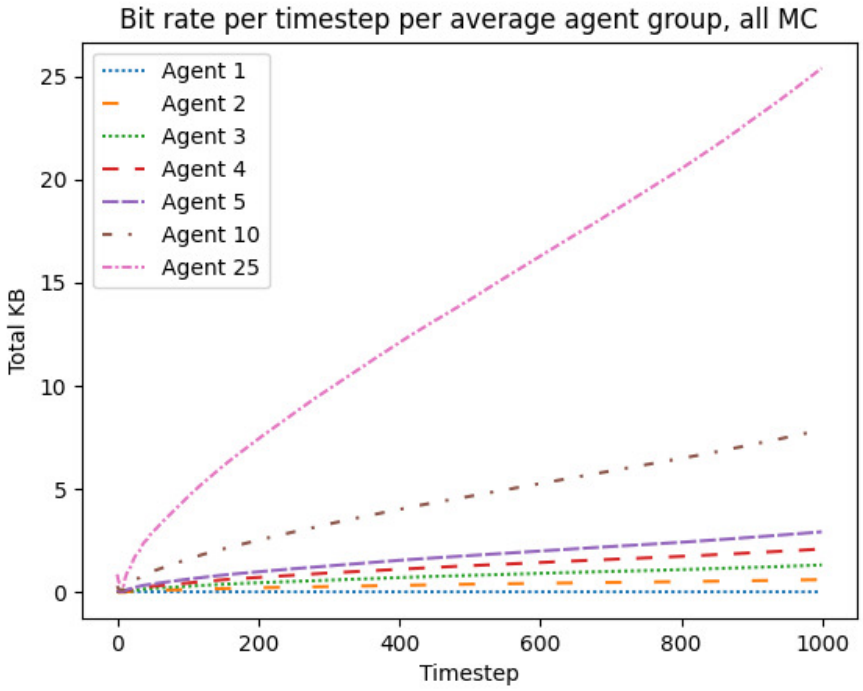

Figure 4.17: Average agent DSGPR message passing data rates (KBps) averaged across densities for all MC runs. As expected, the amount of data to send increases with the number of agents.

updating an agent's model such that other agents will query for the new model found in Section 3.3.2, the third bullet point. The low message passing density causes more new agent connections to occur as agents traverse the boundary space. Once a new connection occurs, the agent will send its most up to date model. Other agents already in communication with this agent will then query for the new model thus causing more message to be passed than in the 1.0 density case. This only occurs in the cases where message passing density is below 1.0. This feature increases the total number of messages sent, but enables agents that are spread out with low message passing density to have access to more up to date models thus improving global accuracy and uncertainty at a faster rate than sending an older less informative model. Another approach would be to not send an updated model with every new connection (expunging the third bullet in Section 3.3.2), this would decrease the total number of messages sent for message passing densities lower than 1.0 while decreasing global model performance.

While the data rates shown in Figures 4.17 and 4.19 are within manageable network ranges (assuming 100 Mbps), it is helpful to know the most messages a single agent can possibly send in a single time step given the different densities. When the message passing density is 1.0, the most messages that can possibly be passed by a single agent in a single time step is $A - 1$ where A is the number of agents in the entire system. This is due to the fact that no agent will ever need to send any model besides its own to another agent. When

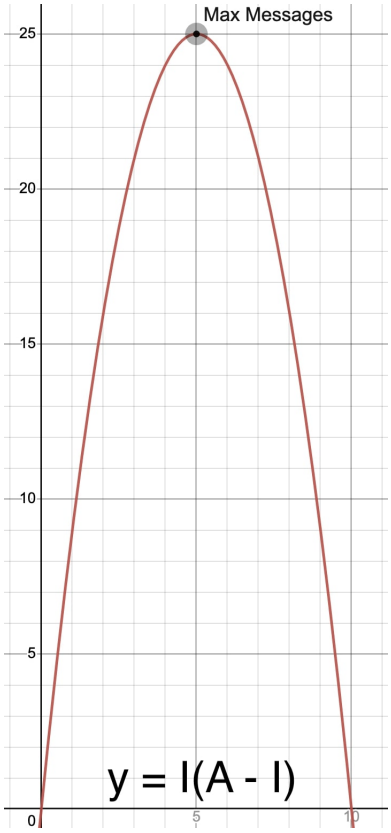

Figure 4.18: Plotted function showing the maximum number of messages that could be sent by an individual agent in a single time step. For illustration purposes, A is chosen to be 10 in this figure. I is the variable input on the x-axis, and the y-axis is the max number of messages that would be sent given a specified I.

an agent updates its own model, it will then send it to $A - 1$ agents for the 1.0 message passing density case.

When the agents are not fully connected, then there is the potential to send more than the fully connected max number of messages in a single time step. When not connected, the max number of messages that can be sent is $I(A - I)$ where I is the number of agents that could potentially query $A - I$ models from an agent that knows all $A - I$ models. This number will never be greater than $(A/2)^2$, this is seen in Figure 4.18.

This equation is shown true by a simple example. Assume $A = 50$, and $I = 25$ such that I is the agents which are completely isolated from the other 25 agents. Assume that agent W knows about 25 agents in the system (including itself). Assume that all I isolated agents come into contact with agent W within the same time step (i.e., all are simultaneously within message passing range). Now agent W must sent the 25 local models it knows about to all I isolated agents. Since $A = 50$ and $I = 25$, then the max number of messages that can possibly be sent by agent W is $25(50 - 25)$ which resolves to 625 messages for that time step. We can see if $I = 24$ then the resulting number of messages is 624, and if $I = 26$ then the result is also 624. Thus showing that the max number of messages that can be sent is

indeed $(A/2)^2$. This example shows how the lower density groups will have greater data rate costs than the fully connected 1.0 density.

We can observe in Figure 4.20 that the highest amount of memory sent by an agent during a single time step was only 50 KB. This shows the real-world viability for this system given the reasonable existing data rates of 100 Mbps. Even in the worst case scenario for 1000 time steps, most existing networks can handle this type of traffic. We can also note/observe that both Figures 4.19 and 4.20 are scaling at rate similar to a linear rate. Thus showing the scalability of this message passing system given varying message passing densities (i.e., varying agent connectivity and sparsity).

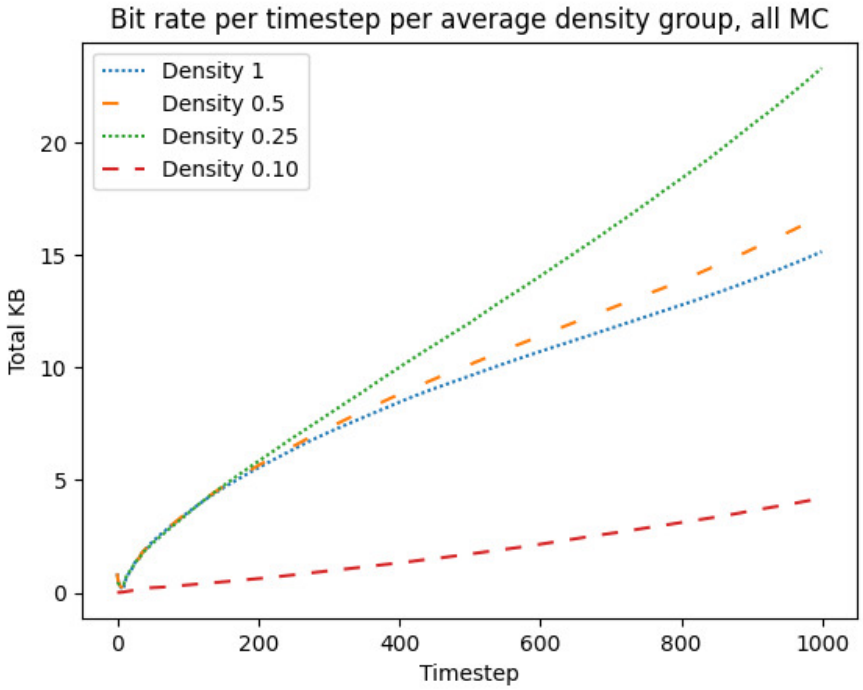

Figure 4.19: The average data rate at each time step for each density group averaged across the number of agents.

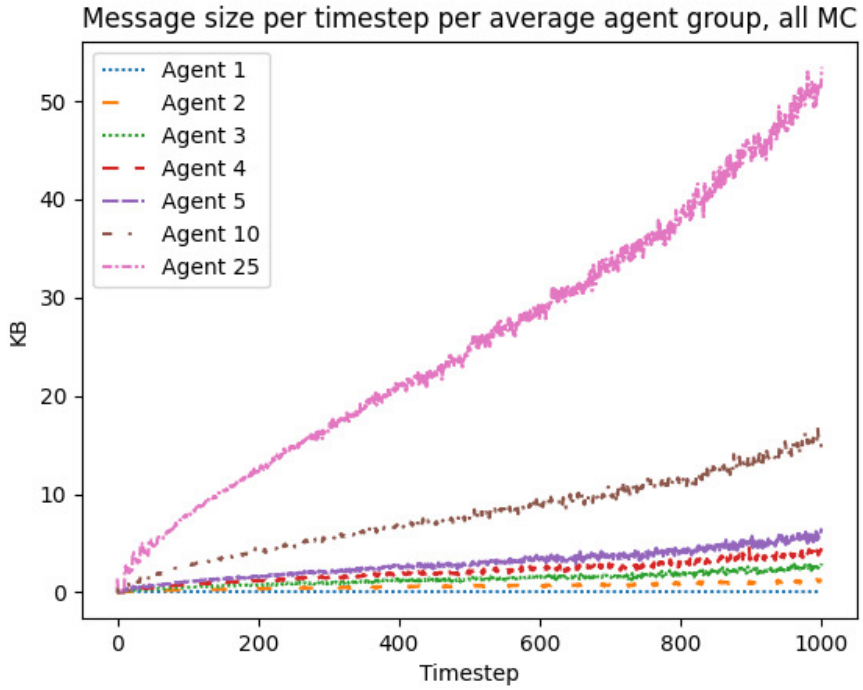

Figure 4.20: Message size per time step averaged across message passing densities. We can observe memory size scales at a linear rate for the first 1000 time steps.

Chapter 5

Conclusion

This paper presents a DSGPR framework with efficient message passing along with event-triggered adaptive inducing points. The purpose of this framework is to allow multiple agents to perform real-time perception of surrounding scalar or vector fields and make quick, well-informed decisions based on that perception.

This work focuses on a specific use-case of multiple aerial agents predicting a three dimensional wind-field with uncertainty bounds on that prediction. Prediction and uncertainty are performed through the presented formulation of DSGPR in Section 3.2. The contributions made in this thesis to accomplish this task more effectively are as follows:

1. A clustering algorithm providing an adaptive number of inducing points and their locations for decentralized SGPR.
2. A guaranteed two and three dimensional upper bound for those adaptive inducing points.
3. An effective event-triggering algorithm for updating inducing point values and locations.
4. A decentralized SGPR framework equipped with message passing heuristics that control message size and message passing frequency.

The contributions can be summed up into a DSGPR framework. The framework consists of each agent creating a local representation of the environment using a sparse Gaussian process regression (SGPR) model. Each agent then passes these local models according to the message passing protocols outlined in Section 3.3.2. These protocols implement maximum consensus of model timestamps for all agents. Even when agents have varying densities of connection, maximum consensus occurs as long as no agent or small groups of agents are completely isolated. Message passing frequency and message size are efficiently controlled using the presented heuristics in Sections 3.3.3 and 3.3.4.

Each agent has an adaptive number of local inducing points according to their own path and trajectory. The method for how agents can adapt their inducing point location and number is outlined in Section 2. An upper bound for the possible number of inducing points is proven for two and three dimensional square/cubic boundary spaces in Sections 2.3 and 2.2. Re-clustering occurs through an event-triggering process described in Section 2.3.

These contributions show that DSGPR is an effective real-time option for MASs modeling a vector field. The effectiveness of DSGPR is shown through the tests and analysis performed in Section 4. The test results show that the DSGPR framework is real-time effective for MASs of small to medium number of agents (1-10). One main advantage of this DSGPR framework over other sparse and exact methods is the low cost to re-measure an area to reduce the total amount of uncertainty (Section 4.2.2). It was also shown that DSGPR is

competitive in terms of reducing MSE as compared to centralized exact GPR and centralized sparse GPR models. The results also show that the RC algorithm for adaptive inducing points has low cost in terms of computational time complexity as seen in Figure 4.14. The message passing cost of this system is also shown to be minimal; therefore, showing that the DSGPR framework is a viable option for integration into real-time systems.

5.1 Future Work

The work presented in this paper has multiple areas where future effort could be placed, including online hyper-parameter tuning and GP regression, summarizing inducing points, and enabling heterogeneous vehicles into the DSGPR framework.

The first and most pressing area would the area of online hyper-parameter tuning in this decentralized framework. Tuning the hyper-parameters allows agents to get a better real-time view of the current vector field being measured. Preset hyper-parameters will never be as good as ones calculated from the current data. On top of doing global online hyper-parameter tuning, another promising avenue is individual agents having their own hyper-parameters which can be combined into a global model. One approach to accomplishing this idea is explored in [17] and gains similar hyper-parameter results to a mixture of experts type of approach. However, neither of them have an adaptive number of inducing points for their local models. Being able to tune hyper-parameters locally and allowing for adaptive inducing points is definitely a future area to pursue. The reason tuning could be different given adaptive inducing points is the question of can there be an objective function such that the adaptive inducing point locations can be tuned as well as the kernel hyper-parameters fast enough for real-time? Progress in the area of online hyper-parameter tuning would increase the viability of this system in real-world scenarios.

Another area to help increase real-world viability is the area of adjusting the framework into a completely online system. Currently, the work done in [3] and [39] are the few approaches where data is streamed online. This is accomplished by updating the model and then forgetting old measurements, while keeping the inducing points that summarize them. This online methodology is an enticing avenue due to the low amount of storage required since old data is quickly forgotten. Adjusting the DSGPR model in this thesis into a fully online model would gain these memory storage benefits while maintaining the adaptive inducing point advantage expressed in this thesis that [3] and [39] do not have.

Another future avenue that would enable lower memory storage costs is another summarization of all agent's inducing points ($\overline{m}$) as presented work in this thesis. Currently, this work does not summarize the combined agent's inducing points $\overline{m}$. This limits DSGPR's scalability to hundreds and thousands of agents. Each agent can have a large number of inducing points and if they each have to maintain a list of all agent's inducing points the total can grow large as the number of agents increase. One option would be to apply the RC algorithm on the received inducing points (similar to how the current framework applies the RC algorithm to the received measurements) to create another summary set of all agents' inducing points. This would enable each agent to compute the global regression model without incurring the large cost of a GP using a large set of inducing points. However, summarizing inducing points would necessitate a loss in the global model's fidelity. It would be critical to find a bound on how this would influence the fidelity of the global model.

Another avenue related to summarizing the combined total of all agent's inducing points is in the pursuit of coordinating heterogeneous vehicles. This thesis is concerned with homogeneous vehicles that have the same sensors and similar mediums through which they travel. However, the decentralized nature of the algorithm could be extended to addressing the problem of how vector field modeling could occur with heterogeneous vehicles. One potential option would be projecting each vehicle's measurements or predictions onto a high-dimensional manifold that holds a global semantic meaning. Each heterogeneous vehicle could gain information from and add information to each other using this higher dimensional manifold. This avenue of research could equip heterogeneous agents with the ability to perform more diverse tasks as compared to a homogeneous group of agents.

Along with these future avenues, the need for cooperative multi-agent systems grows each year. The work shown in this thesis makes several meaningful contributions and sets the foundation for future work to enable multi-agent integration into commercial/professional use.

References

[1] R. Allamraju and G. Chowdhary. Communication efficient decentralized gaussian process fusion for multi-uas path planning. In *2017 American Control Conference (ACC)*, pages 4442–4447, 2017. doi: 10.23919/ACC.2017.7963639.

[2] Sneha N Aware and Varsha H Patil. Survey on various-width clustering for efficient k-nearest neighbor search. *IJARIIE*, 3(2), 2017.

[3] Thang D Bui, Cuong Nguyen, and Richard E Turner. Streaming sparse gaussian process approximations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f31b20466ae89669f9741e047487eb37-Paper.pdf`.

[4] J. Chen, K. H. Low, Y. Yao, and P. Jaillet. Gaussian process decentralized data fusion and active sensing for spatiotemporal traffic modeling and prediction in mobility-on-demand systems. *IEEE Transactions on Automation Science and Engineering*, 12(3): 901–921, 2015. doi: 10.1109/TASE.2015.2422852.

[5] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995. doi: 10.1109/34.400568.

[6] Henry Cohn, Abhinav Kumar, Stephen Miller, Danylo Radchenko, and Maryna Viazovska. The sphere packing problem in dimension 24. *Annals of Mathematics*, 185(3):1017 – 1033, 2017. doi: 10.4007/annals.2017.185.3.8. URL `https://doi.org/10.4007/annals.2017.185.3.8`.

[7] John H. Conway and N. J. A. Sloane. Sphere packings, lattices and groups. In *Grundlehren der mathematischen Wissenschaften*, 1988.

[8] Zachary Cook, Monia Kazemeini, Alexander Barzilov, and Woosoon Yim. Low-altitude contour mapping of radiation fields using UAS swarm. *Intelligent Service Robotics*, 12(3):219–230, 2019. ISSN 18612784. doi: 10.1007/s11370-019-00277-8. URL `https://doi.org/10.1007/s11370-019-00277-8`.

[9] Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian processes. In *32nd International Conference on Machine Learning, ICML 2015*, volume 2, pages 1481–1490, 2015. ISBN 9781510810587. doi: 10.25561/21163.

[10] Arman Fazeli, Alexander Vardy, and Eitan Yaakobi. Generalized sphere packing bound. *IEEE Transactions on Information Theory*, 61(5):2313–2334, 2015. doi: 10.1109/TIT.2015.2413418.

[11] Théo Galy-Fajou and Manfred Opper. Adaptive inducing points selection for gaussian processes, 2021.

[12] TC Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 11 2005.

[13] Thomas Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture, 2015.

[14] Thomas C. Hales. The sphere packing problem. *Journal of Computational and Applied Mathematics*, 44(1):41–76, 1992. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(92)90052-Y. URL https://www.sciencedirect.com/science/article/pii/0377042792900 52Y.

[15] K. Han, Y.T. Feng, and D.R.J. Owen. Sphere packing with a geometric based compression algorithm. *Powder Technology*, 155(1):33–41, 2005. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2005.04.055. URL https://www.sciencedirect.com/science/article/pii/S0032591005001956.

[16] Falk Heße, Vladyslav Prykhodko, Steffen Schlüter, and Sabine Attinger. Generating random fields with a truncated power-law variogram: A comparison of several numerical methods. *Environmental Modelling & Software*, 55:32–48, 2014. ISSN 1364-8152. doi: https://doi.org/10.1016/j.envsoft.2014.01.013. URL https://www.sciencedirect.com/science/article/pii/S1364815214000231.

[17] Trong Nghia Hoang, Jonathan How, Quang Minh Hoang, and Kian Hsiang Low. Collective online learning via decentralized Gaussian processes in massive multi-agent systems. *arXiv*, 2018. ISSN 23318422.

[18] Dohyun Jang, Jaehyun Yoo, Clark Youngdong Son, Dabin Kim, and H. Jin Kim. Multi-robot active sensing and environmental model learning with distributed gaussian process. *IEEE Robotics and Automation Letters*, 5(4):5905–5912, 2020. doi: 10.1109/LRA.2020.3010456.

[19] Ravi Kashyap. The perfect marriage and much more: Combining dimension reduction, distance measures and covariance. *Physica A: Statistical Mechanics and its Applications*, 536:120938, 07 2019. doi: 10.1016/j.physa.2019.04.174.

[20] Michael J. Kuhlman, Dylan Jones, Donald A. Sofge, Geoffrey A. Hollinger, and Satyandra K. Gupta. Collaborating underwater vehicles conducting large-scale geospatial tasks. *IEEE Journal of Oceanic Engineering*, 46(3):785–807, 2021. doi: 10.1109/JOE.2020.3041123.

[21] Chang Liu, Zhihao Liao, and Silvia Ferrari. Rumor-robust Decentralized Gaussian Process Learning, Fusion, and Planning for Modeling Multiple Moving Targets. *Proceedings of the IEEE Conference on Decision and Control*, 2020-December(Cdc):3066–3071, 2020. ISSN 07431546. doi: 10.1109/CDC42340.2020.9304365.

[22] Haitao Liu, Jianfei Cai, Yi Wang, and Yew Soon Ong. Generalized robust Bayesian committee machine for large-scale Gaussian process regression. In *35th International Conference on Machine Learning, ICML 2018*, volume 7, pages 4898–4910, 2018. ISBN 9781510867963. URL `https://github.com/LiuHaiTao01/GRBCM`.

[23] S. Müller, L. Schüler, A. Zech, and F. Heße. `GSTools` v1.3: a toolbox for geostatistical modelling in python. *Geoscientific Model Development*, 15(7):3161–3182, 2022. doi: 10.5194/gmd-15-3161-2022. URL `https://gmd.copernicus.org/articles/15/3161/2022/`.

[24] Barış Nakiboğlu. The sphere packing bound via augustin's method. *IEEE Transactions on Information Theory*, 65(2):816–840, 2019. doi: 10.1109/TIT.2018.2882547.

[25] S. O'Dea. Mobile provider latency in the us 2019, Dec 2021. URL `https://www.statista.com/statistics/818205/4g-and-3g-network-latency-in-the-united-states-2017-by-provider/`.

[26] Ruofei Ouyang and Bryan Kian Hsiang Low. Gaussian process decentralized data fusion meets transfer learning in large-scale distributed cooperative perception. *Autonomous Robots*, 44(3-4):359–376, 2020. ISSN 15737527. doi: 10.1007/s10514-018-09826-z. URL `https://doi.org/10.1007/s10514-018-09826-z`.

[27] Liqian Peng, Doug Lipinski, and Kamran Mohseni. Dynamic Data Driven Application System for Plume Estimation Using UAVs. *J Intell Robot Syst*, 74:421–436, 2014. doi: 10.1007/s10846-013-9964-x.

[28] Yuan Qi, Ahmed H Abdel-Gawad, and Thomas P Minka. Sparse-posterior Gaussian Processes for general likelihoods. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, 2010.

[29] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[30] Matthias Seeger, Christopher K.I. Williams, and Neil D. Lawrence. Fast Forward Selection to Speed Up Sparse Gaussian Process Regression. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, pages 254–261, 2003. URL `http://proceedings.mlr.press/r4/seeger03a/seeger03a.pdf`.

[31] Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume R4 of *Proceedings of Machine Learning Research*, pages 254–261. PMLR, 03–06 Jan 2003. URL `https://proceedings.mlr.press/r4/seeger03a.html`. Reissued by PMLR on 01 April 2021.

[32] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005. URL `https://proceedings.neurips.cc/paper/2005/file/4491777b1aa8b5b32c2e8666dbe1a495-Paper.pdf`.

[33] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264, 2006.

[34] Edward Snelson and Zoubin Ghahramani. Local and global sparse gaussian process approximations. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 524–531, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[35] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Proceedings of the Twelth International Conference on Artificial Intelligence and*

Statistics, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574. PMLR, Apr 2009.

[36] Maryna S. Viazovska. The sphere packing problem in dimension 8. *Annals of Mathematics*, 185(3):991–1015, 2017. ISSN 0003486X. URL http://www.jstor.org/stable/26395747.

[37] S. Wang and W. Ren. On the consistency and confidence of distributed dynamic state estimation in wireless sensor networks. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 3069–3074, 2015. doi: 10.1109/CDC.2015.7402680.

[38] Eric W. Weisstein. "Cubic Close Packing." From Mathworld–A Wolfram Web Resource., 2022. URL https://mathworld.wolfram.com/CubicClosePacking.html.

[39] Brian Wilcox and Michael C. Yip. Solar-gp: Sparse online locally adaptive regression using gaussian processes for bayesian robot model learning and control. *IEEE Robotics and Automation Letters*, 5(2):2832–2839, 2020. doi: 10.1109/LRA.2020.2974432.

[40] Zhenyuan Yuan and Minghui Zhu. Lightweight distributed gaussian process regression for online machine learning, 2021. URL https://arxiv.org/abs/2105.04738.

Appendix A

Two Dimensional Approximation of Maximum Clusters

I outline an equation for finding a less conservative approximation of the maximum number of clusters that can fit in a square boundary space. This is different from the prior approaches (Section 2.2.2 and 2.2.3)to an upper bound which provided a conservative upper bound on this number. This equation is not proven to be the exact upper bound, but is shown below to be extremely accurate.

Let $\lceil\ \rceil$ and $\lfloor\ \rfloor$ refer to the ceiling and flooring functions respectively. Given a cluster radius r and square boundary space with side length of b, the maximum number of inducing points that can be placed by the radial clustering algorithm is given by

$$U_B = s^2 + (s-1)\nu + a_c, \tag{A.1}$$

where

$$s = \left\lceil \frac{b}{r} \right\rceil \tag{A.2}$$

is a bound on the number of points needed to cover one side of the square,

$$a_c = \left\lceil \frac{1}{1+e^{-|\lfloor \frac{d_c}{r} \rfloor|}} - 0.5 \right\rceil \left\lceil \frac{1}{1+e^{-|d_c - r|}} - 0.5 \right\rceil, \tag{A.3}$$

is an additive term that addresses the corner edge case (explained below), with $d_c = \sqrt{(r(s-1)-b)^2 + (r(s-1)-b)^2}$, which is the distance from the most upper-right circle to the upper-right boundary corner (assuming circle placement started in the lower-left boundary corner), and

$$\nu = 2\left(1 - \left\lfloor \frac{d_n}{b} \right\rfloor\right) \tag{A.4}$$

is a value that covers the intersection edge case (explained below) with $d_n = r(s-1) + c$, $c = r\sqrt{3}/2$.

To show that this equation provides a solution, I show that the upper bound on inducing points given by Equation (A.1) provides clusters that completely cover the b^2 boundary space. Since clusters are defined by a single radius, this is equivalent to finding the

number of overlapping circles that can be placed that completely cover the bounding area, assuming each circle center lies one radius away from the previous circle.

Without loss of generality, assume the boundary space starts at $(0, 0)$ and is the center point for the first circle. Equation (A.2) represents how many circles will fit on one boundary edge until the last placed circle's radius is greater than b. I square s in (A.1) to fill the square boundary space. Once this has been calculated, there are two edge cases that need to be accounted for to get the true upper bound. Those edge cases are covered in the last two terms of Equation (A.1).

Using the ν term, I address the edge case where the radii of the last circle in the last column/row is greater than b but the intersection point between adjacent circles is not greater than b; therefore, the whole boundary space b would not be covered. The $(s-1)\nu$ term accounts for this case and adds another row and column of circles to the total upper bound at those intersection points, i.e. the intersection points become the circle centers for the last row/column of circles. I calculate the height/width distance d_n between the intersection point of the last row/column of circles and the boundary space by summing the distance to the last circle centroid $r(s-1)$ with the height/width distance from this centroid to the intersection point c. The distance c is simply the height of the equilateral triangle formed by the intersection point and the two circle centers. If $d_n < b$, then I must add $2(s-1)$ circles, where the 2 accounts for the addition of circles to both the last row and column. If $d_n > b$, then no extra circles need to be added due to a complete coverage being obtained. Once d_n is computed, flooring d_n/b functions as an if statement. When $d_n \geq b$, the flooring results in a value of 1 which yields 0 after the subtraction, ultimately turning off the term such that the intersection circles are not included in the upper bound. However, when $d_n < b$, ν gains a value of 2 which will add the $2(s-1)$ circles to u_b.

Lastly, the a_c term covers an edge case where the last placed circle's radius does not cover the final corner of the square boundary space. The distance from the last column and last row's circle center to the corner of the boundary space is d_c. If the radii of the last circle excludes the corner, then add one circle to the upper bound u_b. The a_c term functions as an if statement to help determine whether or not the corner circle should be added in a given case. If $d_c < r$, then $\lfloor d_c/r \rfloor$ will output 0 which will take the left-hand term of a_c to 0 ensuring that no corner circle is added. If $d_c = r$, the right-hand term of a_c will also compute to 0 under the same logic aforementioned. Lastly, if $d_c > r$, then I add one extra circle to u_b. Computing both left and right terms of a_c outputs a one in this case ensuring that a circle is added to u_b. All of these edge cases are shown in Figure A.1.

The upper bound ultimately determines the maximum number of clusters for given a radius r and square boundary space b^2. This directly equates to the max number of inducing

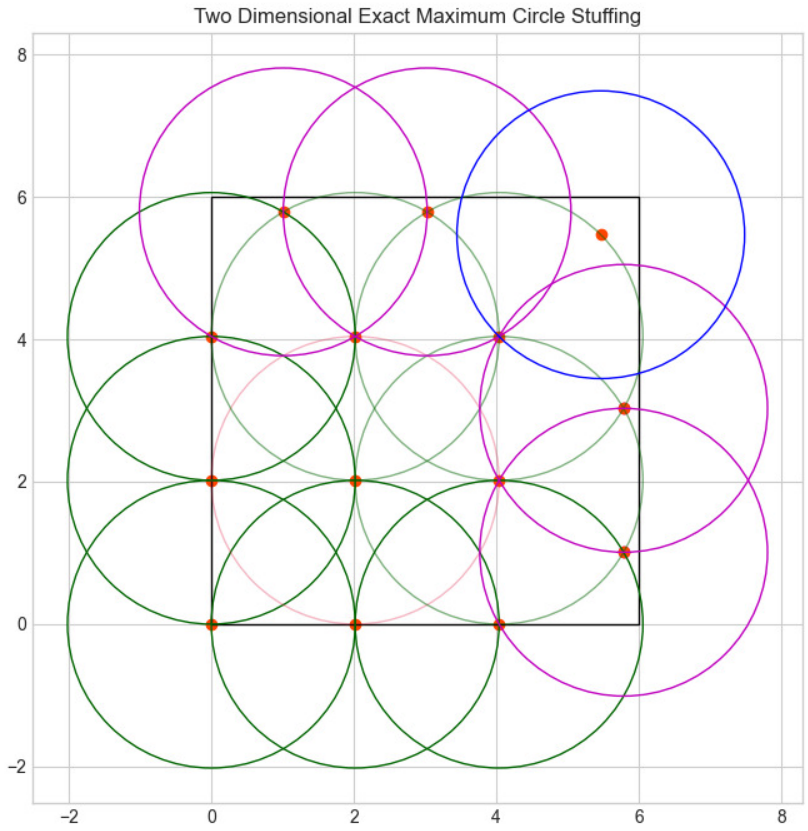

Figure A.1: Example of Radial Clustering Upper Bound in Square Boundary Space. Each red point represents a chosen inducing point, while each circle represents that inducing point's radius of influence.

points possible. Knowing this value provides limits on the computational load and the size of messages communicated between agents. Note that a similar approach could be used to provide a less conservative approximation in the 3D case as well. This is not explored in this thesis.