Strategic usage of Real-time data for Dynamic Data Driven Simulation
A Sensitivity Analysis

Master thesis submitted to Delft University of Technology
in partial fulfillment of the requirements for the degree of

Master of Science
in Engineering and Policy Analysis
Faculty of Technology, Policy and Management

by

Yubin Cho

to be defended publicly on Friday August 23, 2019 at 15:00 AM.

Student number: 4738020
Project duration: March 1, 2019 – August 22, 2019
Thesis committee: Prof. dr. ir. A. Verbraeck, TU Delft, Chair
Dr. Y. Huang, TU Delft, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.
This thesis presents the result of the past six months of research carried out to meet the graduation requirements of the master program Engineering and Policy Analysis in TU Delft. The main topic is the data assimilation of the discrete-event based system. I hope that any reader with basic background knowledge of Modelling & Simulation, Bayesian inference and the state estimation may find it interesting. The topic inspired by its promising potential given that the sensor technology and the big-data storage technology are rapidly developing. During the research, I was able to gain a better understanding of what I have covered for two years of my master study. When I first started my master program, I never imagined that I would write a thesis on this challenging topic. However, I was so fortunate to have my graduation committee chaired by Prof. Dr. Alexander Verbraeck with first supervisor Dr. Yilin Huang. I would not have grown as much as I am without their support and advice throughout the research.

Though the thesis is officially a result of the last six months, it sums up my last two years in the Netherlands. It contains all of my learning process, trial and error and struggles that I have experienced. It would never be possible to accomplish this challenge without my beloved family, especially my wife Soonyeon, who has given me endless support. I want to take this opportunity to express my sincere thanks for respecting the hard decision I made and joining this tough adventure. I also thank my daughter, the greatest gift in my life. I was able to keep moving toward the goal without wandering thanks to the everyday bliss she gave.

Lastly, I thank all my friends who have been struggling together for the past two years. I appreciate the time that we have inspired and been inspired by sometimes sharing small jokes or sometimes serious discussions.

Yubin Cho
August 2019
Modelling simulation is one of the standard methods to study the system of interest. The real-world system is represented by a model to understand the internal process and analyzed by conducting simulation experiments. Recently, however, many socio-technical challenges are derived from the increasing scale and complexity of the system. The so-called "wicked problem" [6, 13, 22] requires a different approach to address these challenges. Dynamic Data-Driven Simulation (DDDS) [14] is one of the frameworks proposed, given that the availability of real-time data is rapidly growing along with the recent development of IoT technology. Unlike the traditional simulation, new data continuously influence the simulation model in the DDDS. In order to incorporate the data in real-time, it is critical to apply the data assimilation, which combines a numerical model with measured data[5, 20].

It is the continuous system that has mainly used the data assimilation, and yet a few research can be found in the Discrete Event System Specification (DEVS). State-of-the-art research focused on how to adapt the data assimilation given the difference between the continuous and the discrete system. [30] developed a framework which adapts the data assimilation technique to the discrete event simulation. A sensitivity analysis was also conducted to study the impact of several factors on the result of the data assimilation.

Nevertheless, there are some other factors less highlighted in the current literature. One is the time interval of the real-time data update, which is no longer a given condition, but a matter of choice given the increased data availability. Furthermore, the number of particles can be constrained by the time interval given the nature of the real-time simulation. For this reason, one needs to find an optimal allocation of the computational resource to the time interval and the number of particles, however, their relation is not well-known. Finally, even though the correct profile of the measurement error can never be known, the limited knowledge about the measurement error was not considered in previous research.
Research Objective

The thesis addresses the following research question based on the research gaps.

**Research Question**

How is the estimation accuracy of the data assimilation affected by the different strategies of using real-time data and assumptions?

Specifically, it conducts a sensitivity analysis to examine the impact of the time interval, the number of particles and the limited knowledge of the measurement error on the estimation accuracy. The distance correlation between the estimated trajectory and the true one, the proxy of the similarity is measured to represent the estimation accuracy. The structure of the analysis is visualized in Figure 1. It discovers the impact of the different factors by varying the value of the time interval, the number of particles, the real measurement error and the perception of the measurement error.

![Figure 1: Structure of experiment in this research](image1)

In order to conduct the sensitivity analysis, the thesis constructs a DDDS application of the simple queuing model visualized in Figure 2. It develops a data assimilation framework for the simple queuing model by integrating a real hardware device and its simulation model.

![Figure 2: Structure of Dynamic Data Driven Simulation in this research](image2)
Research Findings

First, the impact of the time interval and the number of particles are discovered as shown in Figure 3 and 4. Increasing the computation - the shorter time interval and the more particles - is effective to obtain the more accurate estimate, however the impact of the number of particles becomes insignificant as it exceeds 300.

![Figure 3: Time Interval versus Estimation Accuracy](image)

![Figure 4: Number of Particles versus Estimation Accuracy](image)

Then their relation is examined. The large number of particles does not always produce the accurate estimation whereas the short time interval (i.e. the frequent update of the real-time data) exhibits the high accuracy across the wide range of the particle number. Therefore, if the number of particles is already sufficient, it is recommended to use the rest of the computation to shorten the time interval. The result is displayed in Figure 5.
Finally, the impact of the limited knowledge of the measurement error is investigated. As shown in the Figure 6, the perfect knowledge does not result in the best estimation accuracy while a certain over-estimation of the error produces a better estimate.
Future work

One can utilize the DDDS framework of the simple queuing system under the various scenarios and different assumptions. A large variety of the system change can be tested as the research incorporated a specific scenario of the internal change. A different real queuing process with less knowledge can also be used since the research used the simulation model whose operational rule is same as the real hardware. Moreover, the simple queuing system can be scaled up to a larger and more complex system to verify the conclusion further.
# Contents

Preface

Executive Summary

List of Figures

List of Tables

1 Introduction

1.1 Research Motivation ........................................... 1
1.2 Research Question ........................................... 4
1.3 Organization of the thesis .................................... 5

2 Background

2.1 Dynamic Data Driven Simulation (DDDS) ......................... 7
2.2 Particle Filter .................................................. 9
   2.2.1 Bayesian Filtering ....................................... 9
   2.2.2 Sequential Monte Carlo Method ......................... 11

3 Case Study - Single Queuing System

3.1 Queuing Model ............................................... 17
3.2 Scenario Description .......................................... 18
3.3 Implementation of Dynamic Data Driven Simulation .............. 19
   3.3.1 Real System ............................................. 20
   3.3.2 Measurement Model .................................... 22
   3.3.3 Simulation Model ...................................... 23
   3.3.4 Data Assimilation ..................................... 23
3.4 Sensitivity Analysis .......................................... 25
   3.4.1 Time Interval ........................................... 25
   3.4.2 Number of Particles .................................... 26
   3.4.3 Measurement Error ...................................... 26
   3.4.4 Metric of the Estimation Accuracy ..................... 28
# Result

4.1 Entire Range of the Estimation Accuracy ........................................... 31
4.2 High Estimation Accuracy ............................................................... 32
4.3 Low Estimation Accuracy ............................................................... 34
4.4 Sensitivity Analysis ................................................................. 35
   4.4.1 Feature Importance ................................................................. 36
   4.4.2 Effect of the Time Interval ....................................................... 36
   4.4.3 Effect of the Number of Particles ........................................... 37
   4.4.4 Effect of the Measurement Error in Reality .............................. 40
   4.4.5 Effect of the Perception of the Measurement Error ................... 40

# Conclusion

5.1 Answering the Research Question ..................................................... 43
5.2 Discussion ................................................................. 44
5.3 Limitation and Future work .......................................................... 46

# Replication Guide

A.1 Requirements ............................................................... 49
A.2 Implementation ................................................................. 50
A.3 Visualization ................................................................. 52

Bibliography ................................................................. 55
## List of Figures

1. Structure of experiment in this research ................................................. 5
2. Dynamic Data Driven Simulation Based on SMC Methods [14] .................... 8
3. State transition of continuous system .................................................. 9
4. Hidden Markov Process ........................................................................ 9
5. Single Queuing Node ........................................................................... 18
7. Structure of Dynamic Data Driven Simulation in this research .................. 19
8. ESP32 ................................................................................................. 20
9. Distribution of Distance Correlation of 4,800 experiments .......................... 31
10. Best estimation of arrival rate and processing rate .................................. 32
11. Best estimation of length of queue ...................................................... 32
12. Estimation of arrival rate and processing rate with smaller number of particles .................................................................................. 33
13. Estimation of length of queue with smaller number of particles .............. 33
14. Estimation of arrival rate and processing rate with correct assumption ...... 33
15. Estimation of length of queue with correct assumption ........................... 34
16. Worst estimation of arrival rate and processing rate ............................... 34
17. Worst estimation of length of queue .................................................... 34
18. 2nd worst estimation of arrival rate and processing rate .......................... 35
19. 2nd worst estimation of length of queue .............................................. 35
20. Impact of different settings on estimation accuracy in relative scale ......... 36
21. Time Interval versus Estimation Accuracy ............................................ 36
22. 95% Confidence Interval of the different time interval ........................... 37
23. Number of Particles versus Estimation Accuracy ................................. 38
24. Trade-off between the time interval and number of particles ................. 39
25. Measurement Error (real) versus Estimation Accuracy ........................... 40
26. Measurement Error (perception) versus Estimation Accuracy ................. 41
27. Impact of knowledge about error under different measurement error ....... 41
28. Impact of knowledge about error under different measurement error across four scenarios .................................................................................. 42
5.1 Different choice of errors in the measurement model 45
A.1 Full trajectory of the variables 52
A.2 Distribution of the particles at time = 6 53
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The terms and their description used in this research</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Notation of variables</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>ESP32 Specification</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Standard Deviation $\sigma$ of the measurement error</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Effect of the time interval on the estimation accuracy</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Effect of the number of particles on the estimation accuracy (partially shown)</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Partial result of Tukey’s test for $n_{pf} = 300$</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Partial result of Tukey’s test for $n_{pf} = 400$</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Effect of the measurement error in reality on the estimation accuracy</td>
<td>40</td>
</tr>
<tr>
<td>4.6</td>
<td>Effect of the Perception of the Measurement Error</td>
<td>42</td>
</tr>
<tr>
<td>A.1</td>
<td>Result of the Experiment (partial)</td>
<td>52</td>
</tr>
</tbody>
</table>
Introduction

1.1. Research Motivation

Modelling& Simulation (M&S) is one of the standard methods to study the system of interest. It gives us a better understanding of the behavior of that system for a given set of conditions [17]. In practice, the physical system is observed and measured to collect data. Based on the data, a model representing a real-world process is developed to understand, simulate, and comprehend the subject. Then, the simulation experiment is conducted under various scenarios to obtain a set of dynamic behaviors. Finally, the simulation outcome provides with adequate knowledge to evaluate multiple operational strategies, prioritize investments under constraints, and eventually make a better decision.

However, the traditional approach may not be sufficient when designing a policy for a large-scale complex problem, also referred to as wicked-problem or social mess [6, 13, 22]. The core characteristic of such a problem is that numerous interrelated components (i.e., subsystems) form a large structure and dynamically evolve. Consequently, it is extremely tough, if possible, to grasp a holistic view of the problem as [1] mentioned it as "a system of problems."
short, it is hard to make a complete observation and accurate measurement because of its large scale, dynamics, and inter-dependency of subcomponents. The simulation conducted with such incomplete data will diverge from the real behavior of the system [8] and produce misleading information for policymakers. Considering the substantial economic and social cost caused by the wrong policy, the issue is pronounced.

Since complete knowledge of the complex system is hard to obtain through the traditional approach, a new paradigm, the so-called Dynamic Data Driven Application System (DDDAS) [7] emerged. The central concept is that the computation and instrumentation aspects of the system are dynamically integrated. In this way, the system incorporates additional data measured in real-time during the execution and, in reverse, it dynamically steers the measurement process. As the availability of real-time data is rapidly growing along with the recent development of IoT technology, it is empowering many applications. [14] also proposed Dynamic Data Driven Simulation (DDDS) framework, which new data continuously influence the simulation model. The data assimilation [5, 20] is referred to as a core building block of DDDS since its central research topic is to combine a numerical model with measurements. The data assimilation aims to obtain a model state that most accurately approximates the current and future states of the true system and to train the model parameters based on the observed data under the incompleteness of both measurement and model [25].

Continuous system applications have been dominant fields so far using the DA technique due to its origin from the weather forecast domain. Although it is a few, some remarkable studies also can be found in the non-continuous system-based simulation. The examples are incorporating measurement [32] and additional data sources [11] to obtain a precise estimate on Discrete Event System Specification (DEVS) -based wildfire simulation. Likewise, [28, 29] proposed the data assimilation framework for agent-based simulation to predict residents’ behavior inside the building. Apart from the domain-specific literature, [30, 31] focused on implementing the adapted data assimilation technique in DEVS simulation based on Sequential Monte Carlo (SMC) method, also known as Particle Filter. Indeed, most of the literature for the non-continuous system application adopted Particle Filter because of two reasons. First, distinguished from the variational method, it is a sequential method which better fits to the DDDS’s nature in that the model incorporates real-time data sequentially as the measured data arrives. Secondly, the non-linear and -Gaussian behavior of complex simulation models make it difficult to apply the conventional sequential methods (e.g., Kalman Filter and its extensions) due to their strict requirements.

A great deal of prior work of the data assimilation in DEVS-based simulation focused on how DEVS should adopt the data assimilation and what benefit may exist. However, what has been commonly done in continuous system applications has not yet been done in DEVS. The time interval of the measurement data is one of the critical parameters in the data assimilation
since it directly affects a computational requirement. While many prior research in continuous applications, for example, meteorology, geophysics and oceanography [19, 21, 24, 27] studied the influence of different time interval on the data assimilation, little is known for DEVS-based simulation. One of the most recent and extensive research of [30] only covered the data quality, the model errors, and the number of particles. The convention found in the literature is using a fixed time interval depending on the data availability. Considering the increased data availability, however, the thesis argues that the time interval can be seen as a matter of choice rather than a given condition. A clear understanding of the influence of the time interval is required since it will give a better insight of how often the measurement should be fed into the numerical model.

The current literature of the data assimilation in DEVS lacks an empirical examination on how the time interval may affect the estimation accuracy.

The number of particles also determines the simulation replication required. [11] emphasized the importance of the number of particles that a large number can improve the convergence of the estimation accuracy, but greatly decrease the performance. Moreover, the impact of the number of particles on the estimation accuracy of the data assimilation in DEVS is studied by [30]. However, existing literature has not highlighted its impact along with the time interval. In general, the choice of the number of particles was made irrespective to the time interval. In addition to the fact that the time interval is no longer a given condition, the thesis also focuses on the other one. It is that the choice of the time interval will restrict the possible number of simulation runs. In DDDS the measured data arrives in real-time continuously unlike the traditional simulation and every arrival of the data should be processed before the next arrival. Otherwise, the delay of predicting the next state will occur, and therefore, will be accumulated while the real-time data keeps arriving. In order to prevent this issue, the computational resource should be employed as to be enough for running multiple simulation runs within the interval. The research associates these two factors to understand their relation and how the computation should be allocated for both the number of particles and the size of the update interval optimally.

Existing works have not highlighted the relation between the time interval and the number of particles under the computational constraint.

Finally, most of the literature has modelled the measurement error under the assumption
of perfect knowledge in the measuring process. However, it is not always possible to know the exact profile of the measurement error occurred in reality. For example, a modeller may assume the sensor’s measurement error is low, while the actual performance of the sensor is poor. In order to obtain a valid result of the data assimilation, Since there is always a risk of mismatch between our perception and the reality, it is important to understand how the result of the data assimilation may vary depending on this mismatch.

Research Gap 3

In the previous research, the limited knowledge of the measurement error is not taken into account.

To summarize, this thesis is motivated by the promising potential of DDDS and the data assimilation, but the limited knowledge in the state-of-the-art literature. In the next section, the overarching research question is presented.

1.2. Research Question

The research gaps proposed in the previous section are encapsulated into the following main research question.

Research Question

How is the estimation accuracy of the data assimilation affected by the different strategies of using real-time data and assumptions?

In the thesis, the ”data assimilation result” indicates the estimation accuracy. It is a degree of similarity between the ground truth value and the estimate. The precise definition is explained in the subsection 3.4.4. The term ”strategy” includes the time interval, the number of particles, and the way of modeling the measurement error. The Table 1.1 is used to clarify these terms and their notions.

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The time interval</td>
<td>Time difference between the arrivals of measured data</td>
</tr>
<tr>
<td>The number of particles</td>
<td>Number of simulation replications within the time interval</td>
</tr>
<tr>
<td>The perception of the measurement error</td>
<td>Degree of errors assumed by modellers</td>
</tr>
<tr>
<td>The measurement error in reality</td>
<td>Degree of errors occurred during the measuring process</td>
</tr>
<tr>
<td>The estimation accuracy</td>
<td>Similarity between the ground truth value and the estimate</td>
</tr>
</tbody>
</table>

Table 1.1: The terms and their description used in this research

Specifically, the impact of the time interval on the estimation accuracy will be examined
along with the number of particles to verify the trade-off in between. Moreover, the thesis focuses on two different types of the measurement errors. One is the measurement error actually occurred in reality whose true profile can never be known. The other is the perception (i.e. assumption) of modellers about how the measurement error would be. Considering that a mismatch between these two errors can always exist, this research spotlights the limited knowledge of the measurement error critically.

In order to examine how does the estimation accuracy of the data assimilation vary, the research performs a sensitivity analysis as shown in Figure 1.1.

![Figure 1.1: Structure of experiment in this research](image)

The outcome of the research is in twofold. First, it suggests how to select the time interval and the number of particles for a better estimation accuracy under the computational constraint. Secondly, it gives a more in-depth understanding of how to handle the limited knowledge of the measurement error.

### 1.3. Organization of the thesis

The rest of the thesis is structured as follows. In the next Chapter 2, the background of DDDS and its core component, the data assimilation will be presented. Chapter 3 demonstrates a case study - a simple queuing system. It elaborates the configuration used in this thesis to realize DDDS. Then, Chapter 4 described the outcome of the sensitivity analysis. Finally, Chapter 5 answers the research question. It also suggests some practical recommendation with possible future research.
2.1. Dynamic Data Driven Simulation (DDDS)

The relation between real systems and simulation models is rather static in traditional M&S study. Once the model is built based on data collected, the simulation generates outcomes of interest without further interaction with real systems. However, since the real system dynamically and constantly changes over time, making use of real-time data has a great potential to enhance the quality of simulation analysis and prediction. DDDS is a new paradigm proposed by [14] to exploit the potential. (See Figure 2.1.)

The main feature of DDDS is that the simulation model incorporates new data in real-time during the execution, enabled by the close linkage and bi-directional connectivity to real systems. The idea of connecting the instrumentation aspect of the real process to the computational model is also studied by [7], namely Dynamic Data-driven Application System (DDDAS). While the main focus of DDDAS framework is the conceptual connection and feedback between instrumentation and computation, DDDS also focuses on the specific methodology of incorporating the real-time data to the simulation model, namely, data assimilation. The tech-
Background

Figure 2.1: Dynamic Data Driven Simulation Based on SMC Methods [14]

Technique was first used in numerical weather forecast domain [16] and now widely used across diverse domains. In DDDS, the (noisy) measurement of the observed data is assimilated into the model to produce time-series data of (optimal) state estimates [5]. In particular, this thesis focuses on the sequential method instead of the variational method due to its relevance. In a more general viewpoint, the technique can be regarded as recursive Bayesian estimation (i.e., Bayesian filtering), which keeps updating the predictive distribution by using the data to generate the posterior distribution [23]. It is equivalent to the process which the simulation model incorporates new data in real-time when the measurement becomes available. Among the diverse data assimilation techniques, the Kalman filter [15] is the most widely-used method for Data Assimilation in many fields, including meteorology, geoscience, signal processing, transportation, and robot localization. The primary assumption of the Kalman filter is that the real process is linear and Gaussian [9]. Therefore, it is suitable for the system where the transition from the previous to the next state is normally distributed [23]. The continuous trajectory of a robot roaming in a space is one representative case (See Figure 2.2) in which the state transition is characterized by its movement (i.e., change of location). The next location of the robot depends on the previous location while the system error due to diverse sources, such as a slippery floor. Since the robot moves continuously over the space, Gaussian error assumption is valid. In other words, the posterior of the next location follows the normal distribution.

On the other hand, the assumption is easily violated in DEVS, as the state of the system is non-linear and shows a jump-like behavior [32]. As an alternative of the Kalman filter and its extensions, the Sequential Monte Carlo (SMC) method (also known as Particle Filter) was first applied in DEVS-based application and has widely used in most of the current state-of-the-art literature [11, 28–32]. Due to the importance, the next section 2.2 will discuss the SMC
method in more details to demonstrate why this approach is more suitable in DDDS, especially for non-linear and non-Gaussian system application.

2.2. Particle Filter

Among various state estimation techniques, Sequential Importance Resampling (SIR) described in section 2.2.2 is referred to as Particle Filter, specifically. It is one particular case of the more general framework, Recursive Bayesian Estimation (also known as Bayes’ Filter or Bayesian Filtering) which can be represented as Bayesian Network of a Hidden Markov Model (HMM).

2.2.1. Bayesian Filtering

The term optimal filtering traditionally refers to a class of methods that can be used for estimating the state of a time-varying system which is indirectly observed through noisy measurements [23]. It assumes that the state transition is a hidden Markov process, and the measurements are the observed variable in HMM. Figure 2.3 shows this structure.

The unobserved (hidden) variable $X_k$ at each step $k$ is the dynamic system state which is
modelled in a discrete-time equation with the function $g$:

$$X_k = g(X_{k-1}) + W_{k-1}$$  \hspace{1cm} (2.1)$$

where $W$ is the process noise representing the uncertainty in the system dynamics. Likewise, the observed variables $Z$ is the noisy measurement also modelled in a discrete-time equation:

$$Z_k = h(X_k) + V_k$$  \hspace{1cm} (2.2)$$

where the function $h$ maps the system state to the measurement and $V$ indicates the measurement noise. Two critical properties of HMM also hold in Bayesian Filtering. First, the conditional probability of the current state $X_k$ given the previous state $X_{k-1}$ is conditionally independent to other state variables and measurements, as shown in 2.3.

$$p(x_k|x_{0:k-1}, z_{1:k}) = p(x_k|x_{k-1})$$ \hspace{1cm} (2.3)$$

Secondly, the conditional probability of the current observation given the current state is conditionally independent to other state variables and measurements, as shown in 2.4.

$$p(z_k|x_{0:k-1}, z_{1:k-1}) = p(z_k|x_k)$$ \hspace{1cm} (2.4)$$

While the full joint probability distribution of all variables, namely $p(x_{0:k}, z_{1:k})$ can be readily expressed, the objective of Bayesian Filtering is, however, to obtain $p(x_k|z_{1:k})$, the conditional probability distribution of the current state given all the measurements. It is recursively computed by through the following three steps: initialize, predict, and update [23]. First, the prior distribution is defined.

<table>
<thead>
<tr>
<th>Initialize</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x_0) \equiv p(x_0</td>
</tr>
</tbody>
</table>

where $z_0$ indicates that there is no observation.

Next, the predictive distribution, which is the prior belief about $x_k$ before observing the measurement $z_k$, can be computed by using the Chapman–Kolmogorov equation.
Predict

\[ p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1}) \, dx_{k-1} \]  

(2.6)

given that \( p(x_k|x_{k-1}) \) is the system transition density constructed from 2.1 and \( p(x_{k-1}|z_{1:k-1}) \) is already obtained from the previous step \( k-1 \).

And then, the posterior distribution is updated as a new measurement \( z_k \) arrives.

Update

\[ p(x_k|z_{1:k}) = \frac{\text{likelihood} \cdot \text{prediction}}{\int p(z_k|x_k)p(x_k|z_{1:k-1}) \, dx_k} \]  

(2.7)

where the likelihood \( p(z_k|x_k) \) can be obtained from the measurement model described in 2.2, and the denominator is a normalizing constant.

These equations provide a general idea and method to compute the posterior distribution of the state estimation. However, an analytical solution is only possible under a highly limited condition. For example, the Kalman Filter is the analytical solution applicable if both the system and measurement model are linear Gaussian. Some of its extensions based on numerical approach still result in Gaussian approximation. If such assumptions are violated, for example, multi-modal distributions or discrete state variables, a better alternative approach is sample-based technique [23].

2.2.2. Sequential Monte Carlo Method

Monte Carlo sampling and Weight

The key idea of Sequential Monte Carlo (SMC) method is to represent the required posterior density function \( p(x_k|z_{1:k}) \) by a set of random samples (i.e. particles) \( x^i \) with associated weights \( w^i \) [3], namely \( \{(x^i, w^i)|i = 1, 2, \ldots, N \} \). It estimates the state based on these samples and weights. The idea of weight is introduced along with a sampling technique, importance sampling.

Importance Sampling

Given that the purpose of Monte Carlo sampling is to approximate the expected value by using a large number of samples, the expected value of an arbitrary function \( g(x) \) can be approximated as following [23]:

\[ E[g(x)] = \int g(x)p(x) \, dx = \frac{1}{N} \sum_{i=1}^{N} g(x^i) \]  

(2.8)
where \( x^i (i = 1, 2, \ldots N) \) are drawn from the probability density function \( p(x) \). Considering that direct sampling from \( p(x) \) is difficult, an approximate distribution called importance distribution \( \pi(x) \) is used, from which samples can be easily drawn.

\[
E[g(x)] = \int g(x) p(x) \, dx = \int \left[ g(x) \frac{p(x)}{\pi(x)} \right] \pi(x) \, dx
\]

\[
\approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(x^i)}{\pi(x^i)} g(x^i) = \frac{1}{N} \sum_{i=1}^{N} w^i g(x^i)
\]

where the weight is defined as:

\[
w^i = \frac{p(x^i)}{\pi(x^i)}
\]  

(2.9)

By substituting \( p(x) \) to \( p(x | z_{1:k}) \), the weight of samples drawn from \( \pi(x | z_{1:k}) \) is derived as follows:

\[
E[g(x) | z_{1:k}] = \int g(x) p(x | z_{1:k}) \, dx
= \int g(x) p(x | z_{1:k}) \pi(x | z_{1:k}) \, dx
\]

\[
= \frac{\int \frac{p(z_{1:k} | x) p(x)}{\pi(x | z_{1:k})} g(x) \, dx}{\int \frac{p(z_{1:k} | x) p(x)}{\pi(x | z_{1:k})} \, dx}
= \frac{\int \frac{p(z_{1:k} | x) p(x)}{\pi(x | z_{1:k})} \pi(x | z_{1:k}) \, dx}{\int \frac{p(z_{1:k} | x) p(x)}{\pi(x | z_{1:k})} \, dx}
\]

\[
\approx \frac{\sum_{i=1}^{N} \left[ \frac{p(z_{1:k} | x^i) p(x^i)}{\pi(x^i | z_{1:k})} \right] g(x^i)}{\sum_{j=1}^{N} \frac{p(z_{1:k} | x^j) p(x^j)}{\pi(x^j | z_{1:k})}}
\]

(2.10)

The underbraced term \( w^i \) can be calculated as the fraction of the unnormalized weight \( \frac{p(z_{1:k} | x^i) p(x^i)}{\pi(x^i | z_{1:k})} \) over the summation of all the weights of samples.

**Sequential Importance Sampling (SIS)**

As the main focus of Bayesian Filtering, \( p(x_k | z_{1:k}) \) is approximated by repeating the importance sampling recursively as a new observation becomes available one after the other. Starting from \( p(x_{0:k} | z_{1:k}) \) (i.e. the posterior distribution of all states given all observations), one can
obtain the recursive form. (Note that the property of HMM given by 2.3 and 2.4 is used.)

\[
p(x_{0:k} | z_{1:k}) = p(x_{0:k} | z_{1:k-1}, z_k) = \frac{p(z_k | x_{0:k})p(x_{0:k}, z_{1:k-1})}{p(z_{1:k-1}, z_k)}
\]

By using the same line of reasoning in 2.10, the weight of each sample, drawn from \( \pi(x_{0:k} | z_{1:k}) \) at each time step is given by

\[
w^i_k \propto \frac{p(z_k | x^i_k)p(x^i_k | x^i_{k-1})p(x^i_{0:k-1} | z_{1:k-1})}{\pi(x^i_0 | z_{1:k})}
\]

(2.11)

Since the process is recursive, a set of samples which approximates \( x_{0:k-1} \) are already obtained from \( \pi(x_{0:k-1} | z_{1:k-1}) \) during the previous step. If the importance distribution is chosen to be recursive as follows

\[
\pi(x_{0:k} | z_{1:k}) = \pi(x_k | x_{0:k-1}, z_k) \quad \text{(new state)}
\]

\[
\pi(x_{0:k-1} | z_{1:k-1}) \quad \text{(already obtained)}
\]

(2.12)

then, all we need to sample from \( \pi(x_{0:k} | z_{1:k}) \) is that the new state \( x_k \sim \pi(x_k | x_{0:k-1}, z_k) \) to be available. By inserting the assumption 2.12 to 2.11, the weight is given by

\[
w^i_k \propto \frac{p(z_k | x^i_k)p(x^i_k | x^i_{k-1})p(x^i_{0:k-1} | z_{1:k-1})}{\pi(x^i_0 | z_{1:k})} \times w^i_{k-1}
\]

and the equation satisfies the recursion. Accordingly, the samples \( x^i_k \) drawn from \( \pi(x_k | x_{0:k-1}, z_k) \) and their respective weights calculated by 2.11 is used to approximate \( p(x_k | z_{1:k}) \). The entire process of SIS is described in Algorithm 1.

Moreover, it is convenient to select the importance density to satisfy Markovian assumption [23] such that \( \pi(x^i_k | x^i_{0:k-1}, z_{1:k}) = \pi(x^i_k | x^i_{k-1}, z_{1:k}) \) since only a previous state \( x_{k-1} \) is
Algorithm 1: Generic Sequential Importance Sampling [9]

// Initialize
1 for $i = 1$ to $N$ do
2  Draw i-th sample from the prior: $x_i^0 \sim p(x_0)$
3  Set each weight evenly: $w_i^0 \leftarrow 1/N$
4 end

// Estimate
5 for $i = 0$ to $k$ do
6   /* Prediction Step */
7      for $i = 1$ to $N$ do
8          Draw i-th sample from the importance density: $x_i^k \sim \pi(x_k|x_{0:k-1}^i, z_{1:k})$
9      end
10   /* Update Step */
11      Update each weight according to
12          $w_i^0 \leftarrow \frac{p(z_k|x_k^i)p(x_k^i|x_{0:k-1}^i)}{\pi(x_k^i|x_{0:k-1}^i, z_{1:k})} \times w_i^{k-1}$
13      Normalize the weight by dividing over the sum: $w_i^k \leftarrow \frac{w_i^k}{\sum_{i=1}^N w_i^k}$
14 end
15 /* A set of $N$ particles at time step $k$, $(x_i^k, w_i^k)$ are obtained */

required at each time step to estimate $x_k$ and thereby the entire state trajectory can be discarded.

Sequential Importance Resampling (SIR)
One major problem in the SIS algorithm is degeneracy problem, that almost all the particles may have zero or nearly zero weights after a few iterations [3]. Resampling is a common approach to solve the issue. The idea of the resampling procedure is to remove particles with very small weights and duplicate particles with large weights [23] given by Algorithm 2.

Algorithm 2: Resampling [9]
1 Construct a discrete probability distribution given by
2

\[ P(X = x_k^i) = w_k^i \]

in which the weight of each particle is interpreted as the probability of obtaining the sample.
3 Draw $N$ new particles from the discrete distribution and replace the old ones.

The procedure in which the resampling is added to SIS is called Particle Filter [9, 10, 18] as described in Algorithm 3.

In the SIR, the choice of importance density determines the performance [23](Fearnhead,
Algorithm 3: Sequential Importance Resampling (Particle Filter) [23]

// Initialize
for \(i = 1\) to \(N\) do
  Draw \(i\)-th sample from the prior: \(x^i_0 \sim p(x_0)\)
  Set each weight evenly: \(w^i_0 \leftarrow 1/N\)
end

// Estimate
for \(i = 0\) to \(k\) do
  /* Prediction Step */
  for \(i = 1\) to \(N\) do
    Draw \(i\)-th sample from the importance density: \(x^i_k \sim \pi(x_k|x^i_0:k-1, z_{1:k})\)
  end
  /* Update Step */
  for \(i = 1\) to \(N\) do
    Update each weight according to
    \[ w^i_k \leftarrow \frac{p(z_k|x^i_k) p(x^i_k|x^i_{k-1})}{\pi(x^i_k|x^i_0:k-1, z_{1:k})} \times w^i_{k-1} \]
    Normalize the weight by dividing over the sum:
    \[ w^i_k \leftarrow \frac{w^i_k}{\sum_{i=1}^{N} w^i_k} \]
  end
  /* A set of \(N\) particles at time step \(k\), \((x^i_k, w^i_k)\) are obtained */
  /* Resampling */
end
Perform resampling given by Algorithm 2

and Clifford, 2003). One common way is bootstrap filter described in Algorithm 4.

It chooses the system transition function \(p(x^i_k|x^i_{k-1})\) as the importance density function \(\pi(x^i_k|x^i_0:k-1, z_{1:k})\). Despite a certain level of inefficiency, the approach is widely used in many applications due to its simplicity, in particular, the weight calculation.

\[
    w^i_k \propto \frac{p(z_k|x^i_k) p(x^i_k|x^i_{k-1})}{\pi(x^i_k|x^i_0:k-1, z_{1:k})} \times w^i_{k-1}
    = \frac{p(z_k|x^i_k) p(x^i_k|x^i_{k-1})}{p(x^i_k|x^i_{k-1})} \times w^i_{k-1}
    = p(z_k|x^i_k) \times w^i_{k-1}
\]

When using DEVS-based models for Particle Filter, the importance sampling is done through the time advancement of simulation by the time step \(k\), and the weight is calculated by using the measurement model. Section 3.3 will provide more details regarding the implementation of Particle Filter to the specific case.
Algorithm 4: Bootstrap Filtering [3]

// Initialize
for i = 1 to N do
  Draw i-th sample from prior: $x^i_0 \sim p(x_0)$
  Set each weight evenly: $w^i_0 \leftarrow 1/N$
end

// Estimate
for i = 0 to k do
  /* Prediction Step */
  for i = 1 to N do
    Draw i-th sample $x^i_k$ from the system transition function
    $x^i_k \sim p(x_k | x^i_{k-1})$
  end
  /* Update Step */
  for i = 1 to N do
    Update each weight according to
    $w^i_0 \leftarrow p(z_k | x^i_k) \times w^i_{k-1}$
    Normalize the weight by dividing over the sum
    $w^i_k \leftarrow \frac{w^i_k}{\sum_{i=1}^{N} w^i_k}$
  end
  /* A set of N particles at time step k, $(x^i_k, w^i_k)$ are obtained */
  /* Resampling */
  Perform resampling given by Algorithm 2
end
Case Study - Single Queuing System

3.1. Queuing Model

The research uses one of the simplest models in common DEVS-based application, Queuing Model. It is based on Queuing theory, which is the mathematical study of waiting for lines to predict queue lengths and waiting time [12]. It helps to make a decision for allocating the resource efficiently so that the balance between the cost of service and the amount of waiting can be achieved. In the single queuing node shown in Figure 3.1, jobs (e.g., customers, parts) arrive from the left, occupy a server or processor (e.g., cashiers, machine), and depart. If jobs arrive while the server is fully occupied by the earlier ones, then the later ones form a queue in which they wait until the server becomes available. The process is specified with a couple of several parameters, namely inter-arrival time, processing time, processing capacity, and maximum length of the queue.
3.2. Scenario Description

The general queuing model shown in Figure 3.1 is specified as the following process.

**Real Process**

- Arrival: Jobs arrive one at a time, and their inter-arrival time follows a stochastic process. The exponential distribution with mean $\lambda$ is used to model the arrivals. ($\lambda_{\max} = 20$)

- Server: The processing time is determined stochastically by the exponential distribution with mean $\mu$. ($\mu_{\max} = 20$)

- Queue: There is a specific limit $L$, which the queue can accommodate the jobs. It is assumed that if there is no room for jobs to wait, the job immediately leaves the system without being served. In specific, it is referred to as "balking" behaviour [2].

The parameters of the system are not static, but evolve, given that the objective of Data Assimilation is the state estimation of dynamic systems. The parameters $\lambda$ and $\mu$ are modeled as dynamic variables which display random walk behavior along a certain boundary. The maximum rate of both the arrival and the departure is 20.

Furthermore, the system is not fully observable, but only partially observable. Data Assimilation technique makes an optimal estimate of the system under the limited observational condition.

**Partial Observation**

The arrival and departure of jobs are counted. However, the rest parts of the system are non-observable and treated as a black box.
3.3. Implementation of Dynamic Data Driven Simulation

The research constructs DDDS application to experiment with which various input variables pairs to respective outputs. The result of the experiment is analyzed to investigate how the estimation accuracy diverges under different settings. The application consists of essential components of the DDDS framework: Real System, Simulation Model, Measurement Model, and Data Assimilation. Since some parts of Figure 2.1 of [14] is not absolutely consistent to the DA in many literature, the research makes a partial modification as shown in Figure 3.2.

![Figure 3.2: Dynamic Data Driven Simulation modified from [14]](image)

Based on this structure, DDDS for a single queueing system is constructed. A hardware device representing a dynamic system in real-world generates a set of real-time data and it is partially observed. The measured data is combined with the simulation results to make an estimate of the system state. In order to compare the estimation with the ground truth value, there is another channel to store the entire state of the system. In particular, the values are retrieved into mySQL server which is completely separated. Figure 3.3 shows the overall configuration.

![Figure 3.3: Structure of Dynamic Data Driven Simulation in this research](image)
As described in the previous chapter, Particle Filter is chosen given its less restriction and fewer assumptions to apply. Table 3.1 describes the notation of each variable used in this section.

<table>
<thead>
<tr>
<th>Name</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{t,real}$</td>
<td>-</td>
<td>state of the real system at time $t$</td>
</tr>
<tr>
<td>$p()$</td>
<td>-</td>
<td>transition function of the real system</td>
</tr>
<tr>
<td>$S_{t,sim}$</td>
<td>-</td>
<td>state of the simulation system at time $t$</td>
</tr>
<tr>
<td>$sim()$</td>
<td>-</td>
<td>transition function of the simulation system</td>
</tr>
<tr>
<td>$arrRate$</td>
<td>entity / s</td>
<td>arrival rate of the entity</td>
</tr>
<tr>
<td>$procRate$</td>
<td>entity / s</td>
<td>processing rate of the server</td>
</tr>
<tr>
<td>$queLen$</td>
<td>entity</td>
<td>number of entities waiting in a queue</td>
</tr>
<tr>
<td>$numArr$</td>
<td>entity</td>
<td>number of entities arrived in the system within a given time interval</td>
</tr>
<tr>
<td>$numDep$</td>
<td>entity</td>
<td>number of entities left the system within a given time interval</td>
</tr>
<tr>
<td>$measurement_t$</td>
<td>-</td>
<td>measured data at time $t$ from the measurement model</td>
</tr>
<tr>
<td>$prediction_i$</td>
<td>-</td>
<td>predicted data of particle $i$ at time $t$</td>
</tr>
<tr>
<td>$w_i^t$</td>
<td>-</td>
<td>weight of the particle $i$ at time $t$</td>
</tr>
</tbody>
</table>

Table 3.1: Notation of variables

3.3.1. Real System

A hardware device is required to represent the real queuing process and to generate the corresponding data in real-time. Moreover, having a wireless interface is preferred since the ground truth values of all the state variables can be stored separately. Accordingly, this thesis chooses ESP32 in Figure 3.4, one of the standard microcontroller units. (See Table 3.2 for the specification.)

![ESP32 Development Board](image1)

![Schematic Diagram](image2)

Figure 3.4: ESP32
### Table 3.2: ESP32 Specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>Xtensa dual-core 32-bit LX6, 240 MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>520 KiB SRAM</td>
</tr>
<tr>
<td>Wireless connectivity</td>
<td>Wi-Fi: 802.11 b/g/n, Bluetooth: v4.2 BR/EDR and BLE</td>
</tr>
<tr>
<td>Peripheral interfaces:</td>
<td>12-bit SAR ADC up to 18 channels, 2 × 8-bit DACs, 10 × touch sensors, 4 × SPI, 3 × UART, 2 × I²C interfaces, SD/SDIO/CE-ATA/MMC/eMMC host controller, SDIO/SPI slave controller, Ethernet MAC interface with dedicated DMA, CAN bus 2.0, Infrared remote controller, Motor, LED PWM, Hall effect sensor, Ultra low power analog pre-amplifier</td>
</tr>
</tbody>
</table>

The state of the real system $S_{real}$ transits from a given state at time $t$ to the state at time $t + \Delta t$ as,

$$S_{t+\Delta t,real} = p(S_{t,real})$$

where $p$ indicates the internal process. $S_t$ comprises arrival rate, processing rate, length of queue, number of arrival, and number of departure.

$$S_t = \{arrRate_t, procRate_t, queueLen_t, numArr_t, numDep_t\}$$

The internal timer operates every time interval $\Delta t$ to record $S_{t,real}$, the state of the system in any given time. The entire data generated in real-time is archived for the validation in the later step. Given that ESP32 supports a timer interruption, the state is stored by using the following code.

```python
## timer to call the function "stateSave" every time interval
stateTimer = Timer(0)
stateTimer.init(period=timeIntv*1000, mode=Timer.PERIODIC, callback=saveState)

## function which stores the state
def saveState(timer):
```
vals = arrRate, procRate, numArr - lastNumArr, numDep - lastNumDep, len(queue), numBalk

lastNumArr = numArr
lastNumDep = numDep
d = dict(zip(stateDict['keys'], vals))

stateDict.update(d)

3.3.2. Measurement Model

The measurement model partially observes the state of the queuing system with errors. Specifically, it can only receive number of arrival and number of departure from the device through serial communication. It happens in a time interval basis, and then, the error is added by sampling from a probability distribution. The equation of the measurement model can be written as,

$$\text{numArr}_{t, \text{measure}} = \text{numArr}_{t, \text{true}} + \text{error}_{t, \text{arrival}} \quad (3.1)$$

$$\text{numDep}_{t, \text{measure}} = \text{numDep}_{t, \text{true}} + \text{error}_{t, \text{departure}} \quad (3.2)$$

where $\text{numArr}_{t, \text{true}}$ and $\text{numDep}_{t, \text{true}}$ are the data generated in the real process and $\text{error}_{t}$ is the measurement errors. For example, suppose that the time interval is set as five seconds and the number of arrivals and departures within the time interval are 100 and 120, respectively. If the sampled errors are -5 and +6, the measurements become 95 and 126. Note that these two measurement errors can be dependent in reality, however, it is assumed that they are independent to each other. Due to the assumption, the joint probability can be readily obtained by calculating the product of two probabilities. Otherwise, additional assumptions would have been required, when calculating the weight of each particle. The following code demonstrates how the measurement model is implemented.

def measure(self):
    np.random.seed(self.gloEnv.seed)

    # sample the counting errors of numArr and numDep
    # assumed to be i.i.d.
    countError = random.choice([-1, 1]) * self.errorPDF.rvs(size=len(self.gloEnv.obsList))

    # add the errors to the ground truth value
    measure = countError.astype(int) + np.array([v[self.timeNow] for k, v in self.true.traceDict.items() if k in self.gloEnv.obsList])
The time interval-based implementation for the measurement model is merely a choice of the thesis, because the simulation software used in the research runs based on time duration instead of an event.

### 3.3.3. Simulation Model

The simulation model is developed in Python by using Salabim, a discrete-event simulation library to mimic the real queuing process. Each simulation replication represents a queuing model which acts as a particle. It is instantiated from `queModel` class in Python.

```python
from queModel import Model

# instantiate a particle with 3 parameters:
# particle index (pf), random seed and time interval
model = Model(part_seq=pf, random_seed=seed, intv=timeIntv, trace=False)
```

The state of the simulation model, $S_{t,\text{sim}}$ also consists of five variables:

$$S_{t,\text{sim}} = \{\text{arrRate}_{t,\text{sim}}, \text{procRate}_{t,\text{sim}}, \text{numArr}_{t,\text{sim}}, \text{numDep}_{t,\text{sim}}, \text{queLen}_{t,\text{sim}}\}$$  \hspace{1cm} (3.3)

The process of advancing the time of all the simulation instances is equivalent to sampling particles from the state transition function. The simulation time is repeatedly advanced by $\Delta t$, every moment when the measurement becomes available. Therefore, the state transition of each particle $i$ can be written by the simulation process $sim$ as,

$$S^i_{t+\Delta t,\text{sim}} = sim(S^i_{t,\text{sim}})$$  \hspace{1cm} (3.4)

where $i = 1, 2, 3...N$. The simulation outcome of each particle, specifically, $\text{numArr}_{t,\text{sim}}$ and $\text{numDep}_{t,\text{sim}}$ are evaluated to calculate the weight in the data assimilation algorithm.

### 3.3.4. Data Assimilation

For the initialization step in the data assimilation, it is assumed that there is no particular prior knowledge of the system state. N set of input variables comprised of arrival rate and processing rate are sampled by using uniform distribution, and the weight is evenly assigned.

$$\{\text{arrRate}^i_{0,\text{sim}}, \text{procRate}^i_{0,\text{sim}}\} \sim \text{Uniform}(lb, ub)$$

$$w^i_0 = 1/N \hspace{1cm} (i = 1, 2, 3...N)$$

This process is implemented in the code as follows.
```python
def sampleLHS(self):
    sampledParams = lhs(2, samples=self.n_pf)
    lb = np.array([0, 0])
    width = np.array([self.gloEnv.maxRate, self.gloEnv.maxRate])
    for i in range(2):
        sampledParams[:, i] = uniform(loc=lb[i],
                                      scale=width[i]).ppf(sampledParams[:, i])
    return sampledParams

def resetWeight(self):
    weights = np.ones(self.n_pf, dtype=float)
    weights /= self.n_pf
    return weights
```

Then, the simulation time of each particle advances by $\Delta t$ for the prediction step to construct the predictive distribution.

$$S_{0+\Delta t, sim}^i = sim(S_{0, sim}^i)$$

(3.5)

Each simulation outcome $S_{0+\Delta t, sim}^i$ is interpreted as the predictive distribution $p(x_k|x_{k-1}^i)$, the state of the system at $0 + \Delta t$ given the previous state at time 0 as follows.

$$p(S_{0+\Delta t, sim}^i|\{arrRate_{0, sim}^i, procRate_{0, sim}^i, queLen_{0, sim}^i\})$$

The code for running this process is partially given in below.

```python
def update(self):
    for itr in range(self.iter):
        self.timeNow += self.timeIntv
        self.runParticles()
```

At the same time, the measurement becomes available, and accordingly 3.3.2 calculates the weight of each particle by comparing the measurement to the simulation outcome.

$$w_{0+\Delta t}^i = p(measurement_{0+\Delta t}^i|prediction_{0+\Delta t}^i) \times w_0^i$$

where

$$measurement_{0+\Delta t}^i = \{numArr_{0+\Delta t, measure}^i, numDep_{0+\Delta t, measure}^i\}$$

$$prediction_{0+\Delta t}^i = \{numArr_{0+\Delta t, sim}^i, numDep_{0+\Delta t, sim}^i\}$$
### 3.4. Sensitivity Analysis

Given the research objective, several parameters of the data assimilation are varied to identify their impacts and the estimation accuracy is measured.

#### 3.4.1. Time Interval

This parameter is directly related to the research gap 1, how the estimation accuracy is affected by the time interval. In every experiment, the real-time data is assimilated by using different time interval from 5.0 seconds to 0.5 seconds (decreased by 0.5 seconds).
Algorithm 5: Particle filter-based Data Assimilation for Queuing Model

// Initialize
1 for \( i = 1 \) to \( N \) do
2 Draw \( i \)-th sample from uniform distribution:
3 \( \{\text{arrRate}^i_{0,\text{sim}}, \text{procRate}^i_{0,\text{sim}}\} \sim \text{Uniform}(lb, ub) \)
4 Set each weight evenly: \( w^i_0 \leftarrow 1/N \)
4 end

// Estimate
5 for \( t = 0 \) to \( k \) do
   /* Prediction Step */
6 for \( i = 1 \) to \( N \) do
7 Run the simulation and advance the time by \( \Delta t \):
8 \( S^i_{t+\Delta t,\text{sim}} \leftarrow \text{sim}(S^i_{t,\text{sim}}) \)
9 end
   /* Update Step */
10 for \( i = 1 \) to \( N \) do
11 Update each weight: \( w^{i+\Delta t}_t \leftarrow p(\text{Measurement}_{t+\Delta t}|\text{Prediction}^i_{t+\Delta t}) \times w^i_t \)
12 Normalize the weight by dividing over the sum: \( w^i_t \leftarrow \frac{w^i_t}{\sum_{i=1}^{N} w^i_t} \)
13 end
   /* \( N \) simulation instances at time \( t+\Delta t \), \( (x^i_{t+\Delta t}, w^i_{t+\Delta t}) \) are obtained */
   /* Resampling */
14 Perform resampling given by Algorithm 2
14 end

3.4.2. Number of Particles
The number of particles is the number of simulation replications used in every time interval. It contributes to the total number of simulation runs combined with the time interval. For example, if the number of particles and the time interval are chosen to be 400 and 2 seconds, the total number of simulation runs during the one experiment is 10,000. (400 replications \( \times \) 50 seconds \( \div \) 2 seconds = 10,000) The number of particles is varied from 10 to 100 increased by 10, and thereafter from 100 to 2,000 increased by 100.

3.4.3. Measurement Error
As described in Chapter 1 the thesis makes distinction between two types of measurement error. One is the error occurs in the measurement process, and the other is the error used to establish the measurement model. The experiment makes all possible combinations of these error types to investigate how the different prior knowledge of error may affect the estimation accuracy. The next paragraph is dedicated to explain how the thesis defines, classifies and quantifies the errors.
6 levels of errors
There are 6 different levels of the measurement error and each level matches to the corresponding nominal value (Zero, Low, Medium, High, Higher and Highest). The different level is characterized by different standard deviation $\sigma$ of Gaussian distribution whose mean value equals to 0. Table 3.3 demonstrates how each nominal value corresponds to a specific $\sigma$.

<table>
<thead>
<tr>
<th>Time Interval (s)</th>
<th>Zero</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Higher</th>
<th>Highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 3.3: Standard Deviation $\sigma$ of the measurement error

Assumptions for Practical Implementation
Suppose the time interval and the measurement error are 3 seconds and Low, respectively. Additionally, the ground truth value of the number of arrivals was 54 within any given time interval. Then, the measurement error is sampled from $N(0, 3^2)$ and added to 54. If the sampled value was -4.23, it becomes $54 - 4.23 = 49.77$. As the number of arrivals and departures only can integer, the rounded value 50 is recorded as a measured value. Here, the assumption was made that modellers do have a knowledge about the system to avoid the unrealistic outcome. In the update step of the data assimilation, however, the raw value 49.77 can be directly used to calculate the weight of the particle, since the measurement model used to evaluate the particle is also Gaussian. The likelihood will be high if the simulation outcome is close to 49.77, and therefore, a high weight will be assigned to the particle.

The simulation model is also designed to prevent the unrealistic outcomes, for example, the negative value of the number of arrivals. It only takes non-negative value when it runs. It is readily implemented by using clip function of numpy, the python module for linear algebra.

```python
def measure(self):
    np.random.seed(self.gloEnv.seed)

    # sample the counting errors of numArr and numDep
    # assumed to be i.i.d.
    countError = random.choice([-1, 1]) *
                  self.errorPDF.rvs(size=len(self.gloEnv.obsList))
```

Besides, it is assumed that the error is proportional to the time interval. This assumption was made since the number of arrivals and departures proportionally increase as the time interval of the measurement becomes larger. If the degree of the error is same for all the different time intervals, the same standard deviation (for example, \( \sigma = 3 \)) will be applied no matter how many entities arrive and depart.

**Different setting between the real error and the perceived error**

The two rows at the bottom of Table 3.3 indicates that the measurement error in reality is set to be from Zero to High whereas the perceived measurement error from Low to Highest. It is done so for a couple of reasons. First, if the perceived error is set to be Zero, most of particles will be discarded when the weight is calculated. In other words, the measurement model does not allow any small difference between the simulation outcome of particles and the measurement. It can be an issue as none of the particles may produce precisely consistent to the measurement over the experiment. The issue can be pronouncing especially when the number of particles is small. Secondly, the perceived error should have a wider range so that the impact of either underestimating or overestimating can be examined. The objective of differentiating the two types of the errors is to investigate the impact of limited knowledge about the error, but not the different perception itself.

**3.4.4. Metric of the Estimation Accuracy**

The accuracy of the estimation is the key performance index of DA. In this research, Distance correlation [26] between two time series data is chosen to measure the similarity of the true trajectory and the estimated one. It is defined by dividing the distance covariance by the product of the distance standard deviations [4], namely,

\[
dCor(X, Y) = \frac{dCov(X, Y)}{\sqrt{dVar(X)dVar(Y)}}
\]

(3.6)

Among multiple choices of measuring the similarity between time series data, distance correlation is chosen as it is a unit-free and relative measure of similarity. As the subject of the estimation is not a single parameter’s vector, but multiple parameters - number of arrivals,
number of departures, arrival rate, processing rate, and length of queue. Due to the different unit of each parameter, other standard methods such as Euclidean distance requires additional normalizing process and thus, additional assumption. Distance correlation, however, results in 0 for true independence (i.e., complete failure of estimation) and 1 for true dependence (i.e., complete success of estimation) as it only measures the correlation between two vectors. Accordingly, the overall estimation accuracy of multiple parameters can also be readily measured by calculating the average, for instance. Moreover, given the trajectory of those variables is very likely to be non-linear, distance correlation is more suitable than the linear correlation. If the trajectory shows a highly non-linear behavior, a linear correlation may indicate a very low value no matter how the trajectory is well-estimated.
4.1. Entire Range of the Estimation Accuracy

A set of 4,800 experiments was conducted by varying the parameters of the data assimilation and the estimation accuracy was measured. The range of the estimation accuracy was from the highest value 0.9544 to the lowest one 0.0170 as shown in Figure 4.1.

Figure 4.1: Distribution of Distance Correlation of 4,800 experiments
4.2. High Estimation Accuracy

The best result out of 4,800 experiments is shown below. In Figure 4.2, the best estimates of the governing parameters, arrival rate and processing rate were made, while their true values showed jump-like behavior over time.

Accordingly, the output of the queuing model, length of queue was also estimated with a high accuracy showed in Figure 4.3.

The experiment of the best result consumed a considerable amount of computational resources. It used 2,000 particles and assimilated the measured data by every one second. Besides the measurement error in reality was Low, and the perception of the error was also correct. As a result, it exhibited the high estimation accuracy.

In the meantime, such a high accuracy was also achieved in other experiments which used a smaller number of particles. For example, an experiment used only 500 particles combined with a smaller time interval. As shown in Figure 4.4, the estimation quality of both input parameters is comparable to the best result. (distance correlation: 0.9423) Consequently, the true trajectory of length of queue was estimated with high quality, as shown in Figure 4.5, although it has a certain amount of lag observed.
4.2. High Estimation Accuracy

Figure 4.4: Estimation of arrival rate and processing rate with smaller number of particles

Figure 4.5: Estimation of length of queue with smaller number of particles

Additionally, there is one thing to note in this experiment. The assumption of the measurement error to be medium was incorrect as the real error was low. However, the quality of the estimation was sufficiently high and even higher than the one with the correct assumption. (See Figure 4.6 and 4.7.)

Figure 4.6: Estimation of arrival rate and processing rate with correct assumption

The distance correlation of the experiment with the correct assumption was 0.8411, which is lower than the wrong assumption. This relation was still the same, after changing the seed number of the entire experiment. More discussion in depth about the importance of the error assumption will be presented in 4.4.
4.3. Low Estimation Accuracy

The distance correlation of the worst case was 0.0170. 100 particles were used with 5 seconds of the time interval while the measurement error was high. It failed to estimate the true trajectory of arrival rate and processing rate as shown in Figure 4.8.

Consequently, the estimation of length of queue exhibited a huge deviation from the ground truth value. (See Figure 4.9.)
Most of the experiments with the low estimation quality, in general, had a contrasting setting of the parameters (such as small number of particles, large time interval and high measurement error), compared to the one with the high estimation accuracy. However, the contribution of each parameter was highly diverse. For example, increasing the number of particles was not enough to guarantee a accurate estimation if the time interval was large. It can be explained by the second-worst result shown in Figure 4.10 and 4.11.

![Figure 4.10: 2nd worst estimation of arrival rate and processing rate](image)

![Figure 4.11: 2nd worst estimation of length of queue](image)

Although it used 1,500 particles and yet the time interval was 3 seconds under the high measurement error. Consequently, the estimated trajectory of all variables largely differed from the ground truth value, and the distance correlation was only 0.0509. In addition, the perception of the measurement error (in this case *Low*) highly deviated from the real error (in this case *High*). The different impact from the parameters on the estimation accuracy is discussed in the next section.

4.4. Sensitivity Analysis

In this section, the sensitivity of the estimation accuracy to different parameters - the time interval, the number of particles, the measurement errors in reality and the perception of the
measurement errors - is analyzed.

### 4.4.1. Feature Importance

The feature importance of each parameter is presented in Figure 4.12 to provide the impact of each parameter on the estimation accuracy. The most influential factor is the number of particles used \((n_{pf})\) followed by the second one, the size of update time interval \((timeIntv)\). On the other hand, the influence of other factors is relatively small.

![Feature Importance](image)

Figure 4.12: Impact of different settings on estimation accuracy in relative scale

### 4.4.2. Effect of the Time Interval

The time interval has a substantial impact on the estimation accuracy, as shown in Figure 4.13.

![Time Interval versus Estimation Accuracy](image)

As the time interval becomes smaller, the estimation accuracy increases. Furthermore, the
smaller time interval does not only contribute to the estimation accuracy, but also to the narrow confidence interval. Figure 4.14 and Table 4.1 shows the result.

![Figure 4.14: 95% Confidence Interval of the different time interval](image)

<table>
<thead>
<tr>
<th>Time Interval (s)</th>
<th>Mean</th>
<th>Median</th>
<th>25% quantile</th>
<th>75% quantile</th>
<th>95% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.5369</td>
<td>0.5943</td>
<td>0.4557</td>
<td>0.6701</td>
<td>0.7325</td>
</tr>
<tr>
<td>4.5</td>
<td>0.5560</td>
<td>0.6154</td>
<td>0.4871</td>
<td>0.6845</td>
<td>0.7380</td>
</tr>
<tr>
<td>4</td>
<td>0.5931</td>
<td>0.6467</td>
<td>0.5336</td>
<td>0.7036</td>
<td>0.6638</td>
</tr>
<tr>
<td>3.5</td>
<td>0.6065</td>
<td>0.6565</td>
<td>0.5548</td>
<td>0.7257</td>
<td>0.7956</td>
</tr>
<tr>
<td>3</td>
<td>0.6016</td>
<td>0.6553</td>
<td>0.5533</td>
<td>0.7197</td>
<td>0.7529</td>
</tr>
<tr>
<td>2.5</td>
<td>0.6453</td>
<td>0.6880</td>
<td>0.5899</td>
<td>0.7604</td>
<td>0.7301</td>
</tr>
<tr>
<td>2</td>
<td>0.6932</td>
<td>0.7536</td>
<td>0.6347</td>
<td>0.8100</td>
<td>0.6676</td>
</tr>
<tr>
<td>1.5</td>
<td>0.7818</td>
<td>0.8236</td>
<td>0.7565</td>
<td>0.8692</td>
<td>0.5444</td>
</tr>
<tr>
<td>1</td>
<td>0.8271</td>
<td>0.8883</td>
<td>0.8353</td>
<td>0.9131</td>
<td>0.5541</td>
</tr>
</tbody>
</table>

Table 4.1: Effect of the time interval on the estimation accuracy

### 4.4.3. Effect of the Number of Particles

The number of particles also determines the estimation accuracy in a large extent. The estimation accuracy increases rapidly as more particles \( n_{pf} \) are used. As similar to the result of the time interval, the mean value of the estimation accuracy becomes larger and also the confidence interval becomes smaller. The result is shown in Figure 4.13 and Table 4.2 describes the summary of the result.
Table 4.2: Effect of the number of particles on the estimation accuracy (partially shown)

<table>
<thead>
<tr>
<th>Number of Particles</th>
<th>mean</th>
<th>median</th>
<th>25% quantile</th>
<th>75% quantile</th>
<th>95% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.6221</td>
<td>0.6527</td>
<td>0.5332</td>
<td>0.7605</td>
<td>0.7800</td>
</tr>
<tr>
<td>500</td>
<td>0.7206</td>
<td>0.7300</td>
<td>0.6407</td>
<td>0.8303</td>
<td>0.5566</td>
</tr>
<tr>
<td>1000</td>
<td>0.7493</td>
<td>0.7477</td>
<td>0.6838</td>
<td>0.8540</td>
<td>0.4932</td>
</tr>
<tr>
<td>1500</td>
<td>0.7507</td>
<td>0.7583</td>
<td>0.6689</td>
<td>0.8535</td>
<td>0.4410</td>
</tr>
<tr>
<td>2000</td>
<td>0.7560</td>
<td>0.7546</td>
<td>0.6767</td>
<td>0.8600</td>
<td>0.4301</td>
</tr>
</tbody>
</table>

However, when the number of particles exceeds 100, the trend of increase becomes slower and stagnated. In order to investigate the moment when the impact of the number of particles becomes insignificant, Tuckey’s test (CI = 95%) is performed. The statistical significance of each difference is partially displayed in Table 4.3 and 4.4 for \( n_{pf} = 300 \) and 400.

Table 4.3: Partial result of Tukey’s test for \( n_{pf} = 300 \)

<table>
<thead>
<tr>
<th>Group1</th>
<th>Group2</th>
<th>Mean difference</th>
<th>Lower</th>
<th>Upper</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>500</td>
<td>0.0453</td>
<td>-0.0028</td>
<td>0.0933</td>
<td>False</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>0.0789</td>
<td>0.0309</td>
<td>0.1270</td>
<td>True</td>
</tr>
<tr>
<td>300</td>
<td>2000</td>
<td>0.0811</td>
<td>0.0331</td>
<td>0.1292</td>
<td>True</td>
</tr>
</tbody>
</table>

The result shows that increasing the number of particles is not as effective as the beginning stage after it exceeds 400.
Table 4.4: Partial result of Tukey’s test for n_pf = 400

<table>
<thead>
<tr>
<th>Group1</th>
<th>Group2</th>
<th>Mean difference</th>
<th>Lower</th>
<th>Upper</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>500</td>
<td>0.0086</td>
<td>-0.0395</td>
<td>0.0567</td>
<td>False</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>0.0423</td>
<td>-0.0058</td>
<td>0.0904</td>
<td>False</td>
</tr>
<tr>
<td>400</td>
<td>2000</td>
<td>0.0445</td>
<td>-0.0036</td>
<td>0.0926</td>
<td>False</td>
</tr>
</tbody>
</table>

4.4. Sensitivity Analysis

**Relation between the Time Interval and the Number of Particles**

The relation between the time interval, the number of particles and the estimation accuracy is displayed in Figure 4.16. Specifically, the x-axis represents the entire number of simulation replications executed over the one experiment (50 seconds). For example, if the time interval and the number of particles are chosen to be 2 seconds and 1,000 particles, the entire number of simulation replications are 25,000\(^1\). As a large number of particles (large marker) and a small size of update intervals (blue marker) are used, more simulation are executed, and thereby, the estimation accuracy approaches to 1.0.

\[\text{number of simulation run (log scale)} = \frac{2 \text{ seconds}}{2 \text{ seconds / updates}} \times 50 \text{ seconds} = 25 \text{ updates}\]

\[25 \text{ updates} \times 1,000 \text{ runs / updates} = 25,000 \text{ runs}\]

![Figure 4.16: Trade-off between the time interval and number of particles](image)

(a) Real scale (b) Log scale

Figure 4.16: Trade-off between the time interval and number of particles

The estimation accuracy increases as more replications are executed. Nevertheless, it is seen that the increasing trend is restricted by the time interval. There is hardly a red marker located close to the top. In other words, if a proper time interval is not selected, such a large number of particles can be a waste of computational resources. Note that many large (large number of particles) and reddish (long time interval) markers located below the 0.8 horizontal line.) Meanwhile, if the time interval is appropriately selected, the quality of estimation can be
sufficiently high, even though a small number of particles is used. Such cases are represented in the graph as small (small number of particles) and blue (short time interval) markers located close to the top. To summarize, once the number of particles is chosen to be sufficient (400 in this research - simple queuing process), the rest of computational resource shall be allocated to shorten the time interval (i.e. increase the update frequency) of real-time data to ensure the estimation quality.

4.4.4. Effect of the Measurement Error in Reality
As the error in the measurement process ($errorTypeReal$) increases, the estimation accuracy decreases and its value also largely deviates. The result is shown in Figure 4.17 and Table 4.5.

![Figure 4.17: Measurement Error (real) versus Estimation Accuracy](image)

### Table 4.5: Effect of the measurement error in reality on the estimation accuracy

<table>
<thead>
<tr>
<th>Measurement Error (Real)</th>
<th>Mean</th>
<th>Median</th>
<th>25% quantile</th>
<th>75% quantile</th>
<th>95% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0.6934</td>
<td>0.7168</td>
<td>0.6210</td>
<td>0.8213</td>
<td>0.7553</td>
</tr>
<tr>
<td>Low</td>
<td>0.6890</td>
<td>0.7127</td>
<td>0.6148</td>
<td>0.8170</td>
<td>0.7470</td>
</tr>
<tr>
<td>Med</td>
<td>0.6601</td>
<td>0.6911</td>
<td>0.5757</td>
<td>0.8025</td>
<td>0.8286</td>
</tr>
<tr>
<td>High</td>
<td>0.6238</td>
<td>0.6548</td>
<td>0.5179</td>
<td>0.7840</td>
<td>0.8815</td>
</tr>
</tbody>
</table>

4.4.5. Effect of the Perception of the Measurement Error
The error used for the measurement model represents how the modellers perceive the actual error occurs in the real system. Although assuming that there is a higher error seems to increase the estimation accuracy, its impact is not very clear, as shown in Figure 4.18.

The purpose of setting the different error on the real measurement and the perception is to investigate the impact of limited knowledge about the measurement error. Therefore, the estimation accuracy depending on the difference between the reality and the perception is examined in Figure 4.19. In particular, each number on the x-axis indicates the extent of either
4.4. Sensitivity Analysis

Figure 4.18: Measurement Error (perception) versus Estimation Accuracy

under- or over-estimation of the real error. For example, if the error is assumed to be Low under the real High error. Given the errors used in the research have 6 levels, the value should be -2 as it under-estimates the error by 2 level. On the basis of zero, the ones on the left represent under-estimated results, while those on the right have over-estimated results.

Figure 4.19: Impact of knowledge about error under different measurement error

It is shown that the perfect (i.e. precisely correct) knowledge of measurement error does not necessarily result in a better estimation. Under-estimating the measurement error than the reality may lower the estimation quality, and yet, overestimating often leads a higher estimation accuracy. Thereafter, the estimation accuracy becomes low as the extent of the overestimation increases. See Table 4.6 for the result.


<table>
<thead>
<tr>
<th>Difference of the Error Level</th>
<th>Mean</th>
<th>Median</th>
<th>25% quantile</th>
<th>75% quantile</th>
<th>95% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>0.5894</td>
<td>0.6230</td>
<td>0.4747</td>
<td>0.7585</td>
<td>0.8866</td>
</tr>
<tr>
<td>-1</td>
<td>0.6533</td>
<td>0.6873</td>
<td>0.5474</td>
<td>0.8113</td>
<td>0.7829</td>
</tr>
<tr>
<td>0</td>
<td>0.6744</td>
<td>0.7025</td>
<td>0.5860</td>
<td>0.8207</td>
<td>0.7665</td>
</tr>
<tr>
<td>1</td>
<td>0.6887</td>
<td>0.7207</td>
<td>0.6015</td>
<td>0.8257</td>
<td>0.7762</td>
</tr>
<tr>
<td>2</td>
<td>0.6750</td>
<td>0.7076</td>
<td>0.5970</td>
<td>0.8120</td>
<td>0.8388</td>
</tr>
<tr>
<td>3</td>
<td>0.6674</td>
<td>0.6996</td>
<td>0.5953</td>
<td>0.7967</td>
<td>0.8257</td>
</tr>
<tr>
<td>4</td>
<td>0.6603</td>
<td>0.6802</td>
<td>0.5958</td>
<td>0.7865</td>
<td>0.7896</td>
</tr>
<tr>
<td>5</td>
<td>0.6341</td>
<td>0.6667</td>
<td>0.5687</td>
<td>0.7673</td>
<td>0.8552</td>
</tr>
</tbody>
</table>

Table 4.6: Effect of the Perception of the Measurement Error

Furthermore, Figure 4.20 displays that this relation still holds under the different real errors. Except the scenario with High error, the correct knowledge does not necessarily outperform when the real error is Zero, Low and Med.
5.1. Answering the Research Question

The research question proposed in Chapter 1 is presented to revisit the overarching theme of the thesis.

Research Question

How is the estimation accuracy of the data assimilation affected by the different strategies of using real-time data and assumptions?

The result described in Chapter 4, showed that the estimation accuracy of the data assimilation varies depending on the different settings - the time interval, the number of particles, the measurement error in reality and the perception of the measurement error. The research question will be answered by addressing each research gap based on the result in the following paragraph.
• Gap1: The current literature of the data assimilation in DEVS lacks an empirical examination on how the time interval may affect the estimation accuracy.

The smaller time interval, in other words, the more frequency update of the real-time is effective to increase the estimation accuracy. In the simple queuing system, the mean value of the accuracy increases from 0.5369 to 0.8271 as the time interval decreases from 5 to 1. Not only the accuracy is enhanced, but also the confidence level of the result increases. The confidence interval of the estimation shrinks from 0.7325 to 0.5541. It is shown that the value of the estimation accuracy less fluctuates across the different parameters. However, the time interval directly determines the required computation. Hence, the selection of the time interval should be made along with the consideration of the number of particles.

• Gap2: Existing works have not highlighted the relation between the time interval and the number of particles under the computational constraint.

Increasing the number of particles or shortening the time interval is the choice should be made under the computational constraint. Once the number of particles reaches to a certain number, the rest of the computational resource should be allocated to shorten the time interval to enhance the estimation accuracy. In this research, the mean value of the accuracy increases from 0.6221 to 0.7560 as the number of particles increases from 100 to 2,000. However, the effectiveness of increasing the number of particles drops quickly after \( n_{pf} = 400 \). Therefore, merely increasing the number of particles regardless of other conditions can be a waste of computational resources. Likewise, the time interval cannot guarantee the estimation to be accurate unless the sufficient number of particles is used.

• Gap3: In the previous research, the limited knowledge of the measurement error is not taken into account.

In this research, underestimating the error always produces the worse estimation. Moreover, the correct perception of the measurement error does not always guarantee a better estimation accuracy. On the other hand, if the error is overestimated to some extent, the estimation accuracy of the data assimilation can outperform the one made under the correct assumption. In this research, overestimating the measurement error by one level showed the better result compared to the correct assumption. After that, a further overestimation (more than two levels) degrades the estimation accuracy.

5.2. Discussion

In this section, the result of the limited knowledge of the measurement error will be discussed, given that the is regarded as counter-intuitive in the thesis. The result described in the subsec-
tion 4.4.5 shows that considering an additional error may generate even better estimation than the one with the correct measurement model. In this specific case (simple queuing model), it is recommended to allow a certain margin of the measurement error, instead of finding out the error model closest to its real profile. One experiment result is shown in Figure 5.1 to explain this point in more details.

While the measurement error in the reality is Low for both cases, the upper case (a) has a correct assumption (Low) and the lower case (b) overestimates (Med) the error. The major difference is that how these two cases are responsive to the sudden change of arrival rate at time 15s. In the upper case (a), it is not responsive enough as the case (b). Consequently the trajectory of length of queue hugely diverges from the true value.

The reason of this phenomena can be inferred from the different spread of the particles which depicted as the gray dots. In the case (a), the likelihoods of many particles are close to zero unless the occurred error is sufficiently small under the assumption of Low measurement error. Then, most of the particles formerly located on the wide space are discarded, and only particles with sufficiently small deviation can “survive” over the experiment. Consequently, the change in the system cannot be detected quickly. Note that the spread of gray dots is narrower for the case (a) than the case (b). In the case (b) as many particles are spread on the larger space, it can quickly converge to the true value under the sudden change of the system. It can be concluded that the widely-spread particles show more robust estimation, especially in detecting the sudden change of the system behavior.
5. Conclusion

In practice, the complete profile of the true error can never be known while the measurement error is usually constructed by selecting a probability distribution. Under the limited and uncertain knowledge of the measurement error, the choice of the measurement model can be highly influential to the robustness of the estimation. Therefore, the modellers may adopt a large variety of perception when modelling the measurement error to handle the dynamic change of the system.

5.3. Limitation and Future work

Despite the findings described in above, the research has the following limitations, and thus, the corresponding future works are suggested.

• Since the research incorporated the most straightforward model in DEVS-based simulation, M/M/1 queue, the outcomes of the thesis may be too specific to generalize to the entire range of DEVS-based simulation. Therefore, the next step is scaling up this simple model to a large-scale complex model in order to verify the conclusion further.

• In terms of the change of system state, there is only a single scenario which may be seen as overly well-designed. The change of the increase and the decrease of arrival rate and processing rate can be varied. The future work should examine the research question and test the findings under the diverse scenarios where the system change shows diverse patterns.

• The thesis used a several assumptions which strongly indicates that we know about the internal process. For example, some negative values of the measurements were automatically calibrated when the simulation runs. It is done for being practical, however, this calibration created the additional difference between the real measurement error and the perception\(^1\). Furthermore, the internal process is often not visible to make such an assumption, hence, there should be a better method to handle the unrealistic outcomes.

• The thesis used the real hardware for having the high controllability and also generating the "real" data instead of simulated one. Nevertheless, it still mimics the real system, and therefore, there is hardly a difference between the model and the system. (i.e. perfect model) Since the model on the base of the perfect knowledge is impossible, the next research may apply the error derives from the limited knowledge of the internal process.

• In the research, the intake of real-time data to the simulation is done every fixed time interval due to the simulation package used. However, given that entities arrive one-by-one in the queuing process, the event-based approach is more natural to model the

\(^1\)By doing so, the Gaussian error of the measurement model is no longer symmetry
system. Thus, the extension of the research may use the event-based simulation, which proceeds its execution by the event that occurred.

- Knowing the relation is different from finding out the optimal setting. The research discovered how different settings of the data assimilation might lead to the different outcomes. The next step is to focus on how the optimal setting can be found beyond the simple queuing model. For example, the prospective findings may include the algorithm to optimize the number of particles and the time interval under the different scenarios.
The appendix provides how to replicate the DDDS used in the research. Since it involves the full factorial experiment across the different parameters, the process of executing requires an extensive computations. Specifically, as the number of particles increases and the time interval becomes shorter, the required resource rapidly increases. It is recommended to use a multicore processor at least more than 8. All the code can be found in Github: https://github.com/yuvenious/ddds_queuing. Note that the uploaded code can generate the "true" value of the real process without using a real hardware.

A.1. Requirements

In order to implement the DDDS in the thesis, the following components are required.

- Python (version of ≥3.6 is recommended.)
- Salabim\(^1\) (DEVS simulation library of Python)
- Analytic libraries of Python, including numpy, pandas, matplotlib and seaborn. In order to meet the requirement, an integrated Python package, such as Anaconda\(^2\) is recommended.
- Hardware with a microcontroller unit which is compatible to Arduino.

\(^1\)https://www.salabim.org/
\(^2\)https://www.anaconda.org
One modification was made in Salabim library, specifically, when instantiating the different simulation instances. As each instance runs independently, the simulation environment also has to be instantiated separately. It was critical to run all the simulation runs in parallel. However, the original code has a small bug when implementing. When Queue class is instantiated, Monitor class is used to monitor the time staying in the queue (length_of_stay), while the Monitor is not instantiated with its own environment. Therefore, the modification was made as follows. It allows the Monitor object to have a separated environment in order to run the parallel simulation. (In the Github repository, the modified code is already archived.)

```python
## Original
class Queue(object):
    ...
    def __init__(self, name=None, monitor=True, fill=None, env=None, *args, **kwargs):
        ...
        self.length_of_stay = Monitor("Length of stay in " + self.name(),
                                      monitor=monitor, type="float")

## Modified
class Queue(object):
    ...
    def __init__(self, name=None, monitor=True, fill=None, env=None, *args, **kwargs):
        ...
        self.length_of_stay = Monitor("Length of stay in " + self.name(),
                                      monitor=monitor, type="float", env=self.env)
```

A.2. Implementation

There are three layers in the code, realized by three classes, Model, Execution, and Experiment. The single queuing system can be instantiated by using the Model class defined in queModel.py. Three arguments are used - the index of the particle for distinguish particles, the random seed to produce different outcomes given the same inputs and the time interval. The simulation time of the instance advances through env.run() method.

```python
## instantiate a single particle
from queModel import Model
model = Model(part_seq=0, random_seed=0, timeIntv=1)
model.env.run(duration=5)
```

As mentioned above, the process to mimic the real system is also included in the code. While the data is generated, the internal measurement process is executed. Based on the result, all the simulation instances run to produce the prediction and thereafter, the measured data is combined to make an estimate. This process keeps running over 50 seconds and can be readily implemented by using Execution class written in the code. This class represents a single DDDS with different parameters - the number of particles, the time interval and the
measurement errors. For example, if the number of particles is 200, the time interval is 2.5 seconds, the real measurement error is "Zero" and the perception of the error is "Low", one can run the execution as follows.

```python
## Execution class which represents the entire run of DDDS

class Execution(object):
    def __init__(self, n_pf, seed, timeIntv, errorTypeReal, errorTypePer, gloEnv):
        self.n_pf = n_pf
        self.seed = seed
        self.timeIntv = timeIntv
        self.errorTypeReal = errorTypeReal
        self.errorTypePer = errorTypePer
        ...

execution = Execution(n_pf=200, seed=0, timeIntv=2.5,
                      errorTypeReal="Zero", errorTypePer="Low")
execution.update()
```

Given that the thesis requires a large number of experiments, the Experiment class is used to run multiple Execution instances with the different settings.

```python
## Experiment class to run multiple Executions

class Experiment(object):
    def __init__(self, execDict, n_iter=50, maxRate=20):
        # global variables
        self.varList = ['numArr', 'numDep', 'arrRate', 'procRate',
                         'numBalk', 'queLen']
        ...

        # Define the parameters
        inputVars = ['n_pf', 'seed', 'timeIntv', 'errorTypeReal', 'errorTypePer']

        # Define the range of each parameter
        n_pfList = sorted(list(range(10, 100, 10)), reverse=True)+list(np.arange(100, 2100, 100))
        # n_pfList = list(np.arange(100, 2100, 100))
        timeIntvList = [.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
        errorTypeRealList = ['zero', 'low', 'med', 'high']
        errorTypePerList = ['low', 'med', 'high', 'higher', 'highest']

        # Run experiment
        experiment = Experiment(execDict=execDict)
        experiment.run()
```

The final outcome is stored as the pandas dataframe object. See the partial result shown in Table A.1
## Table A.1: Result of the Experiment (partial)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>timeIntv</th>
<th>n_pf</th>
<th>errorTypeReal</th>
<th>errorTypePer</th>
<th>errorTypeReal</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>100</td>
<td>zero</td>
<td>low</td>
<td>zero</td>
<td>0.2221</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
<td>200</td>
<td>zero</td>
<td>low</td>
<td>zero</td>
<td>0.2796</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>100</td>
<td>zero</td>
<td>low</td>
<td>zero</td>
<td>0.1910</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>100</td>
<td>zero</td>
<td>low</td>
<td>zero</td>
<td>0.1023</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>100</td>
<td>zero</td>
<td>med</td>
<td>zero</td>
<td>0.1500</td>
</tr>
</tbody>
</table>

### A.3. Visualization

The code also provides a large variety of functions for visualizing the result. Once the Experiment is finished, all the Execution instances can be loaded by PlotObject class.

```python
## instantiate Plot Object by using the output
output = execution.output
plot_object = PlotObject(output)
```

Then, several things are possible to examine the trajectory of the real process, the measurement and the estimation. First, the entire trajectory of the estimation can be plotted by using `plot_trace()` method.

```python
## Show the full trace
fig = plot_object.plot_trace(params=["arrRate", "procRate", "queLen"], ncols=3)
```

![Figure A.1: Full trajectory of the variables](image)

Secondly, if one wants to investigate the distribution of the particles at a certain time, `plot_dashboard` can be used. For example, if we are interested in the time = 6, the following code displays the result.

```python
## Show what happen at time = 6
fig = plot_object.plot_dashboard(time=6)
```

The result is shown below.
Finally, the entire process of the estimation can be animated by using `plot_animate`.

```r
# Show the animation. params can be either "input", "output" or "measurement".
plot_object.plot_animate(params="output")
```

The animation can be found in the readme file of Github repository.

---

**Figure A.2:** Distribution of the particles at time = 6


[23] Simo Särkkä. *Bayesian filtering and smoothing*.


