Data Assimilation for Online Model Calibration in Discrete Event Simulation

Xiaolin Hu, Mingxi Yan
Department of Computer Science
Georgia State University
Atlanta, GA, 30303, USA

Corresponding Author: Xiaolin Hu
Corresponding Author Email: xhu@gsu.edu
Corresponding Author Address: Department of Computer Science, Georgia State University, 25 Park Place, Room 726, Atlanta, GA 30303

Abstract
The increasing availability of real-time data collected from dynamic systems brings opportunities for simulation models to be calibrated online for improving the accuracy of simulation-based studies. Systematical methods are needed for assimilating real-time measurement data into simulation models. This paper presents a particle filter-based data assimilation method to support online model calibration in discrete event simulation. A joint state-parameter estimation problem is defined and a particle filter-based data assimilation algorithm is presented. The developed method is applied to a discrete event simulation of a one-way traffic control system. Experiments results demonstrate the effectiveness of the developed method for calibrating simulation models' parameters in real time and for improving data assimilation results.

Keywords:
Data assimilation, Dynamic data driven simulation, Online model calibration, Discrete event simulation, Particle filters

1. Introduction

Simulation models have long been used to study dynamic systems. As real-time data collected from dynamic systems become more and more available, there is growing interest in using real-time data to improve the accuracy of simulation-based studies. A new paradigm of Dynamic Data Driven Simulation (DDDS) is emerging that allows a simulation system to continuously and systematically assimilate real-time data to support real-time prediction and analysis for dynamic systems [1]. Four activities are identified in DDDS, including *dynamic state estimation*, *online model calibration*, *external input modeling & forecasting*, and *simulation-based prediction/analysis*. In particular, the online model calibration activity calibrates a simulation model's parameters based on real-time data to make the model more accurately reflect the real-time characteristics of a system. This calibration is carried out in an online fashion (i.e., runs in parallel with a system in operation) due to the following two reasons. First, it is common for some characteristics of a system to be known only after the system operates in the real field. This means one cannot simply assign some "typical" or "average" values to the corresponding model parameters. Instead, the parameter values need to be estimated in real time based on how the system actually works. Second, complex dynamic systems may dynamically shift their characteristics due to changes in the operating environments. For these systems, the corresponding model parameters are not static and need to be dynamically estimated based on real-time data from the system.

How to carry out online model calibration remains to be an active research topic. In some cases, one may estimate a parameter's value directly from measurement data. For example, the production rate of a manufacturing machine may be a parameter that needs to be calibrated in real time based on how the machine works in the field. The value of this parameter may be derived directly from the measurement data of job processing time (assuming such measurements are available). More generally, due to the existence of data noise and the unobservability of many model parameters (see discussions in [1]), it is undesirable

or infeasible to derive parameter values directly from measurement data. This asks for systematical methods to carry out parameter estimation. An effective approach is to treat the parameters that need to be estimated as part of the state vector and formulate a joint state-parameter estimation problem, and then employ *data assimilation* techniques to estimate the state and parameters at the same time.

Data assimilation is a methodology that combines measurement data with a dynamic model to optimally estimate the evolving state of a system. It was originally developed in the field of meteorology and has gained popularity in many other science and engineering fields, such as geosciences, oceanography, hydrology, and robotics. The work of data assimilation can be classified into variational data assimilation and sequential data assimilation. This paper focuses on sequential data assimilation that assimilates data and corrects the state estimate each time when a new measurement becomes available. Examples of sequential data assimilation include Kalman filter-based data assimilation and particle filter-based data assimilation. The former assumes a continuous state space with Gaussian probability density and that the state transition model and measurement model are linear Gaussian models [2]. These assumptions do not work for discrete event simulations, which typically use discrete state variables and have non-linear non-Gaussian behavior. On the other hand, particle filters [3,4] work with arbitrary probability density functions and do not rely on specific format and characteristics of the underlying models. This makes it a desirable method to support data assimilation for discrete event simulations.

This paper presents a particle filter-based data assimilation method for online model calibration in discrete event simulation. We define the joint state-parameter estimation problem for online model calibration, and describe the particle filter-based data assimilation method. The developed method is applied to a discrete event simulation of a one-way traffic control system. A series of experiments are designed to demonstrate the different aspects of data assimilation for online model calibration, covering data assimilation with and without parameter estimation, calibration of a static parameter, and calibration of a dynamic parameter. The contributions of this paper are two-fold. First, it defines the online model calibration problem in a formal way and presents a general particle filter-based data assimilation method for online model calibration in discrete event simulation. Second, it evaluates the different aspects of online model calibration using a concrete discrete event simulation example. The evaluation and analysis based on this example demonstrate the effectiveness of online model calibration and provide guidelines to apply the developed method to other simulation applications.

The remainder of this paper is organized as follows. Section 2 describes the related work of online model calibration. Section 3 presents the online model calibration problem and the particle filter-based data assimilation method. Section 4 describes the one-way traffic control system application. Section 5 presents the experiment results covering the different aspects of online model calibration. Section 6 discusses several aspects related to this work and concludes this work.

2. Related Work

Within the simulation field, model calibration refers to the process of adjusting the parameters of a simulation model to make it better model a system under study. This can be done in both an offline fashion and online fashion. Offline calibration uses historical data to calibrate a simulation model before the model is used. It is often formulated as a global optimization problem using historical data that may span a long period of time. On the other hand, online calibration uses real-time data from a system to dynamically adjust a simulation model to make it match the real-time characteristics of a system. It typically works in an iterative way, where each iteration uses newly arrived real-time data to continuously update the model parameters.

The majority work on model calibration belongs to offline calibration. Offline calibration is considered part of the model evaluation process, which is usually divided into three activities: verification, calibration, and validation [5]. Several methods of offline calibration were described in [6], including parameter sweeps, hill climbing, simulated annealing, and genetic algorithms. The work of [7] provided a review of validation and calibration methods within the context of health care modeling and simulation. It discussed the different components of model calibration that include calibration parameters, calibration targets, objective functions, and algorithm for optimizing. An incremental mixture approximate bayesian computation method is

developed and applied to the problem of microsimulation model calibration [8]. Another work of [9] developed an agent-based model calibration framework, which is based on approaches from the fields of uncertainty quantification and model optimization. An example of calibrating agent-based models of innovation diffusion is presented in [10], which uses a gradient-based calibration method. Other examples of offline calibration include building energy simulation calibration [11,12,13] and traffic simulation calibration [14,15].

Compared to offline calibration, less work exists for online model calibration. An online calibration algorithm was developed to support real-time calibration of large-scale traffic simulators [16]. The developed algorithm is based on the extended Kalman filter framework. The work of [17] developed a joint state-parameter estimation method using ensemble Kalman filter to support online calibration of a dynamic model for the application of real-time wind farm control. The topic of online model calibration received more attention recently due to the growing interest in digital twin technologies. A machine learning-based method was developed to support online autonomous calibration of digital twin models for nuclear power plants [18]. The work of [19] presented a particle filter-based method for continuous calibration of a digital twin model, and compared its performance with static and sequential Bayesian calibration approaches. Another work of [20] developed a Bayesian calibration method and applied it to online model calibration using real measurement data from a lab-based demonstrator bridge. None of these works focused on online model calibration for discrete event simulations.

The activities of offline calibration and online calibration are not isolated from each other. An in-depth discussion about the difference and relationship between offline calibration and online calibration can be found in [1]. In particular, offline calibration provides a baseline model, from which online calibration is carried out. From a simulation project lifecycle point of view, offline calibration belongs to the modeling phase where the goal is to develop a high-quality model that match with historical observations. After the model is developed, online calibration can be carried out using real-time data collected from a system. The online calibration starts from the model calibrated from offline calibration.

3. Particle Filter-based Data Assimilation for Online Model Calibration

3.1. The Online Model Calibration Problem: Joint State-Parameter Estimation

The goal of online model calibration is to dynamically calibrate a simulation model's parameters based on real-time measurement data to make the model more accurately capture the characteristics of a system in operation. It is essentially an estimation problem to estimate the parameters based on real-time measurement data. As an estimation problem, this is usually formulated in a probabilistic way. Let $y_k := y(t_k)$ be the measurement data y at time t_k; $y_{0:k} := (y(t_0), y(t_1), \dots, y(t_k))$ be the sequence of measurements up to time t_k. Let θ be the parameter vector to be calibrated. We define $\theta_k := \theta(t_k)$ be the parameter vector θ at time t_k. Then the online model calibration problem can be defined as

$$p(\theta_k | y_{0:k}), \tag{1}$$

i.e., computing the probability distribution of θ_k conditioned on the measurements $y_{0:k}$. This is carried out in an iterative way: when new measurement data become available at time t_{k+1}, a new calibration is carried out to update the estimate of the model parameters.

Online model calibration can be carried out using data assimilation methods. Data assimilation provides a systematical way to estimate a system's state from observation data (also called measurement data). In data assimilation, a dynamic system is generally formulated as a dynamic *state-space model*, which is composed of the *state transition model* of Equation (2) and the *measurement model* of Equation (3) as described below:

$$x_k = f_k(x_{k-1}, u_k, \gamma_k) \tag{2}$$

$$y_k = g_k(x_k, \varepsilon_k) \tag{3}$$

where x_{k-1} and x_k are the state vectors of step $k-1$ and step k, respectively; u_k is the external input vector of step k, and y_k is the measurement vector of step k. Due to the probabilistic nature of state estimation, the state transition model (Equation (2)) is also called the *state transition density* and is

expressed by $p(x_k|x_{k-1}, u_k)$; the measurement model (Equation (3)) is also called the *measurement density* and is expressed by $p(y_k|x_k)$.

The function $f_k()$ defines the dynamics of the state transition and the function $g_k()$ defines the mapping from the state to the measurement. The γ_k and ε_k are two independent random vectors modeling the noises (or uncertainties) involved in the state transition and measurement. The former is called the *process noise* and the latter is called the *measurement noise*. At step k, the sequences of external inputs $u_{1:k}$ (i.e., input data) and measurements $y_{1:k}$ (i.e., measurement data) up to this step are assumed to be known. The initial state x_0 is assumed to be known too based on some prior knowledge. Nevertheless, the states $x_{1:k}$ are hidden and cannot be observed directly. They need to be estimated.

The state transition model (2) captures the knowledge about how the dynamic system evolves its state over time. This knowledge is important for data assimilation as it can generate a prediction of the belief of the state at a future time. A simulation model specifies a systems' state transition and thus can serve as a state transition model. The measurement model (3) links state with measurement data so that information from the latter can be utilized to update the (predicted) belief of the state. A measurement model can be viewed as the model of the sensors that collect the measurement data. It describes, at some level of abstraction, how the state x_k causes sensor measurement y_k. These two models each plays unique roles in estimating the dynamically changing state of a system.

To applying data assimilation to online model calibration, a common approach is to formulate it as a joint state-parameter estimation problem. In this approach, the to-be-estimated parameters are included as part of the state vector that needs to be estimated. Let x_k be the n-dimensional state vector and θ_k be the h-dimensional parameter vector that need to be estimated at step k. Typically, the set of parameters that need to estimated is a small subset of all the parameters of a dynamic model. We define an augmented state vector z_k by appending the parameter vector θ_k to the state vector x_k, i.e.,

$$z_k = \begin{pmatrix} x_k \\ \theta_k \end{pmatrix} \text{ or } z_k = (x_{1,k}, x_{2,k}, \cdots, x_{n,k}, \theta_{1,k}, \theta_{2,k}, \cdots, \theta_{h,k})^T, \quad (4)$$

where z_k is a $n + h$ dimensional vector, $x_{i,k}$ $(i = 1, \dots, n)$ is the ith element of the state vector, and $\theta_{j,k}$ $(j = 1, \dots, h)$ is the jth element of the parameter vector.

The dynamics of the state x_k is defined by the state transition model (i.e., the simulation model). To formulate the state-space model for the joint state-parameter estimation, we need a way to define how the parameters θ_k evolve over time. A popular treatment is to add small random perturbations to the parameters in each step of the transition [21]. Typically, the random perturbations are modeled by zero-mean Gaussian distributions with some specified variances for each parament element, i.e.,

$$\theta_k = \theta_{k-1} + \zeta_k, \quad (5)$$

$$\zeta_k \sim N(0, W_k), \quad (6)$$

where ζ_k are the random perturbations modeled as zero-mean Gaussian noises, and W_k is a diagonal covariance matrix for the Gaussian noises that has variance σ_j^2 for the jth parameter. The initial parameter values can be set based on information from the offline model calibration. Adding random perturbations to the parameter values allows generating new parameter values in each step of the data assimilation. This supports robust estimation even if the initial values are not close to the true values. Large variances of the Gaussian noises lead to large changes of the parameter values in each step. The large variances may be necessary if the estimation has not converged or if one expects the parameter values change dynamically in a fast pace. Otherwise, small variances are preferred. For example, when the parameter under estimation is expected to be a static parameter, a small variance will lead to more stable estimation results.

Combining the parameters' dynamic model Equation (5) with the state transition model Equation (2), we have a new state transition model for the augmented state vector z_k:

$$z_k = \tilde{f}_k(z_{k-1}, u_k, \gamma_k, \zeta_k) = \begin{pmatrix} f_k(x_{k-1}, \theta_{k-1}, u_k, \gamma_k) \\ \theta_{k-1} + \zeta_k \end{pmatrix}. \quad (7)$$

Note that the $f_k()$ function in Equation (7) explicitly lists the model parameters θ_{k-1} because the state transition of step k should use the estimated model parameters from step $k - 1$. In joint state-parameter

estimation, the model parameters at different steps need to be differentiated because their values vary over time.

Using the augmented state vector z_k, the measurement model can be rewritten to

$$y_k = \tilde{g}_k(z_k, \varepsilon_k) \stackrel{\text{def}}{=} g_k(x_k, \varepsilon_k), \quad (8)$$

where $\tilde{g}_k()$ maps from the augmented state vector z_k to the measurement vector y_k. The $\tilde{g}_k(z_k, \varepsilon_k)$ is equivalent to $g_k(x_k, \varepsilon_k)$ because y_k is completely defined by the state x_k due to the Markov assumption of the state space formulation (see details in [1]).

Equations (7) and (8) together form the state-space model for the joint state-parameter estimation problem.

3.2. Particle Filter-based Data Assimilation

The state space formulation described in the previous section allows us to carry out data assimilation to estimate the state variables and model parameters at the same time. Due to the discrete state variables and the non-linear non-Gaussian behavior of discrete event simulation models, we develop data assimilation based on particle filters. Specifically, we choose to employ the bootstrap filter algorithm [3,4] to carry out the data assimilation. The bootstrap filter algorithm uses the state transition model (i.e., the state transition density) to evolve particles during the sampling step. For a discrete event simulation application, the state transition model is defined by the discrete event simulation model. Another major advantage of the bootstrap algorithm is that it simplifies the computation of particles' importance weights, where the weight is defined by the likelihood probability that can be computed from the measurement model (i.e., the measurement density).

Particle filters represent the belief distribution of a state vector under estimation using a set of weighted samples, each of which is called a particle. A particle is a concrete instantiation of the state vector. Let $\{\langle x_k^{(i)}, w_k^{(i)}\rangle \mid i = 1, \ldots, N\}$ be the set of weighted particles, where N is the size of the particle set, $x_k^{(i)}$is the ith particle, and $w_k^{(i)}$is the non-negative importance weight of the ith particle. The importance weights from all the particles sum to one. The bootstrap filter implements a *sequential importance sampling with resampling (SISR)* procedure. It uses the state transition density $p(x_k|x_{k-1}, u_k)$ as the proposal distribution in importance sampling, and invokes resampling in every step. The bootstrap filter algorithm is shown below, where k denotes the time step and y_k^{md} denotes the measurement data at step k.

The Bootstrap Filter Algorithm

1. Initialization, $k = 0$:
   - for $i = 1, \ldots, N$, sample $x_0^{(i)} \sim p(x_0)$.
   - Advance the time step by setting $k = 1$.
2. Importance sampling step (input: $\{x_{k-1}^{(i)}\}_{i=1}^N$, u_k, y_k^{md})
   - for $i = 1, \ldots, N$, sample (i.e., predict) new state $\bar{x}_k^{(i)}$ using the state transition model (the simulation model): $p(x_k|x_{k-1}^{(i)}, u_k)$.
   - for $i = 1, \ldots, N$, compute the importance weight as the likelihood probability: $\bar{w}_k^{(i)} = p\left(y_k = y_k^{md} \middle| \bar{x}_k^{(i)}\right)$.
   - for $i = 1, \ldots, N$, normalize the importance weights: $w_k^{(i)} = \bar{w}_k^{(i)} / \sum_{j=1}^N \bar{w}_k^{(j)}$.
3. Resampling step (input: $\{\langle \bar{x}_k^{(i)}, w_k^{(i)}\rangle\}_{i=1}^N$)
   - Resample N particles $\{x_k^{(i)}\}_{i=1}^N$according to the normalized weights.
4. Advance the time step and start a new cycle:
   - Set $k \leftarrow k + 1$, and go to step 2.

The algorithm starts from an initialization step that generates N initial particles following the belief of the initial state $p(x_0)$. Afterwards, each iteration of the algorithm starts from a particle set $\{x_{k-1}^{(i)}\}_{i=1}^{N}$ representing the posterior belief from the previous iteration. In the importance sampling, each particle uses the state transition model (which is the simulation model) to predict its next state. The importance weight of the particle is then computed as the likelihood probability based on the predicted state and the measurement data y_k^{md} of this step. A higher likelihood probability leads to a larger weight and a lower likelihood probability leads to a smaller weight. The importance weights of all the particles are then normalized. The outcome of the importance sampling is a set of weighted particles $\{\bar{x}_k^{(i)}, w_k^{(i)}\}$, which act as an intermediate approximation for the posterior distribution at step k. In the resampling step, N offspring samples are drawn with probability proportional to the normalized weights. The particles with large weights are selected multiple times while the particles with small weights may be eliminated. The set of resampled particles represent the final posterior distribution of the state at this step. This set of particles also serve as the input for the next iteration of the bootstrap filter algorithm.

Initialization of the bootstrap filter algorithm needs to generate N initial particles following the belief of the initial state. When knowledge about the initial state is available, the initial set of particles can be generated using that knowledge. Otherwise, a common practice is to generate the initial particles randomly covering a wide state space in a uniform way. For the model parameters that need to be calibrated online, their initial values may be sampled around the baseline values resulting from the offline model calibration.

Detailed implementation and explanation of the bootstrap filter algorithm can be found in the literature (e.g.,[1, 2, 22]). In particular, to introduce data assimilation to broader audience in the modeling and simulation community, the work of [22] offers a tutorial on Bayesian sequential data assimilation and particle filtering within the context of discrete event simulation. Other related works include [23, 24, 25, 26], which studied various aspects of particle filter-based data assimilation for discrete event simulations.

4. The One-Way Traffic Control System

To demonstrate online model calibration, we consider the one-way traffic control system described in [27] as an example. The one-way traffic control system is illustrated in Figure 1. During a road construction, the one-way traffic control is managed by two persons deployed to the west and east ends of the road segment. Each person carries a STOP/SLOW hand-held traffic paddle to control the traffic, where the STOP sign means vehicles should stop and wait, and the SLOW sign means vehicles can slowly move ahead to pass the road segment. It is assumed that the two persons coordinate and always use the STOP/SLOW signs in opposite directions: when one uses the STOP sign the other would use the SLOW sign. In the following description, we refer to the STOP sign as the red traffic light and the SLOW sign as the green traffic light, and refer to vehicles' moving directions as *east-moving* (moving towards east) and *west-moving* (moving towards west). During the time when the traffic light is green on a specific direction (east-moving or west-moving), the arrival vehicles moving in the opposite direction are queued. The queues at the west side and east side of the road segment are named as the *west-side queue* and *east-side queue*, respectively.

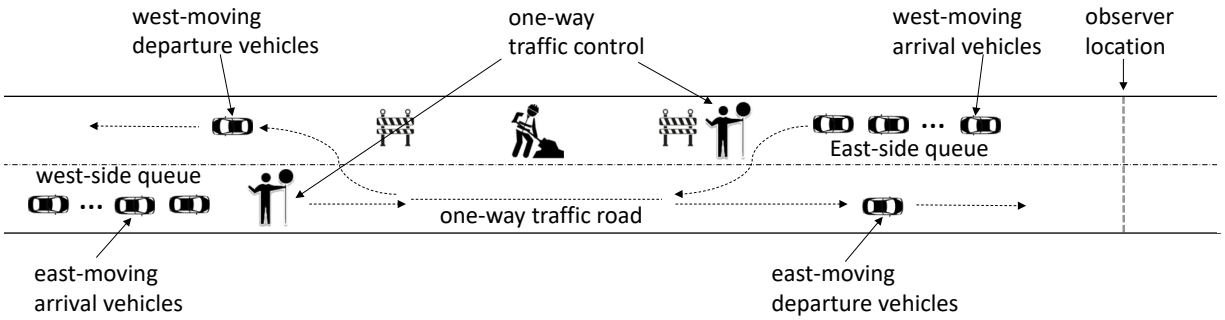


Figure 1: The one-way traffic control system [27]

To ensure construction workers' safety, only one vehicle is allowed to move on the one-way traffic control road at any time. During a green light period, the traffic-control person on the corresponding side would signal a vehicle to move ahead only after the previous vehicle has finished crossing the road segment. The one-way traffic control system switches the traffic lights under the following two conditions:

1. If the elapsed time for the current moving direction has reached a pre-defined threshold (120 seconds in this example) and the opposite moving direction has cars waiting, switch the traffic light. Note that if the opposite moving direction has no car waiting, the traffic light does not switch even after the 120-second threshold. Also note that the traffic light switches only after the road segment is cleared if there is a car already moving on the road.
2. If the current moving direction has no car waiting and the opposite moving direction has cars waiting in the queue, switch the traffic light even if the 120-second threshold is not reached.

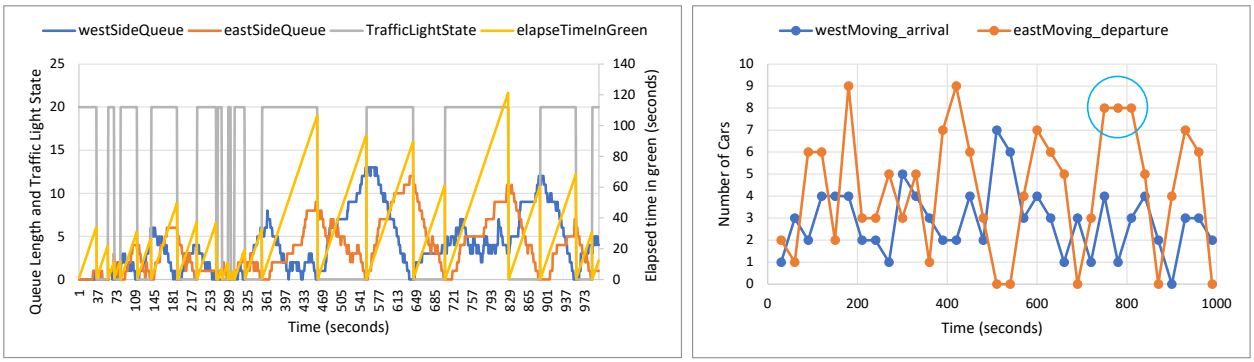
The vehicles at both sides of the road segment arrive randomly and independently, modeled by Poisson distributions. For the east-moving vehicles, the Poisson distribution has an arriving rate $\lambda_{eastMov} = 1/7$ (1 car per 7 seconds in average). For the west-moving vehicles, the arriving rate is $\lambda_{westMov} = 1/10$ (1 car per 10 seconds in average). This means there are more east-moving vehicles than west-moving vehicles. The time it takes for a vehicle to cross the road segment is also a random number, modeled by a truncated normal distribution that has the mean $\mu_{crossT} = 4.0$ (seconds), variance $\sigma^2 = 0.5^2$, and lies within the range of $[\mu_{crossT} - 1, \mu_{crossT} + 1]$.

To collect measurement data from the system, an observer (sensor) is deployed at the location on the east end of the one-way traffic road that is marked as the "observer location" in Figure 1. The observer is able to count the number of vehicles moving crossing its location for both the east-moving departure vehicles (named as *eastMoving_departure*) and west-moving arrival vehicles (named as *westMoving_arrival*). The observer reports data every 30 seconds. It does not record the specific time that a vehicle crosses the observation location – all it reports is the total number of vehicles that have departed and arrived in the past time interval. The data reported by the observer is noisy, with a 10% noise added to the actual number of vehicles crossing the observer location.

We use a DEVS [28]-based model to model the one-way traffic control system. This is a DEVS coupled model that includes three atomic models: *eastMovCarGenr*, *westMovCarGenr*, and *oneWayTrafficRoad*, as described below.

- *eastMovCarGenr*: generates the east-moving traffic arriving at the west side of the road segment. Traffic are generated based on a Poisson distribution with rate $\lambda_{eastMov} = 1/7$.
- *westMovCarGenr*: generates the west-moving traffic arriving at the east side of the road segment. Traffic are generated based on a Poisson distribution with rate $\lambda_{westMov} = 1/10$.
- *oneWayTrafficRoad*: models the one-way traffic road segment, including the traffic control logic as well as the time for vehicles to cross the road segment. This model has two input ports *eastMovArival* and *westMovArrival* that receive vehicles generated from the *eastMovCarGenr* and *westMovCarGenr*, respectively. The vehicles finishing crossing the road segment are sent out through the *eastMovDeparture* and *westMovDeparture* output ports. An important parameter of this atomic model is the average cross time μ_{crossT} as described above.

Figure 2(a) shows the results of a specific simulation scenario that runs for 1000 seconds of simulation time. The values of four state variables *westSideQueue* size, *eastSideQueue* size, *TrafficLightState*, and *elapsedTimeInGreen* are displayed over time. The *trafficLightState* has two possible values: "east-moving green" or "west-moving green". The former is displayed as value 20, and the latter is displayed as value 0. The *elapsedTimeInGreen* is the elapsed time in the "east-moving green" or "west-moving green" state. It is a continuous variable with non-negative values. The *elapsedTimeInGreen* is reset to zero whenever the traffic light switches.



(a) Simulation results (b) Measurement data

Figure 2: Sample simulation scenario and measurement data

One can see that the traffic light switches irregularly, governed by the traffic control rules and influenced by the number of vehicles waiting on the east side and west side queues. When both queues have vehicles, the traffic light switches after *elapsedTimeInGreen* reaches 120 seconds. When the queues on both sides are small, the traffic light may switch frequently because once all the vehicles on one side pass the road segment the traffic light switches immediately to allow vehicles on the other side to use the road. One can also see that when the *TrafficLightState* is "east-moving green" (value 20), the *westSideQueue* decreases because vehicles on the west side can move forward. A similar pattern can be observed for the *eastSideQueue*.

Figure 2(b) shows the measurement data *eastMoving_departure* and *westMoving_arrival* for the simulation run shown in Figure 2(a). Since the measurement data is collected every 30 seconds, 33 measurement data were collected over the 1000 simulation time. One can see that the measurement data for both *eastMoving_departure* and *westMoving_arrival* varied between 0 and 9. The measurement data (indirectly) reflect the state of the one-way traffic system. For example, when the traffic light state is "east-moving green" and there are many vehicles waiting on the west side queue, the *eastMoving_departure* measurement would have relatively large values for several steps in a row (see the circled measurements in Figure 2(b) as an example). Despite the information carried by the measurement data, one cannot directly derive the state values (e.g., the exact number of vehicles in the west-side or east-side queues) from the measurement data.

Data assimilation is useful for this application because an accurate state estimation from real-time measurement data allows one to initialize simulation runs using the estimated real-time state to predict the future behavior of the system. Such predictions are useful for supporting real-time decision making, e.g., to direct traffic away if the predicted waiting time for crossing the road segment is too long. The work of [27] provides a concrete example of using data assimilation to support simulation-based real-time prediction/analysis for this system.

5. Experiment Results

5.1. Identical Twin Experiment

Based on the one-way traffic control system described in Section 4, we carry out a series of experiments to demonstrate different aspects of online model calibration. We use the identical twin experiment design [1] to carry out the experiments. In the identical twin experiment design, a simulation (referred to as the "true system") is first run and the corresponding data are recorded. The measurement data obtained from this simulation are regarded as coming from the "true system" (also called the "real system"), and the state trajectory recorded from this simulation and the parameters used in this simulation are considered the "true" state and "true" parameters, respectively. Using the measurement data collected from the "true system", data assimilation is then carried out and the state/parameter estimates are checked against the true

state/parameters. The first simulation that serves as the "true system" is often based on the same simulation model as the one used in data assimilation. Nevertheless, it has a different model configuration, e.g., different initial state, input trajectory, random number seeds, or model parameters for generating behaviors that are "unknown" to the data assimilation. In this way, it acts as a surrogate for the real system and generates synthetic measurements to be assimilated. To differentiate the model used in the "true system" from the one used in data assimilation, we name the former the *"true system" model* and the latter simply *the simulation model*.

Figure 3 illustrates the identical twin experiment design used in this paper. For the one-way traffic control system, the dynamic behavior is driven by the east-moving and west-moving traffic as well as the time for vehicles to cross the road segment, all of which are influenced by the random number generation in the model. By changing the seeds of random number generation, we can produce different simulation scenarios. Based on this idea, in the identical twin experiment we set up the "true system" model using a specific set of random number seeds that are unknown to the data assimilation. The different random numbers between the true system model and the simulation model lead to different model behaviors between the two. Without knowing the actual behavior of the true system model, the data assimilation estimates the true system state by assimilating measurement data from the true system.

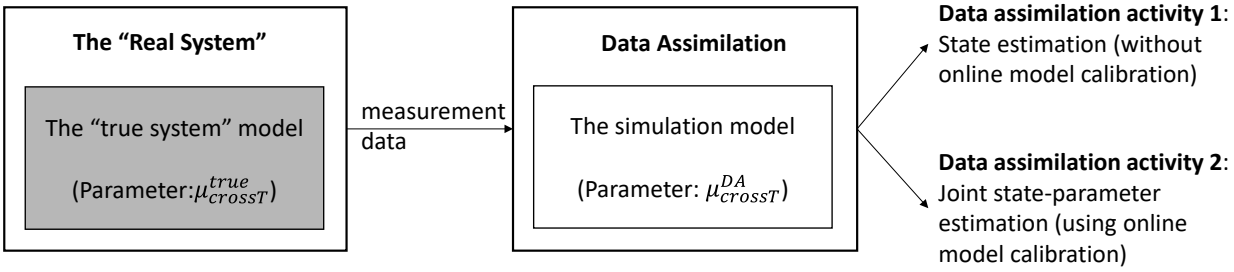


Figure 3: The identical twin experiment design

Besides the different random number seeds, the identical twin experiment design also focuses on an important model parameter: μ_{crossT}, and make the true system model and the simulation model to have different values for this parameter. The goal is to study the impact of this model parameters on the data assimilation results and to evaluate how well the joint state-parameter estimation can estimate this parameter. Recall that the parameter μ_{crossT} specifies the average time for vehicles to cross the road segment (i.e., the mean of the truncated normal distribution from which vehicles' crossing time is sampled). The default value of μ_{crossT} is 4.0. A smaller μ_{crossT} means vehicles pass the road segment faster, and thus leads to less vehicles in the queues on both sides of the road segment. A larger μ_{crossT} means vehicles pass the road segment slower, and thus leads to larger queue sizes on both sides.

In the following description, we use μ_{crossT}^{true} to refer to the parameter of the true system model, and μ_{crossT}^{DA} to refer to the parameter of the simulation model. Specific values of μ_{crossT}^{true} and μ_{crossT}^{DA} vary between experiments and are described in each experiment section.

We consider two types of data assimilation activities (shown on the right side of Figure 3) as described below.

- **Data assimilation activity 1**: state estimation without online model calibration (i.e., without parameter estimation). This activity aims to demonstrate the effectiveness of data assimilation and the impact of imperfect model parameters on data assimilation results. In this activity, the true system model and the simulation model may use different model parameters.
- **Data assimilation activity 2**: joint state-parameter estimation (using online model calibration). In this activity, the μ_{crossT}^{true} used by the true system model is unknown to the simulation model. The goal is to show how well the data assimilation is able to estimate this parameter over time, and how the joint state-parameter estimation can improve state estimation results.

In all the experiments, the measurement data *eastMoving_departure* and *westMoving_arrival* are collected every 30 seconds. The values of the measurement data depend on the specific behavior of the true system model, which is driven by the arrival vehicles on both sides of the road segment (controlled by the random number seeds of the traffic generation models). To save space, we do not show the specific values of the measurement data.

All the experiments consider 200 data assimilation steps (total 6000s because each data assimilation step is 30s). When showing the state estimation results, we focus only on the *westSideQueue* size and *eastSideQueue* size because they have the most impact on vehicles' delay time for crossing the road segment. The particle filtering algorithm in all the experiments uses 4000 particles.

5.2. Data Assimilation with Imperfect Model Parameter

This set of experiments aim to demonstrate the effectiveness of data assimilation for state estimation without using online model calibration. To evaluate the impact of imperfect model parameters on state estimation results, we intentionally make the μ_{crossT}^{true} different from the μ_{crossT}^{DA}. The mismatch of these two parameters, in a way, represent that a simulation model does not model a real system perfectly.

Specifically, the true system model uses different μ_{crossT}^{true} to generate measurement data in different experiment, but the simulation model in all the experiments use the same μ_{crossT}^{DA}. In this way, the measurement data is generated from a true system that is different from the simulation model used in data assimilation. We then study how the different levels of discrepancy between the μ_{crossT}^{true} and μ_{crossT}^{DA} influence the state estimation results. Specific settings of the experiments are described below.

- **True system model**: each experiment uses a different μ_{crossT}^{true} that varies from 3.0 to 5.0 with an 0.25 interval, i.e., $\mu_{crossT}^{true} \in \{3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75, 5.0\}$.
- **The simulation model**: all experiments use the same $\mu_{crossT}^{DA} = 4.0$.

Note that the random number seeds used in the true system model and the simulation model are also different. The specific values of the random number seeds for all the experiments are omitted in this paper.

Figure 4 shows the state estimation results for the *westSideQueue* and *eastSideQueue* from three data assimilation runs whose μ_{crossT}^{true} are 3.5, 4.0, and 4.5, respectively. The state estimates in each data assimilation step is computed as the average from all the particles (after the resampling), marked as *DA_aveWestSideQueue* and *DA_aveEastSideQueue* in the figure. The corresponding "true" states for the two state variables are also displayed in the figure, marked as *real_westSideQueue* and *real_eastSideQueue*.

As can be seen, for the data assimilation run where $\mu_{crossT}^{true} = 3.5$ and $\mu_{crossT}^{DA} = 4.0$ (the top diagram), the true west-side queue and east-side queue oscillate but stay in a relatively low level (less than 10). This is because the μ_{crossT}^{true} is small and vehicles can pass the road segment fast, and thus leading to few vehicles in the queues. For this experiment, the data assimilation accurately estimates the small number of vehicles as well as their oscillations in the two queues, even though the simulation model's μ_{crossT}^{DA} is larger than the true system model's μ_{crossT}^{true}. This is because the measurement data carry information that reflect the frequent switches of the traffic light due to the relatively small number queued vehicles on both sides of the road segment. Combining this information with the knowledge of how the system works (defined by the simulation model), the data assimilation is able to estimate the traffic light states and the queue sizes over time. We note that an accurate estimation does not mean the estimated state perfectly matches the real state. Instead, it means it has relatively small errors.

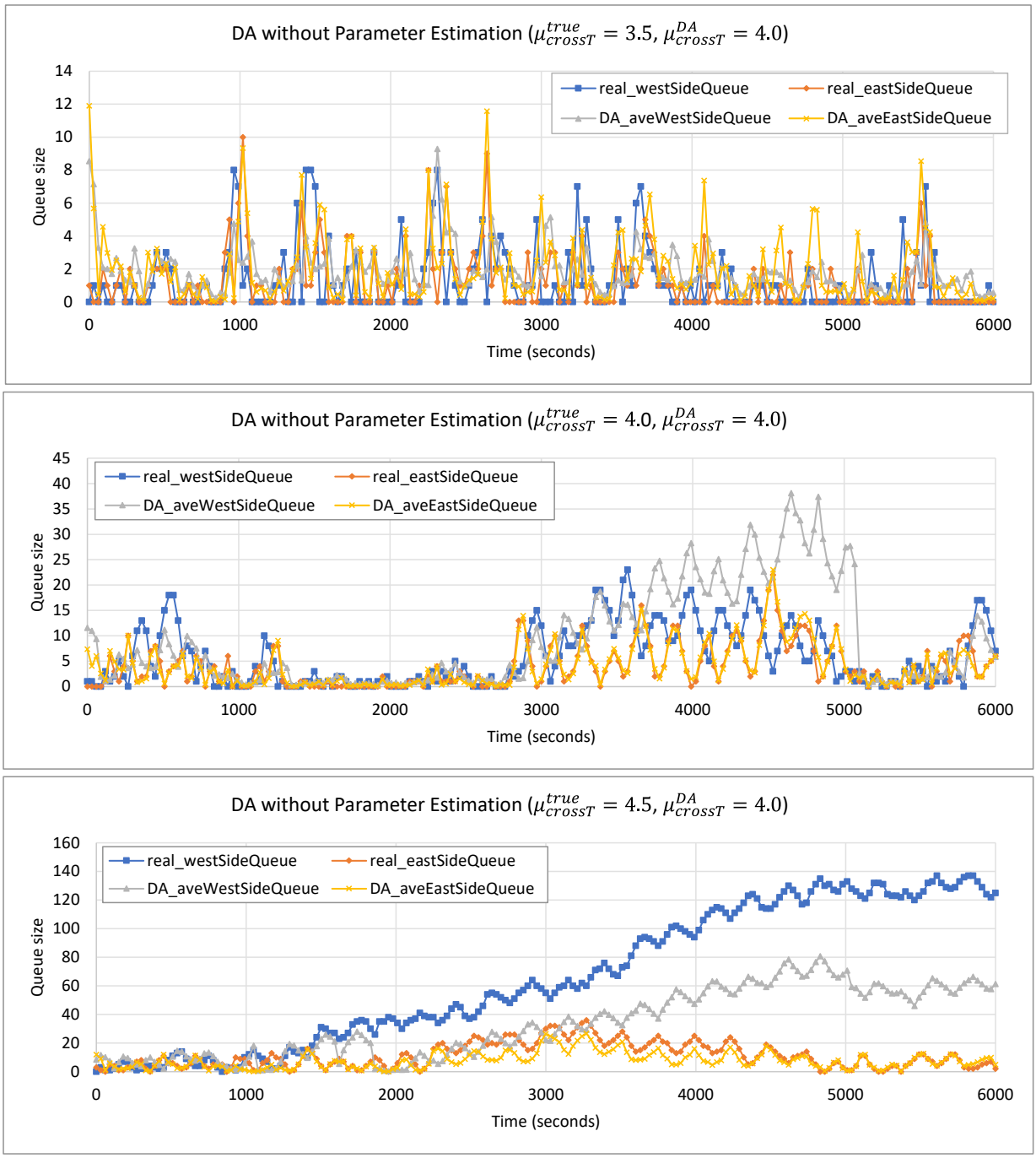


Figure 4: State estimation without using online model calibration: (top) state estimation results when $\mu_{crossT}^{true} = 3.5$ and $\mu_{crossT}^{DA} = 4.0$; (middle) state estimation results when $\mu_{crossT}^{true} = 4.0$ and $\mu_{crossT}^{DA} = 4.0$; (bottom) state estimation results when $\mu_{crossT}^{true} = 4.5$ and $\mu_{crossT}^{DA} = 4.0$.

For the data assimilation run where $\mu_{crossT}^{true} = 4.0$ and $\mu_{crossT}^{DA} = 4.0$ (the middle diagram), the μ_{crossT}^{true} becomes larger compared to the previous experiment. This means it takes longer for vehicles to cross the road segment and thus it is likely for more vehicles to be queued. In particular, during time between 3000s and 5000s, there are significant number of vehicles queued on both sides in the real system. The west-side has a longer queue because vehicles on this side arrive more frequently. For this experiment, the data assimilation is able to estimate the east-side queue accurately almost all the time. This is because the

westMoving_arrival measurement data has direct information about the number of vehicles arrived on the east side. For the west-side queue, between time 3500s and 5100s, the estimated queue size is larger than the real queue size. During this period of time, the measurement data indicate that there are many vehicles queued on the west side; nevertheless, it does not carry information about the actual number of queued vehicles. This results in some errors for the west-side queue estimation. Nevertheless, by the time of 5100s, the real system's west-side queue is cleared and that is reflected in the measurement data. This allows the data assimilation to adjust its estimation of the west-side queue and correctly reset the estimation to close to zero. This example illustrates how data assimilation can combine information from both the measurement data and simulation model to estimate the dynamically evolving system state.

Finally, for the data assimilation run where $\mu_{crossT}^{true} = 4.5$ and $\mu_{crossT}^{DA} = 4.0$ (bottom diagram), it takes even longer time for vehicles in the real system to cross the road segment. This causes many vehicles to be queued on the west side of the road segment. For this experiment, the data assimilation is still able to estimate the east-side queue. However, for the west-side queue, although the estimation indicates the west side queue has many vehicles, the specific estimated value is much smaller than the real value. This is because the $\mu_{crossT}^{DA} = 4.0$ is smaller than the $\mu_{crossT}^{true} = 4.5$ used by the true system model. The smaller μ_{crossT}^{DA} means the data assimilation assumes vehicles pass the road segment faster than they do in the real system, and thus leading to smaller estimation of the queue size. In this experiment, one can see that the data assimilation provides some level of state estimation; however, one of the state variables has significant estimation errors due to the inaccurate model parameter of μ_{crossT}^{DA}.

To quantitatively show the impact of imperfect model parameters on state estimation results, we systematically change the μ_{crossT}^{true} from 3.0 to 5.0 with an increment of 0.25 but keep the μ_{crossT}^{DA} to be a constant of 4.0. For each set of parameters (e.g., $\mu_{crossT}^{true} = 3.0$ and $\mu_{crossT}^{DA} = 4.0$), we carry out 20 independent data assimilation runs and compute the Root Mean Square Error (RMSE) of the data assimilation results. The RMSE is calculated based on the differences between the estimated state and the corresponding true state for the *westSideQueue* and *eastSideQueue* state variables. Since the data assimilation results are represented by a set of particles, we calculate the RMSE using the estimations from all the particles. For each data assimilation run, a RMSE is first calculated by averaging all particles' RMSE in each data assimilation step and then the overall RMSE of the data assimilation run is computed as the average of the RMSEs from all the data assimilation steps (the first 10 steps are excluded from the computation because it takes several steps for the data assimilation to converge). After obtaining the RMSEs from all the 20 runs, we remove the largest and the smallest RMSE values and then compute a final average RMSE using the remaining 18 values.

The blue curve (marked as "*DA without Parameter Estimation*") in Figure 5 shows the computed RMSE for the different sets of parameters. The horizontal axis represents the different μ_{crossT}^{true} values (3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75, 5.0). The μ_{crossT}^{DA} is always 4.0 and thus is not shown in the figure. The vertical axis represents the RMSE value. Note that Figure 5 also shows the results for the "*Joint State-Parameter Estimation*" (the red curve), details of which will be described in the next section.

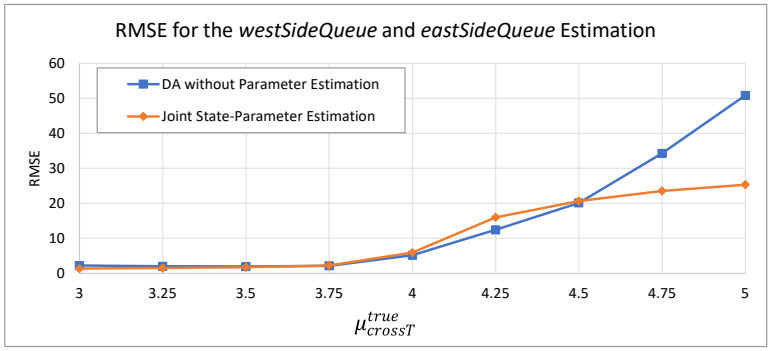


Figure 5: Quantitative results of state estimation with and without parameter estimation. The RMSEs are computed from 20 independent data assimilation runs for each case.

One can see that for the cases where μ_{crossT}^{true} equals to or is less than 4.0, the RMSEs stay at a relatively low level even though the μ_{crossT}^{DA} is still different from the μ_{crossT}^{true}. This is because in these cases the vehicles in the real system can pass the road segment fast and thus there are few queued vehicles on both sides. The small number of queued vehicles result in frequent switches of traffic light, which are reflected by the measurement data from the real system. By assimilating these measurement data, the data assimilation is able to limit the estimation error in a relatively small range, as exemplified by the top and middle diagrams of Figure 4. For the cases where μ_{crossT}^{true} is larger than 4.25, the RMSE increases: the larger the μ_{crossT}^{true}, the higher the RMSE. This is because for larger μ_{crossT}^{true}, the discrepancy between the μ_{crossT}^{true} and the μ_{crossT}^{DA} becomes larger. A larger discrepancy leads to higher state estimation errors, as exemplified by the bottom diagram of Figure 4. For these cases, the impact of the imperfect model parameter of μ_{crossT} is significant.

5.3. Joint State-Parameter Estimation for a Single Static Parameter

In this set of experiments, instead of using a fixed model parameter that may be incorrect, the data assimilation assumes the model parameter is unknown and needs to be calibrated based on real-time measurement data. In other words, the data assimilation carries out join state-parameter estimation. Similar as the previous experiments, we focus on the μ_{crossT} parameter and make the true system model use different μ_{crossT}^{true} to generate measurement data. The μ_{crossT}^{true} considered in this section is a static parameter that does not change over time. The simulation model assumes the μ_{crossT}^{DA} is unknown and estimates it together with the state variables. Specific setting of the experiments is given below.

- **True system model**: each experiment uses a different μ_{crossT}^{true} that varies from 3.0 to 5.0 with an 0.25 interval, i.e., $\mu_{crossT}^{true} \in \{3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75, 5.0\}$.
- **The simulation model**: μ_{crossT}^{DA} is unknown and needs to be estimated.

Figure 6 shows the results of three data assimilation runs where the true system models use exactly the same configurations as the three cases shown in Figure 4, respectively. In other words, the true system models' μ_{crossT}^{true} are 3.5, 4.0, and 4.5, respectively, and the measurement data and the true states are the same as the ones shown in Figure 4. The difference is that the data assimilation runs here use joint state-parameter estimation as opposed to using a fixed parameter $\mu_{crossT}^{DA} = 4.0$. Both the state estimation results (for the *westSideQueue* and *eastSideQueue*) and the parameter estimation results are shown. Similar as before, estimation results in each step is computed as the average from all the particles (after the resampling). The corresponding true states and true parameter are also displayed in the figure.

One can see that the joint state-parameter estimation in all three cases were able to converge to the true parameter values after some time. The estimated parameter values do not stay constant over time but are close to the true parameter values. On the state estimation side, when μ_{crossT}^{true} is 3.5 and 4.0, the state estimation results have similar errors as the ones shown in Figure 4. Nevertheless, for the case when $\mu_{crossT}^{true} = 4.5$ (the bottom diagram), the joint state-parameter estimation leads to much more accurate estimation results compared to the one shown in Figure 4. This is because by correctly estimating the μ_{crossT}^{true} parameter, the simulation model is able to more accurately simulate the dynamics of the system and thus lead to more accurate estimation results. For this specific case, by using an estimated μ_{crossT}^{DA} of 4.5 as opposed to an incorrect value of 4.0, the data assimilation knows that it takes longer time for vehicles to cross the road segment, and thus estimates that the west-side queue has more queued vehicles. This example demonstrates how the online model calibration can significantly improve state estimation results.

To quantitatively show the impact of the join parameter-state estimation on state estimation, we use the same way as described in Section 5.2 to compute the RMSEs of the estimated *westSideQueue* and *eastSideQueue* based on joint state-parameter estimation. The results are shown in Figure 5 (the red curve that is marked as "*Joint State-Parameter Estimation*"). One can see that for the cases when μ_{crossT}^{true} equals to or is less than 4.0, the RMSEs from the joint state-parameter estimation are about the same as the data assimilation without using parameter estimation (the blue curve). This is because in these cases the data

assimilation without parameter estimation has relatively small errors already (as explained in Section 5.2). These results show that the joint state-parameter estimation does not degrade the estimation results for the cases where the previous data acclimation has accurate state estimations already. After μ_{crossT}^{true} becomes larger than 4.5, the joint state-parameter estimation clearly improves the state estimation results by significantly reducing the RMSEs. We expect more improvement will be observed if μ_{crossT}^{true} is larger than 5.0 (not shown in the figure). An interesting result exists for the case of $\mu_{crossT}^{true} = 4.25$, where the joint state-parameter estimation has a slightly larger RMSE than the data assimilation that uses a fixed $\mu_{crossT}^{DA} = 4.0$. This is explainable because the joint state-parameter estimation does not perfectly estimate the μ_{crossT}^{true}. The errors between the estimated parameter and the true parameter may be even larger than using a fixed $\mu_{crossT}^{DA} = 4.0$. This means when the discrepancy between the true system model and the simulation model is relatively small, there may be little benefit of using joint state-parameter estimation.

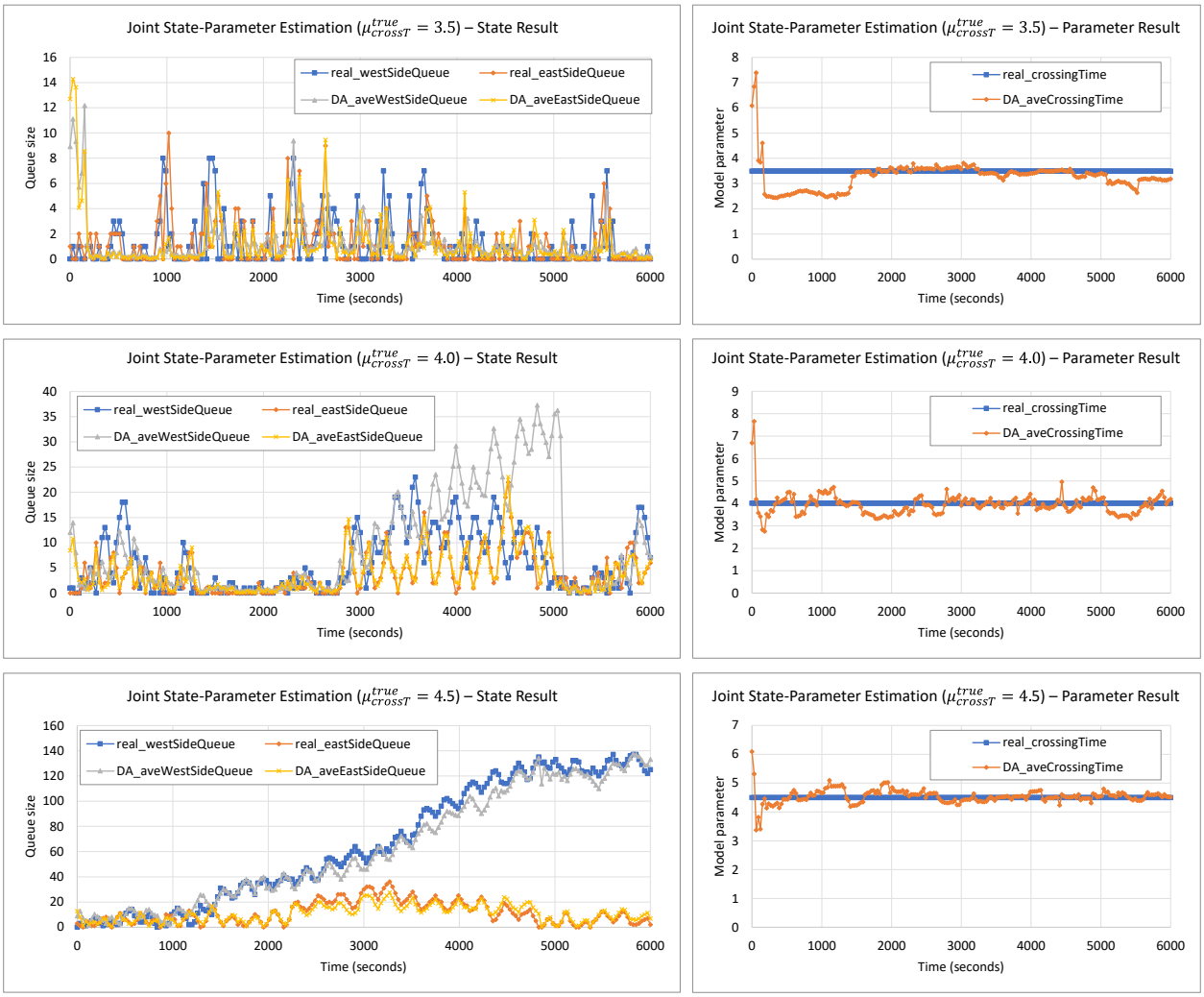


Figure 6: Data assimilation using joint state-parameter estimation: (top) state and parameter estimation results when $\mu_{crossT}^{true} = 3.5$; (middle) state and parameter estimation results when $\mu_{crossT}^{true} = 4.0$; (bottom) state and parameter estimation results when $\mu_{crossT}^{true} = 4.5$.

This set of experiments show that the joint state-parameter estimation is able to estimate a static parameter of the true system model correctly. Furthermore, it is able to improve the state estimation for the cases where the data assimilation without using parameter estimation have large errors.

5.4. Joint State-Parameter Estimation for a Dynamic Parameter

In all the previous experiments, the μ_{crossT}^{true} is a static parameter that does not change over time. This section studies how data assimilation works for real systems that have dynamic parameters, i.e., parameter that dynamically change over time. To set up the experiments, we make the true system model use a dynamic μ_{crossT}^{true} as described below. Similar as before, we carry out two data assimilation activities: 1) activity 1: state estimation without parameter estimation; and 2) activity 2: joint state-parameter estimation. For the activity 1, the μ_{crossT}^{DA} has a constant value of 4.0. For the activity 2, the μ_{crossT}^{DA} is unknown and is estimated together with the state variables. Specific settings of the experiments are described below:

- **True system model**: μ_{crossT}^{true} dynamically change in the following way: it stays at 4.0 from 0 to 1000s, and gradually increases to 6.0 from 1000s and 3000s; it then gradually decreases to 3.0 from 3000s to 4200s, and again gradually increases to 4.0 from 4200s to 5000s, and then stays at 4.0 from 5000s to 6000s. The blue curve in the middle diagram of Figure 7 displays these changes.
- **The simulation model of activity 1** (without parameter estimation): use a constant $\mu_{crossT}^{DA} = 4.0$.
- **The simulation model of activity 2** (joint state-parameter estimation): μ_{crossT}^{DA} is unknown and needs to be estimated.

Figure 7 shows the data assimilation results from both data assimilation activity 1 and activity 2. The top diagram shows the state estimation results of *westSideQueue* and *eastSideQueue* from a data assimilation run without parameter estimation. The middle diagram and bottom diagram show the results from a data assimilation run using joint state-parameter estimation: the middle diagram shows the μ_{crossT}^{true} and the parameter estimation result; the bottom diagram shows the state estimation results.

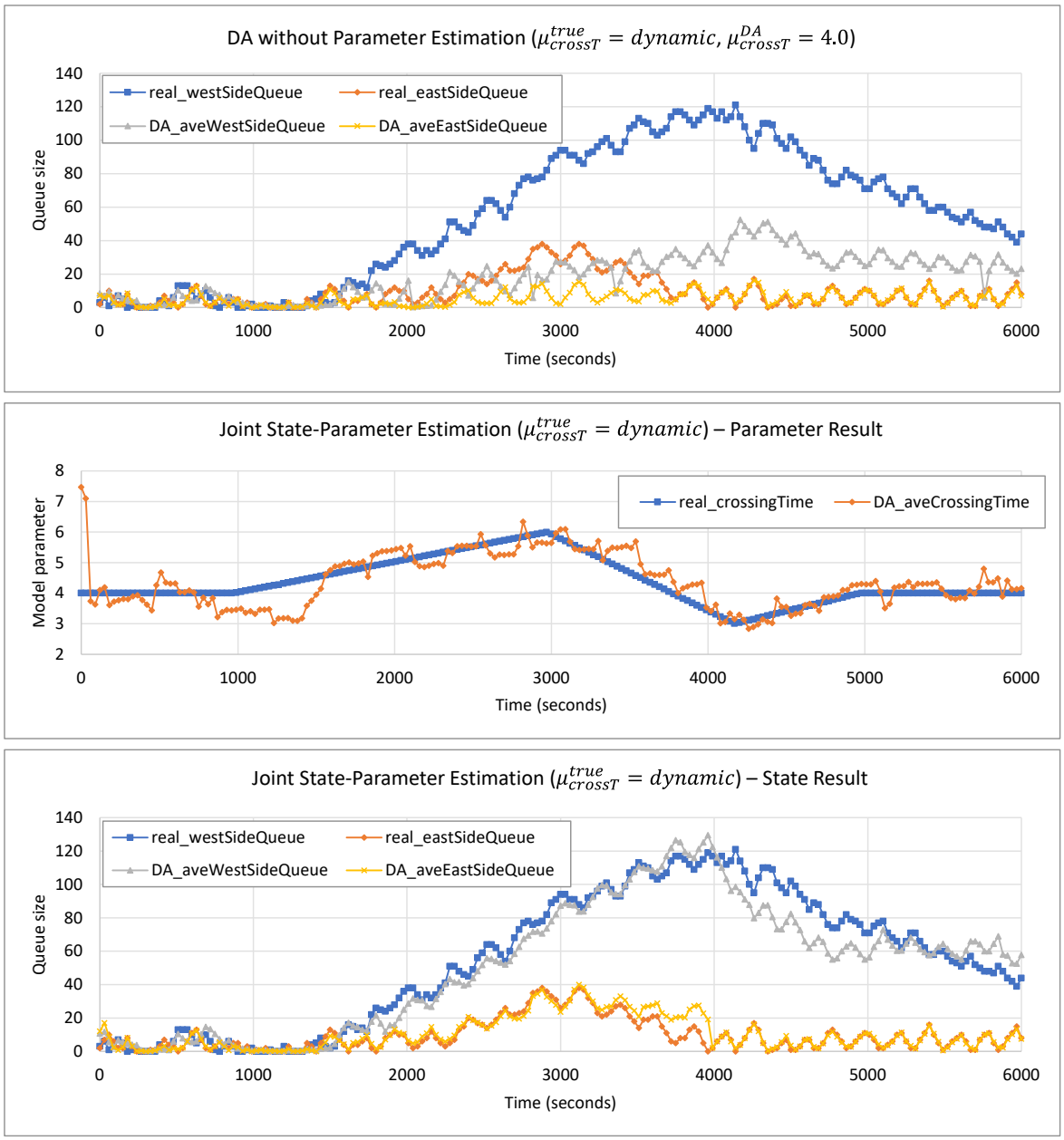


Figure 7: Data assimilation results for a system with dynamic parameter: (top) state estimation results without using parameter estimation; (middle) parameter estimation results using joint state-parameter estimation; (bottom) state estimation results using joint state-parameter estimation.

The dynamic changes of μ_{crossT}^{true} influence the number of vehicles queued on both sides of the road segment in the real system. One can see that the west-side queue starts to increase after 1000s. It reaches about 120 at around 4000s, and then gradually decreases. The east side queue increases after 2000s, reaches its peak (at about 40) at around 3000s, and then decreases to a low level at around 3800s. These changes are caused by the dynamical changes of the μ_{crossT}^{true}. Specifically, the μ_{crossT}^{true} starts to increase after 1000s, making it slower for vehicles to cross the road segment. Thus, the queues on both sides start to increase after 1000s. After the μ_{crossT}^{true} starts to decrease after 3000s, the east-side queue starts to decrease at around 3000s, and the west-side queue starts to decreases at around 4000s.

From the top diagram of Figure 7, one can see that the data assimilation without parameter estimation is able to estimate both queues before 1500s. After that, the estimations for both queues start to have

significant errors, with the error of the west-side queue reaches as high as 90 (at around 4000s). The significant estimation error for the west-side queue persists untill the end of the experiment, although the error starts to reduce after 4000s. One can also see that during the time when there exist significant errors, the estimated queue sizes are smaller than the true queue sizes. This is because the $\mu_{crossT}^{DA} = 4.0$ is smaller than the μ_{crossT}^{true} between 1000s and 3800s, which makes vehicles cross the road segment faster than they actually do in the real system. This incorrect model parameter leads to smaller estimated queue sizes.

For the joint state-parameter estimation, the middle diagram shows that the estimated parameter is able to track the dynamic changes of the true parameter over time. For example, during the time when μ_{crossT}^{true} increases from 4.0 to 6.0 between 1000s and 3000s, and then decreases from 6.0 to 3.0 between 3000s and 4200s, the estimated parameter increases and then decreases too. During the time when μ_{crossT}^{true} maintains at 4.0 (before 1000s, and after 5000s), the estimated parameter stays at around 4.0 too. This diagram also shows that the estimated parameter lags behind the changes of the true parameter. This is because without knowing if the true parameter actually changes or not, particles tend to maintain their parameter values unless there is significant evidence from measurement data showing otherwise. This tendency results in a delay for the estimated parameter to track the changes of the true parameter.

The bottom diagram shows that using the joint state-parameter estimation, the state estimation results are significantly improved. Estimates of both the west-side queue and east-side queue are able to track the true queue sizes with relatively small errors. This improvement is due to the fact that the estimation of the dynamical changes of μ_{crossT}^{true} reduces the discrepancies between the simulation model and the true system model, and thus lead to more accurate state estimation results.

To quantitively compare the results of data assimilation with and without using parameter estimation, we carry out 20 independent runs of data assimilation for each data assimilation activity and compute the RMSE of the state estimation results just like before. The resulting RMSE from data assimilation activity 1 (without parameter estimation) is 23.2, and from activity 2 (using joint state-parameter estimation) is 17.3. The smaller RMSE from the latter indicates that the joint state-parameter estimation improves the state estimation results. Similar results have been observed from other experiments using different dynamic change patterns of the μ_{crossT}^{true}. Due to space limitation, these results are not included in the paper.

6. Discussion and Conclusion

Several observations can be made based on the experiments described in the previous section. First, the data assimilation is able to assimilate real-time measurement data from a system to provide state estimates. In some cases, accurate state estimation can be achieved even when there are non-minor discrepancies between the simulation model and the true system model. This is because in these cases the measurement data carries "distinct" information about the true system state, assimilating which lead to accurate state estimates. For example, when the west-side queue has few vehicles (as exemplified by the top diagram of Figure 4), a relatively large discrepancy between μ_{crossT}^{true} and μ_{crossT}^{DA} still leads to estimation results that have small errors. On the other hand, when measurement data do not carry "distinct" information about the true system state (as exemplified by the bottom diagram of Figure 4), a discrepancy of the model parameters can lead to significant errors of the state estimates. In these cases, the joint state-parameter estimation can significantly improve the results, as demonstrated by the experiments in Section 5.3 and 5.4.

Second, the experiments show that the developed particle filter-based data assimilation method can effectively support joint state-parameter estimation for discrete event simulations. The results show that for a single parameter that is either static or dynamic, the estimated parameter is able to converge to and stay around the true parameter value. Furthermore, the joint state-parameter estimation leads to more accurate state estimations in most of the experiments, and does not degrade the estimation accuracy in other experiments where the data assimilation without parameter estimation already performs well. The fact that more accurate state estimates are achieved is because the joint state-parameter estimation allows the simulation model used in data assimilation to have less discrepancies from a real system under study.

The joint state-parameter estimation uses an augmented state vector (see equation (4)) that has a higher-dimensional state space. State estimation in a higher-dimensional state space requires the particle filter

algorithm to use more particles in order to converge to the true system state. All the experiments in this paper use 4000 particles. Nevertheless, preferably the joint state-parameter estimation should use more particles than the data assimilation without parameter estimation. The impact of the number of particles on joint state parameter estimation will be studied in future work. It is also important to note that more particles will lead to higher computation cost.

Another issue that is important for the joint state-parameter estimation is how to decide the level of perturbation for the dynamics of the parameter. All the experiments in this paper use the same Gaussian variance when adding perturbations to the parameter in each step, regardless if it is a static parameter or dynamic parameter. Ideally, if one knows that the parameter under estimation is a static parameter (i.e., does not change over time), a smaller variance is preferred. Otherwise, a larger variance should be used. The impact of variance on parameter estimation is another topic that needs further investigation.

This paper focuses on parameter estimation of a single parameter. The developed data assimilation method can be directly applied to joint state-parameter estimation with multiple parameters. Nevertheless, adding more parameters will bring more complexity such as the combination effect of multiple parameters that may not be distinguishable by the measurement data, as well as the higher dimension of the augmented state vector. An important future research is to develop online model calibration with multiple parameters.

To conclude, this paper presents a particle filter-based data assimilation method to support online model calibration for discrete event simulations. A joint state-parameter estimation problem is formally defined and a particle filter-based data assimilation algorithm is presented. A series of experiments are carried out to demonstrate the effectiveness and evaluate the different aspects of online model calibration. This paper focuses on online model calibration with a single parameter. Future work includes further investigating the performance of joint state-parameter estimation and developing online model calibration with multiple parameters.

Acknowledgments

This work is supported in part by the US Department of Agriculture (USDA) National Institute of Food and Agriculture (NIFA) under grant number 2019-67021-29011.

References

[1] Hu X. *Dynamic data-driven simulation: Real-time data for Dynamic System Analysis and prediction*. Hackensack, NJ: World Scientific Publishing Co. Pte. Ltd., 2023.

[2] Burgard W, Fox D. *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.

[3] Doucet A, Freitas N, Gordon N. An introduction to sequential Monte Carlo Methods. *Sequential Monte Carlo Methods in Practice* 2001; 3–14.

[4] Arulampalam MS, Maskell S, Gordon N, et al. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 2002; 50: 174–188.

[5] Rykiel EJ. Testing ecological models: The meaning of validation. *Ecological Modelling* 1996; 90: 229–244.

[6] Malleson N. Calibration of simulation models. *Encyclopedia of Criminology & Criminal Justice* 2014; 40: 115-8.

[7] Dahabreh IJ, Chan JA, Earley A, et al. Review of Validation and Calibration Methods for Health Care Modeling and Simulation. *Modeling and Simulation in the Context of Health Technology Assessment: Review of Existing Guidance, Future Research Needs, and Validity Assessment*. Rockville (MD): Agency for Healthcare Research and Quality (US), 2017.

[8] Rutter CM, Ozik J, DeYoreo M, et al. Microsimulation model calibration using incremental mixture approximate Bayesian computation. *The Annals of Applied Statistics* 2019; 13. Epub ahead of print 2019. DOI: 10.1214/19-aoas1279.

[9] McCulloch J, Ge J, Ward JA, et al. Calibrating agent-based models using uncertainty quantification methods. *Journal of Artificial Societies and Social Simulation* 2022; 25. Epub ahead of print 2022. DOI: 10.18564/jasss.4791.

[10] Kotthoff F, Hamacher T. Calibrating agent-based models of innovation diffusion with gradients. *Journal of Artificial Societies and Social Simulation* 2022; 25. Epub ahead of print 2022. DOI: 10.18564/jasss.4861.
[11] Manfren M, Aste N, Moshksar R. Calibration and uncertainty analysis for computer models – a meta-model based approach for Integrated Building Energy Simulation. *Applied Energy* 2013; 103: 627–641.
[12] Robertson J, Polly B, Collis J. Evaluation of automated model calibration techniques for residential building energy simulation. National Renewable Energy Lab.(NREL), Golden, CO (United States); Sep 2013.
[13] Yang Z, Becerik-Gerber B. A model calibration framework for simultaneous multi-level Building Energy Simulation. *Applied Energy* 2015; 149: 415–431.
[14] Bartin B, Ozbay K, Gao J, et al. Calibration and validation of large-scale traffic simulation networks: A case study. *Procedia Computer Science* 2013; 130: 844–849.
[15] Mudigonda S. *Methods for robust calibration of traffic simulation models*. PhD Dissertation, The State University of New Jersey, USA, 2014.
[16] Zhang K. *Real-Time Calibration of Large-Scale Traffic Simulators: Achieving Efficiency Through the Use of Analytical Mode*. PhD Dissertation, Massachusetts Institute of Technology, USA, 2020.
[17] Doekemeijer BM, Boersma S, Pao LY, et al. Online model calibration for a simplified les model in pursuit of real-time closed-loop wind farm control. *Wind Energy Science* 2018; 3: 749–765.
[18] Song H, Song M, Liu X. Online autonomous calibration of digital twins using machine learning with application to nuclear power plants. *Applied Energy* 2022; 326: 119995.
[19] Ward R, Choudhary R, Gregory A, et al. Continuous calibration of a digital twin: Comparison of Particle Filter and bayesian calibration approaches. *Data-Centric Engineering* 2021; 2. Epub ahead of print 2021. DOI: 10.1017/dce.2021.12.
[20] Titscher T, van Dijk T, Kadoke D, et al. Bayesian model calibration and damage detection for a digital twin of a bridge demonstrator. *Engineering Reports* 2023. Epub ahead of print 2023. DOI: 10.1002/eng2.12669.
[21] Liu J, West M. Combined parameter and state estimation in simulation-based filtering. In: Doucet A, de Freitas JFG, Gordon NJ, editors. *Sequential Monte Carlo methods in practice*. New York: Springer-Verlag; 2001. pp. 197–223.
[22] Hu X. A tutorial on Bayesian sequential data assimilation for dynamic data driven simulation. In: *P roceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM)* 2023. pp. 680–695.
[23] Xue H, Gu F, Hu X. Data assimilation using sequential Monte Carlo Methods in wildfire spread simulation. *ACM Transactions on Modeling and Computer Simulation* 2012; 22: 1–25.
[24] Hu X, Wu P. A data assimilation framework for discrete event simulations. *ACM Transactions on Modeling and Computer Simulation* 2019; 29: 1–26.
[25] Xie X, Verbraeck A. A particle filter-based data assimilation framework for discrete event simulations, *Simulation*, 2019, Vol. 95(11), 1027–1053
[26] Huang Y, Xie X, Cho Y, Verbraeck A. Particle filter–based data assimilation in dynamic data-driven simulation: sensitivity analysis of three critical experimental conditions, *Simulation*, 2023, Vol. 99(4), 403-415
[27] Hu X. Data Assimilation for Simulation-based real-time prediction/analysis. *2022 Annual Modeling and Simulation Conference (ANNSIM)* 2022. Epub ahead of print 2022. DOI: 10.23919/annsim55834.2022.9859329.
[28] Zeigler BP, Kim TG, Prähofer H. *Theory of modeling and simulation: Integrating discrete event and Continuous Complex Dynamic Systems*. 2nd ed. Amsterdam: Academic Press, 2000.

Biography

Xiaolin Hu is a full professor of the Computer Science Department at Georgia State University. He received his Ph.D. degree from the University of Arizona. His research interests include modeling and simulation theory and application, data assimilation, dynamic data driven simulation, and agent and multi-agent systems.

Mingxi Yan is a PhD candidate of the Computer Science Department at Georgia State University. Her research interests include modeling and simulation, and digital twin technology.