Data Assimilation in Agent Based Simulation of Smart Environments Using Particle Filters

Minghao Wang*, Xiaolin Hu

Department of Computer Science, Georgia State University, Atlanta, GA, USA

Abstract

Agent-based simulations are useful for studying people's movement and to help making decisions in situations like emergency evacuation in smart environments. These agent-based simulations are typically used as offline tools and do not assimilate real time data from the environment. With more and more smart buildings equipped with sensor devices, it is possible to utilize real time sensor data to dynamically inform the simulations to improve simulation results. In this paper, we propose a method to assimilate real time sensor data in agent-based simulation of smart environments based on particle filters (PFs). The data assimilation aims to estimate the system state, i.e., people's location information in real time, and use the estimated states to provide initial conditions for more accurate simulation/prediction of the system dynamics in the future. We develop a PF-based data assimilation framework and propose a new resampling method named as component set resampling to improve data assimilation for multiple agents. The proposed framework and method are demonstrated and evaluated through experiments using a sparsely populated smart environment.

Keywords: Data Assimilation, Smart Environment, Agent-based Simulation, Particle Filter, Occupancy Estimation

1. Introduction

Location information of occupants in building structures is useful for supporting decision makings such as scheduling efficient energy utilization and developing emergency evacuation strategies. Agent-based simulation is an important tool for studying the dynamics of people's movement in building structures [6, 20]. It uses a bottom up approach to model individuals' behavior and their interactions with the environment, and can provide valuable information for helping system design. While many agent-based simulations have been developed, they are typically used as offline tools and do not utilize real time data of building environments. With advances of sensor and communication technologies, more and more building environments are equipped with sensors that provide real time information about the environment. In this paper, we refer to indoor building environment equipped with sensors as smart environment. With real time data provided by sensors, how to dynamically assimilate these data into simulation to support real time decision making becomes an important research topic.

Data assimilation refers to the process of incorporating observation data into a running model to produce improved estimates of interested state variables [29, 30]. It is made necessary by the fact that no model is perfect, and therefore there is a need to obtain the best estimates of states (which are usually unobservable) of a dynamic system by incorporating observations into a model of the system. Data assimilation has gained much success in weather forecast and many geoscience fields. In this paper, we develop a data assimilation method for agent-based simulation of smart environments. The data assimilation aims to dynamically adjust the simulation to achieve more accurate simulation/prediction of people's movement in smart environments and thus to support real time decision makings. This is in contrast to traditional off-line simulation, where a simulation starts from an initial state and is not influenced by real time data during the simulation. With data assimilation, real time sensor data are assimilated to estimate the "current" system state. The estimated system states are then used to provide initial conditions for running simulations that simulate/predict the system dynamics in the future. Because the initial states are dynamically estimated from real time sensor data, simulations starting from these initial states will lead to more accurate simulation/prediction results. We note that this work is different from the work of dynamically calibrating or modifying the agent-based model based on real time data. In our work, the model is not dynamically changed. Instead, the simulation is dynamically reset to start from initial conditions that are estimated from real time sensor data, and thus the simulation/prediction results are dynamically adjusted in real time. Also note that even though this

* Corresponding Author. Tel : +16786469617
Email: csloader@gmail.com

work does not calibrate the agent-based model over time, it is possible to treat some of the model parameters as part of the system state and dynamically estimate (calibrate) those parameters based on real time sensor data.

Data assimilation relies on several elements to work, including the sensor data that provides real time observation of the system under study, the simulation model that captures the system dynamics, and the data assimilation algorithm for carrying out state estimation. Various sensor devices exist in smart environments. High resolution sensor devices such as video cameras provide more precise measurement; however they are often intrusive and incur high cost. In this paper, we consider binary proximity sensors that are low cost and provide non-intrusive information. Besides sensor data, the quality of the agent-based simulation model also has an impact on data assimilation results. Many agent-based simulation models have been developed in the literature for simulating people's movement. Since the main focus of this paper is on the data assimilation framework instead of the agent-based model, we develop data assimilation based on a relatively simple agent-based model. In this model, each agent is specified by several basic behaviors that simulate people's movement in smart environments. We note that the developed data assimilation framework is not restricted to the agent-based model and sensor data used in this paper – it can be adapted to work with other models and sensor data. The state estimation algorithm is another key component of data assimilation. A unique feature of the agent-based simulation model is that the model is specified by behaviors or rules and lacks the analytic structures (e.g., those in partial differential equation models) from which functional forms of probability distributions can be derived. This makes it difficult to apply conventional state estimation techniques such as Kalman filter and its variances. In our work, we carry out state estimation using Particle Filters (PFs), which are a set of sample-based methods that use Bayesian inference and stochastic sampling techniques to recursively estimate the state of dynamic systems from some given observations [8, 27]. PFs approximate the sequence of probability distributions of interest using a large set of random samples, named particles, each of which represents an estimation of the system state. PFs are non-parametric filters and thus work well with the agent-based simulation model.

Based on PFs, this paper develops data assimilation for agent-based simulation of smart environments. We first present the data assimilation framework based on the bootstrap filter algorithm [8]. Within this framework, the system state is composed from the states of all agents, and the state transition function is defined based on the agent-based simulation model. One issue associated with the standard bootstrap filter algorithm is that it suffers from the particle deprivation problem due to the high dimensional system state associated with multiple agents. In this paper, observing that the high dimensional system state is composed from states of individual agents (where each agent's state variables can be thought of as one component of the overall system state), we propose a PF algorithm with a new resampling method named as *component set resampling*. In the proposed component set resampling, instead of resampling each particle as a whole as in standard particle filters, we divide the system state into sub-components and resample at the component level to “construct” new particles. The goal is to use the same number of particles to represent more possible combinations of system state variables, and thus alleviate the particle deprivation problem. We show how the component set resampling can work together with the standard resampling to support data assimilation for agent-based simulation of smart environments. Experiment results based on a sparsely populated smart environment are provided to demonstrate the proposed data assimilation framework and method. This paper makes contributions in the following two aspects: 1) it develops a data assimilation framework for agent-based simulation of smart environments; 2) it proposes a new resampling method (the component set resampling) to improve data assimilation for multiple agents. As far as we know, this is one of the first efforts that assimilate real time sensor data in agent-based simulation of smart environments.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 describes the smart environment and the agent-based model used in this work. Section 4 presents the data assimilation framework. Section 5 shows experiment results and analysis. Section 6 provides some discussions of this work, and Section 7 concludes this work.

2. Related Work

A smart environment is equipped with sensors that provide information about the environment to help accomplishing tasks such as navigation, energy consumption scheduling, activity notification, and evacuation planning. Occupants' location information is one of the most important information that one would like to derive from the sensors because occupancy is directly related to many aspects of the contextual information of a building. One research topic related to deriving moving objects' location information from sensors is target tracking. The problem of target tracking is usually divided into two sub problems: 1) data association problem, which links a specific target to the observation data it generates; 2) single target tracking problem, which identifies a target's

trajectory in an environment using estimation techniques [15]. Binary proximity sensors are often used in target tracking due to their non-intrusive and inexpensive properties [17, 18]. Because these sensors provide no identity information about the targets, it is difficult to link observation with specific target. Techniques such as target counting have been developed to help address this issue [31, 32]. The work of target tracking deals with general moving objects (not limited to people). They typically assume motion models specified by equations based on physical laws instead of agent-based models that model occupants' behavior in smart environments.

Agent-based models are widely used to simulate people's moving behavior in building environments. Examples include an agent-based model for analyzing pedestrian movement in a multi-story office building [6], and an agent-based simulation to study people's movement in a railway station [20]. These works study the dynamics of people's movement to help system design. They are used as offline tools and do not assimilate sensor data from the buildings under study. With sensors increasingly deployed in smart environments, how to incorporate sensor data into agent-based models becomes a relevant research topic. In [22], the author developed an agent-based model to model the movement of occupants and fire emergency in a building. A guidance system was developed that uses real time sensor data to help agents to find a shortest path to the exit when there is a fire emergency. The focus of using the sensor data was to dynamically update the guidance system to be used by the agents, which is different from our work of estimating occupants' location for more accurate simulation results. Several works used models and data to help estimating building occupancy. The work of [14] combined sensor data, building structure and prior knowledge about the building utilization to estimate building occupancy. The estimation problem was regularized as solving a convex optimization problem based on the analytic structure of the occupancy dynamics and the observation function. The work of [19] combined two types of sensors and applied extended Kalman filter to estimate zone level occupancy information. This work used a reduced-order flow-based model represented on a graph so that the extended Kalman filter can be applied. The work of [13] proposed an integrated approach to occupancy modeling and estimation. In this work, the occupancy estimation was carried out using a covariance graph model based on the least minimum variance estimator to estimate room-level occupancy in the building. To evaluate the estimation method, an agent-based stochastic model was developed and was validated with sensor data for a special case of one room and one occupant. The developed agent-based model was then used to generate "artificial" sensor data for evaluating the occupancy estimation method. This work is different from our work that carries out data assimilation using the agent-based model.

Utilizing data has been a recent trend in the agent-based modeling and simulation community. The different ways of using data can be categorized into three broad approaches, each of which serves a different purpose. The first approach emphasizes the importance of data during the model development stage for informing the model development. The work of [11] argued that traditional agent-based modeling in social simulation does not use data and thus this work promotes data-driven agent-based modeling. The works of [24, 3] discussed the potential of using data mining technologies during the development of agent-based models. A framework to structure agent-based modeling data for social-ecological systems was presented in [1]. The second approach focuses on incorporating data for model validation. The work of [23] identified the need of incorporating data for validating the built agent-based model and argued that all the models should be validated in order to be published. A detailed discussion about empirical validation of agent-based models was provided in [25]. A set of guidelines for improving the predictive performance and the robustness of agent-based models were proposed in [26], where a main topic was data collection and model validation. The third approach explores how to use data to set up simulation runs. For example, the works of [12, 21, 5] set up agent-based models using contextual data or data from observations. The need of using data for simulation initialization was discussed in [11]. Our work is related to simulation initialization to some extent because we carry out state estimation and then use the estimated system state as initial conditions for running simulations. However, we focus on a dynamic process where real time sensor data are dynamically assimilated to estimate the system state during the simulation procedure.

The deprivation problem in high-dimensional particle filtering is well documented (see [28] for a detailed discussion on this). Several methods have been proposed in the literature to deal with the deprivation problem, including diversifying the particles [10], adjusting number of particles [9], and dealing with high dimensionality of state space analytically [16]. The idea of diversifying particles [10] is incorporated in our work by adding noise to the particles in each estimation step. Other methods (e.g., the one described in [16]) generally exploited the analytical properties of the models so that advanced techniques can be applied. They are not suitable for agent-based models that are specified by behaviors/rules and lack analytical structures. In this paper, we develop a novel resampling method (the component set resampling) to alleviate the particle deprivation problem for data assimilation in agent-based simulation of smart environments.

3. The Smart Environment and Agent-based Simulation Model

To set the stage for the proposed data assimilation framework, we first describe the smart environment and sensors considered in this work, and then present the agent-based simulation model used in the data assimilation.

3.1. The Smart Environment and Sensors

In our work, we consider the smart environment as a single-floor indoor office environment composed of office rooms and corridors. The environment is equipped with binary proximity sensors, which report 1 when one or more occupants are within its sensing area and reports 0 otherwise. These sensors have low resolution in detecting the movement of occupants and have the following features:

1. The sensors provide anonymous location information of occupants. They do not provide information about occupants' identity.
2. The location information obtained is ambiguous when there are multiple occupants in the sensing area of a sensor. The sensor does not have the capability of measuring the number of occupants in its detection area.
3. The sensors are subject to errors and environmental noises.
4. Sensors are deployed sparsely. As a result, occupants' locations become latent variables when occupants are outside sensors' detection areas.

These features make it impossible to directly measure occupants' location and to differentiate different occupants based on sensor data. The goal of data assimilation is to dynamically estimate occupants' location and other system state variables to supply initial conditions for more accurate simulation/prediction of the system dynamics.

3.2. The Agent-based Model(ABM)

We develop a relatively simple agent-based simulation model and use it to carry out the data assimilation work. The agent-based model runs in a stepwise fashion. It models individual occupants' moving behavior such as navigating and avoiding each other. It adopts a behavior-based control mechanism, in which behaviors that match the current world condition may fire subject to conflict resolution among competing alternatives [2]. More specifically, we model each agent to have two basic behaviors in the smart environment: *avoidance* and *move-to-destination*. We note that the main focus of this paper is on the data assimilation, thus we do not add other behaviors such as follow-crowd and emergency evacuation. Those behaviors can be added into the agent-based model if needed, and the developed data assimilation framework should still work if more behaviors are modeled. Also note that in this work, we assume agents' walking speed is a constant and does not change over time.

The goal of the avoidance behavior is to enable an agent to perform appropriate maneuvers when it is blocked by other agents or obstacles. Since the corridors and office rooms typically have limited space, congestion can occur when multiple occupants are in the vicinity of each other. When that happens, agents stop moving to their destinations and follow some rules trying to resolve the congestion. In our model, an agent performs its avoidance behavior according to three rules: *keeping comfortable distance*, *moving sideways*, and *re-planning a temporary path*. The first rule, keeping comfortable distance, defines that agents try to stay away from each other if they get too close to each other: an agent will move to the opposite direction of its nearest agent if their distance is smaller than a comfortable distance threshold. The moving sideways rule defines that if an agent is blocked on its heading direction (i.e., if the distance to the nearest agent/obstacle on its heading direction is smaller than a pre-defined distance threshold), the agent scans its side ways to seek for a direction that is clear of obstacles and move towards that direction. The third rule, re-planning motion, is applied when the agent does not find a direction that is clear of obstacles while applying the second rule; when this happens, the agent tries to re-schedule a new route so that it can pass the obstacle. We note that the first rule and the second rule (or the third rule) are not mutually exclusive because they are triggered by different conditions. It is possible for an agent to invoke the first rule and then the second rule (or the third rule) in the same time step. These rules together help agents to avoid each other while moving and when congestions happen.

The goal of the move-to-destination behavior is to navigate an agent to a pre-defined destination so that the "planning" type of behavior can be modeled. Occupants in office environment usually move rationally and their movements are associated with clear purposes. This means instead of walking around randomly, an occupant would set a destination before starting to move. After that, the moving direction of the occupant can be determined based on its current location and its destination. In our work, we implement the move-to-destination behavior using a

waypoint graph. This graph is constructed in such a way that for an arbitrary position on the floor map, there is at least one vertex in the waypoint graph that can be connected to the position without crossing any obstacles (e.g., walls). Typically, the vertices of such a waypoint graph are set to be the intersections between corridor and corridor or between corridor and room. Fig 1 shows an example of the waypoint graph for a floor structure:

Fig 1. Floor structure with waypoint graph

In Fig 1, the blue bold lines represent the walls. The black circles represent the vertices (named as waypoints) of the waypoint graph and the lines connecting them represent the connectivity among the waypoints. We assume all agents are aware of the floor structure and therefore have access to the waypoint graph. With the waypoint graph, an agent generates a set of intermediate route points that navigate to the final destination based on its current location and its final destination. To generate the intermediate route points, an agent connects its current location and its final destination to all the possible vertices in the waypoint graph without crossing any walls; the agent then uses Dijkstra's[7] algorithm to find a shortest path from its current location to the final destination. The series of waypoints on the shortest path is the sequence of intermediate route points of the agents. At any given time, an agent's moving direction is determined based on its current location and the next intermediate route point. Formally, let G represent a waypoint graph $G = \{V, E, D\}$, where V, E, D are sets of the vertices, edges, and distances of the edges, respectively. Each element in V has a coordinate that indicates the location of the waypoint. To generate a route from an agent's starting location l_1 to its destination l_2 , the following procedure is used:

1. Add starting location l_1 and final destination l_2 of the agent to the waypoint graph G by connecting them to all the possible vertices in V such that the new generated connections(edges) do not cross any obstacles (walls in our work). The graph with the new edges is represented by G' .
2. Find a shortest path from the starting location l_1 to the destination l_2 over the graph G' .
3. Store the waypoints of the shortest path in a list d :

$$d = \{l_1, m_{x_1, y_1}, m_{x_2, y_2}, \dots, l_2\}$$

, where m_{x_i, y_i} are the intermediate route points in the route. After the route is determined, an agent set its moving direction according to its current location and the next intermediate route point. After the agent arrives at an intermediate route point, that point is removed from the list d and the next element in the list is chosen as the next intermediate route point. Once the agent reaches its final destination, a new destination is generated (currently it is randomly generated) and the agent moves to the new destination.

The avoidance behavior and move-to-destination behavior together define how an agent updates its moving direction and plans its motion. To resolve conflict between the two behaviors, we set the avoidance behavior to have higher priority than the move-to-destination behavior in the behavior-based control. In other words, the move-to-destination behavior is performed only when there is no need to avoid obstacles or other agents.

The agent-based simulation model is specified by a set of behaviors and rules. While this brings advantages like the ability to add new behavior on top of existing ones, it adds complexity to data assimilation because the prior and posterior distributions of system state variables cannot be expressed in analytical forms (e.g., using equations). In this work, we use the non-parametric particle filters to carry out data assimilation for the agent-based simulation model. We describe the particle filter-based data assimilation framework in the following section.

4. Data Assimilation Framework

4.1. Overview of Particle Filter-based Data Assimilation

Data assimilation involves incorporating observations into a computer model to obtain the best estimate of the system state. It is generally framed as a state estimation problem for calculating posterior probability distributions of

some state variables of interest, given prior distributions from a forecast model and data from observations. In this work, we develop data assimilation based on Particle Filters (PFs), which are a set of sample-based methods that use Bayesian inference and stochastic sampling techniques to recursively estimate the state of dynamic systems from some given observations (see [8, 27] for more details about PFs). Formally, particle filters work with a state-space model that has a generic form containing a state transition function (also called the system transition model) and a measurement function (also called the observation model):

$$x_t \sim p(x_t / x_{t-1})$$

$$y_t \sim p(y_t / x_t)$$

, where x_t is the system state at time t evolved from state at the previous time step $t-1$; and y_t is the measurement at time t given the system state at time t . In particle filters, posterior distribution of the system state is approximated by a set of weighted particles (also called samples), where each particle represents an estimation of the entire system state. The weight of a particle is calculated based on the likelihood of the particle, i.e., the probability for the state represented by the particle to generate the observation (sensor data collected from the real system). Various particle filters have been developed. In this work, we carry out data assimilation using the bootstrap filter [33], which is one of the basic particle filter algorithms. In the bootstrap filter, particles are propagated over time by sampling from the prior distribution that represents the state transition, which is the simulation model in our work.

Data assimilation using the bootstrap filter runs in an iterative manner. In each iteration the algorithm receives a set of particles representing the previous belief of the system state, and an observation, and then generates a new set of particles representing the posterior belief. Each iteration includes three major steps. The first step is the sampling step, which involves drawing samples (particles) from the state transition function, i.e., the agent-based simulation model. To do this, for each particle we use the particle's state as the initial state and run the simulation to the next observation time according to the agent-based model. The second step is to calculate the weights of particles, which involves weighing a particle based on the probability for the particle to generate measurement that matches the observation (i.e., the sensor data, which is the ground truth measurement). To achieve this we need to define the observation model according to the specific type of sensors used in the system, and then calculate measurement of each particle using the observation model. The weight of a particle is then calculated based on the distance between the ground truth measurement and the measurement of each particle. The third step is resampling, which involves multiplying particles with higher weights and eliminating those with lower weights. In the standard resampling method, a set of offspring particles are drawn with probability proportional to the normalized particle weights. To effectively increase the diversity of the sample space and to represent a larger sample space using limited number of particles, in this paper we also developed a new resampling method named as the *component set resampling*. The set of resampled particles represent the posterior belief of the system state, and used for the next iteration of the data assimilation. We note that the data assimilation step is different from the simulation step. The data assimilation step is typically defined by how often sensor data are collected/assimilated, and may cover multiple simulation steps. In this paper, unless explicitly stated, the time step t represents the data assimilation step instead of the simulation step.

Applying particle filters to data assimilation for the agent-based model needs to develop several key components. These include defining the state and formulating the agent-based model as a state space model, defining the observation model that computes sensor data, and developing the weight calculation and resampling methods. Below we describe these components in detail.

4.2. State and State Transition Function

The state transition function defines how the system state evolves over time, and is used in the particle filter algorithm to generate new samples in each data assimilation step. In this work, we define the state of the system as the locations and destinations of all the agents in the smart environment. Since agents' moving directions and intermediate destinations, i.e., routing points, are calculated from agents' locations and destinations (e.g., an agent's moving direction is decided by its location and its next intermediate routing point), we do not include them explicitly as part of the system state. Formally, the state of the system is defined as:

$$x_t = \langle r_t^1, r_t^2, r_t^3, r_t^4, \dots, r_t^m \rangle$$

$$r_t^i = \langle l_t^i, d_t^i \rangle$$

, where m is the total number of agents and r_t^i is the state of agent i at time t , and l_t^i, d_t^i represent the current location and final destination of agent i at time t .

The state transition function defines how the state of a particle evolves from time step t to time step $t+1$ and is based on the agent-based model described in section 3. The state transition function is formulated as follows.

$$x_{t+1} = ABMTransition(x_t) + Q_t$$

where Q_t is the processing noise and $ABMTransition$ represents the agent-based model, which define how agents update their locations over time (see Section 3.2). Once an agent reaches its current final destination, a new destination is generated. Since particle filters work with stochastic models, in our work the new destination is generated stochastically based on a distance that is measured by how many waypoints away from the agent's current destination in the waypoint graph. Specifically, we use the following method to generate a new destination once an agent reaches its current final destination:

1. Determine the heading direction, denoted by θ_a , of the agent. The heading direction is the direction where the agent moves to before reaching the destination.
2. Generate a waypoint distance D' from the current destination. Let $D' = \lfloor GP(D, 1) \rfloor$ where GP is a Gaussian process with mean to be D , which is a pre-defined constant and represents the average waypoint distance from the current destination, and variance to be 1. D' is the number of waypoints that the new destination will be set away from the current destination.
3. Set initial value $\theta = \theta_a$, $l = l_a$, where θ_a is the heading direction of the agent and l_a is the location of the agent. The following procedure is executed:

For $i=1$ to D'

- Decide a new moving direction θ' based on direction θ . This is done by connecting location l to all of its nearest waypoints in the waypoint graph and then selecting a direction along the edges of these connections. The probability of selecting a direction that is opposite to the direction of θ is set to be a pre-defined constant p_b and the probability for the agent to select other directions is set to be $1-p_b$.
- Find the next waypoint $l_{waypoint}$ on the waypoint graph along the direction θ' .
- Set $\theta = \theta'$, $l = l_{waypoint}$.

End For

4. Randomly pick a position between l and its nearest waypoint (rejecting the points that are inside the walls). This position is considered as the new destination of the agent.

To improve the quality of data assimilation, how to model the process noise Q_t is also important. In our work, we model the process noise from two aspects: the noise added to the location of the agents and the noises added to the destinations of the agents. For agents' locations, we simply add a random position noise using uniform distribution within a range. We also add noise to agents' destinations so that particles re-sampled from the same parent can have different destinations. The method for adding noise to agents' destinations is similar to the method for generating new destinations described above, except that l is initialized with the destination instead of the position of the agent and θ is initialized with the heading direction of the destination (the direction from the second last intermediate destination to the final destination). Fig 2 shows an example of adding noises to a destination:

Fig 2. An example of adding noise to a destination to generate new destinations.

In Fig 2, the black dot represents the original destination and the green dots are different instances of destinations generated by adding noises. Totally 100 instances of new destinations (green dots) are displayed by randomly adding noises to the original destination (black dot) using the method described above. In this example, D is set to 1; the heading direction of the original destination is 180 degree (from right to left); the probability p_b for generating a new destination on the opposite direction is 30% (which is arbitrarily chosen). As can be seen, the new

destinations are generated around the original destination, with most of them are in the forward areas along the heading direction of the original destination, and the majority of them are within 1 waypoint distance from the original destination.

4.3. The Observation Model and Weight Calculation

In our work, the observation data is collected from a set of binary proximity sensors. A sensor produces 1 if there is at least one occupant in its detection area and produces 0 otherwise. Assuming there are n sensors deployed in the smart environment, the measurement is then a vector containing n 0s or 1s. Formally, given a set of occupants:

$$O = \{o_1, o_2, o_3, o_4, \dots, o_m\}$$

, where m is the total number of occupants; a set of sensors, S , that are deployed sparsely in the environment:

$$S = \langle s_1, s_2, s_3, s_4, \dots, s_n \rangle$$

for each sensor s_i , its location is denoted as l_i^s and its detection area is specified by a distance r_i^s , which is the radius of the area it is able to observe. The observation y_i generated by s_i at time k is then defined as follows:

$$y_i = h(\text{Observe}(O, l_i^s), \omega_{i,k})$$

$$\begin{cases} \text{Observe}(O, l_i^s) = 1 & \text{if } \exists o_j \in O, \text{dist}(l_j^o, l_i^s) < r_i^s \\ \text{Observe}(O, l_i^s) = 0 & \text{otherwise} \end{cases}$$

where l_j^o is the location of occupant o_j and $\omega_{i,k}$ is the observation noise of sensor s_i (i.e., the probability for the sensor to report false readings), $\text{dist}()$ is a function that returns the distance between two positions, $h()$ is a process that takes into account the observation noise of the sensor to produce an observation. In our work, we assume all sensors have the same level of observation noise of 5%. This means the probability that a sensor reports 0 if there are occupants in the sensor's detection area and the probability that the sensor reports 1 if there is no occupant in the sensor's detection area are both 5%.

To calculate the importance weight of a particle, the ground truth measurement vector y_{real} is compared to the measurement vector y_{particle} of the particle. The two measurement vectors are denoted below:

$$y_{\text{real}} = \langle y_1^{\text{real}}, y_2^{\text{real}}, y_3^{\text{real}}, \dots, y_n^{\text{real}} \rangle$$

$$y_{\text{particle}} = \langle y_1^{\text{particle}}, y_2^{\text{particle}}, y_3^{\text{particle}}, \dots, y_n^{\text{particle}} \rangle$$

In the bootstrap filter, the weight of a particle is calculated based on the likelihood $p(y_i/x_i)$. Since the measurement is a vector, like in many Kalman Filter applications, we define that the likelihood as a multivariate Gaussian distribution. In our work, the sensors are independent from each other. The importance weight w_k^j of a particle j at time k is calculated as follows:

$$w_k^j = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}} e^{-\frac{d_i^2}{2}}$$

, where d_i is the distance between the measurement of the particle and ground truth measurement for sensor s_i . We define that d_i is calculated as follows:

$$\begin{cases} d_i = 0 & \text{if } y_i^{\text{real}} = 1 \wedge y_i^{\text{particle}} = 1 \\ d_i = 1/a & \text{if } y_i^{\text{real}} = 0 \wedge y_i^{\text{particle}} = 0 \\ d_i = a & \text{otherwise} \end{cases}$$

In our work, a is arbitrarily chosen to be 2, thus $a > 1/a > 0$. For a sensor, we assign different values to d_i in different situations to indicate the different information it carries in calculating the importance weight of a particle: 1) when the ground truth measurement and the particle's measurement both have value 1, this means the two measurements match and carry strong information (the sensor detects occupant movement within its detection range). In this case, $d_i = 0$ to indicate the distance between the particle's measurement and the ground truth measurement is the smallest; 2) when both have value 0, this means the two match but carry less information (the

sensor does not detect occupants within its detection range), and thus d_i has a larger value but is still smaller than 1; 3) when the two measurements mismatch, d_i has a value greater than 1.

4.4. The Particle Filtering Algorithm

The particle filtering algorithm is the bootstrap filter, which follows the standard sequential importance sampling with resampling (SISR) procedure [8]. The algorithm first initializes all the particles so that they are randomly distributed over the state space. After that, samples are drawn by advancing the state of each particle using the state transition function described in section 4.2 for a period of time Δt (Δt depends on how often the observation data is collected and assimilated). The weight of each particle is calculated based on the comparison between the measurement computed using the observation model and the real observation data collected from the sensors. The weights are then normalized so that in the resampling step, particles with higher weights have greater chances to be selected and reproduced.

Resampling in particle filters deals with the degeneration problem by eliminating particles with lower likelihood and multiplying those with higher likelihood. The resampling step first normalizes particles' weights so that the sum of all particles' weights equals to 1. The normalized weight of a particle then becomes the probability for the particle to be selected during the resampling. A new set of particles is generated by performing the selection N times (where N equals to the total number of the particles). The procedure of a standard resampling is summarized as follows:

Algorithm I: Standard Resampling

Standard Resampling

Resampling step:

Input: The set of particles $\{x_t^{(i)}; i = 1, \dots, N\}$ with associated weights $\{\tilde{w}_t^{(i)}; i = 1, \dots, N\}$;

Output: A new set of particles $\{\hat{x}_t^{(i)}; i = 1, \dots, N\}$

1. Normalize the weights according to:

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^N \tilde{w}_t^{(j)}}$$

2. **For** $k = 1$ to N

Select particle $\hat{x}_t^k$ from the set :

$X = \{x_t^{(i)}; i = 1, \dots, N\}$ according to $W = \{w_t^{(i)}; i = 1, \dots, N\}$

End for

Although avoiding the degeneration problem, the standard resampling method brings two potential issues: sample impoverishment and particle deprivation. The sample impoverishment phenomena occurs when after several iterations, all the particles are generated by multiplying a small number of particles (because these particles have dominant weight or because other particles have much lower likelihood.). As a result, the diversity of the particles decreases after resampling and particles tend to collapse to a single point or several identical points on the state space after some steps. At this stage, due to loss of diversity, if the states of particles are all far away from the true state of system, there is no way for particles to rewind and recover. The other issue of particle deprivation occurs when there is no particle in the vicinity of true state; as a result, it is impossible for the algorithm to converge to posterior. Particle deprivation is caused by many reasons. One of the reasons is loss of diversity of particles introduced by sample impoverishment. Another reason is combinatorial explosion of possible states due to high dimensional state space, which means that the number of possible combinations of state values increases exponentially as the dimension of system state increases. For high dimensional state space, a large number of particles are required in order to cover all possible combinations of state values to prevent deprivation, thereby significantly increasing the computation cost [4]. In this work, we aim to estimate occupants' location on a 2D space. Since each agent has two state variables (location and destination), the number of dimensions of the entire state space is $2 \times n$ where n is the total number of agents. As the number of agents increases, the high dimensional state space of the system leads to deprivation caused by combinatorial explosion and loss of diversity. To alleviate this problem, we propose a new resampling method named as component set resampling as described below.

4.4.1. Component Set Resampling

In the standard particle filter resampling procedure, the entire system state is treated as a whole when being re-sampled. Since the system state is a vector consisting of many state variables, it can be thought of having many components, each of which is a sub-set of the state variables. For the agent-based simulation model, the system state is composed from states of individual agents, thus each agent's state variables can be treated as a component of the whole system state. For example, for an agent-based model with two agents *agent1* and *agent2*, one component would be *agent1*'s location and destination; another component would be *agent2*'s location and destination. The basic idea of the component set resampling is to break the system state into components and carry out resampling at the component level. In other words, instead of resampling each particle as a whole as in standard particle filters, we divide the state space into components and resample from the set of components to "construct" new particles. By reconstructing new particles with components from different particles, we aim to increase the diversity of the sample space using the same number of particles, because new combinations of components can be generated.

The particle filtering algorithm with the component set resampling differs from the standard particle filtering algorithm only in how particles are re-sampled. It does not change the weight calculation of particles, that is, a particle is still treated as a whole when calculating its importance weight. After weight calculation and normalization, we divide a particle into sub-components, all of which inherit the same weight from the particle. In this work, the components are divided based on the agents in the system, where each agent's state (location and destination) is a component. Then for each component (e.g., *agent1*' state), we construct a component set that consists of the corresponding component instances from all particles. When reconstructing a new particle, we select an instance (according to components' weight) from each component set for all the component sets to form a new particle. Formally, let

$$x^i = \langle r_1^i, r_2^i, r_3^i, r_4^i, \dots, r_m^i \rangle$$

represents the system state of particle i , where m is the number of components in the system state (in our work, m is the number of agents in the system), and

$$r_j^i = \langle l_j^i, d_j^i \rangle$$

represents agent j 's state in particle i , i.e., agent j 's position and destination. Overall, the states of particle i consists of components $r_1^i, r_2^i, r_3^i, \dots, r_m^i$. Each component r_j^i has the same important weight that is inherited from the particle's importance weight.

$$w_t^{r_j^i} = w_t^i = p(y_t | x_t^i)$$

Since there are m agents, there are m component sets. Assuming there are N particles, these m component sets are defined as:

$$\begin{aligned} R_1 &= \{ r_1^1, r_1^2, r_1^3, r_1^4, \dots, r_1^N \} \\ R_2 &= \{ r_2^1, r_2^2, r_2^3, r_2^4, \dots, r_2^N \} \\ R_3 &= \{ r_3^1, r_3^2, r_3^3, r_3^4, \dots, r_3^N \} \\ &\vdots \\ R_m &= \{ r_m^1, r_m^2, r_m^3, r_m^4, \dots, r_m^N \} \end{aligned}$$

, where R_k is the k th component set, and each r_j^i has an associated weight $w_t^{r_j^i}$. With these components sets, a new particle is constructed as:

$$\tilde{x}^i = \langle \tilde{r}_1^i, \tilde{r}_2^i, \tilde{r}_3^i, \tilde{r}_4^i, \dots, \tilde{r}_m^i \rangle$$

where

$$\tilde{r}_k^i = \text{select}(R_k)$$

and $\text{select}(X)$ is the function of multinomial selection to select a component from set X according to the weight associated with the components.

Fig 3 illustrates how the component set resampling works. For simplicity, we assume each component consists of only agent's location (the destination is not considered). In this example, there are 3 particles where each particle is composed from 3 components l_1 , l_2 , and l_3 . In the figure, P^1 , P^2 and P^3 denote the three particles before the resampling, and l_i^j represents the component l_i of particle P^j . Assuming the weights of the three particles P^1 , P^2 and P^3 are w^1 , w^2 , w^3 respectively, we have $w^2 > w^1$, and $w^1 = w^3$. Fig. 3(a) illustrates the standard resampling where the 3 particles are selected according to their weights (the left part of the figure shows the three particles before the resampling, and the right part of the figure shows the resampling results). In this example, P^1 is selected once, P^2 is selected twice, and P^3 is not selected. This is because $w^2 > w^1$, and $w^1 = w^3$, thus P^2 is more likely to be selected than P^1 and P^3 . Fig. 3(b) illustrates the component set resampling where resampling is carried out at the component level. The three vertical dashed boxes show the component sets for l_1 , l_2 , and l_3 , respectively before resampling. As can be seen, each component set is re-sampled and their results are combined together to form a new particle. Taking the second new particle (the middle one as shown in the figure) as an example, this particle is composed from the first component l_1^1 of P^1 , the second component l_2^2 of P^2 , and the third component l_3^2 of P^2 . The components from P^2 are more likely to be selected in constructing new particles because P^2 has the highest weight and all the components from P^2 inherit its weight.

Fig 3. Comparison of standard resampling and component set resampling.

The component set resampling procedure is summarized as follows:

Algorithm II: Component Set Resampling

Component Set Resampling

Resampling step:

Input: The set of particles $\{x_t^{(i)}; i = 1, \dots, N\}$ with associated weights $\{\tilde{w}_t^{(i)}; i = 1, \dots, N\}$;

Output: A new set of particles $\{\hat{x}_t^{(i)}; i = 1, \dots, N\}$

1. Normalize the weights according to:

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^N \tilde{w}_t^{(j)}}$$

2. **For** $i=1$ to N

Divide state space into m components

$$x^i = \langle r_1^i, r_2^i, r_3^i, r_4^i, \dots, r_m^i \rangle;$$

Assign weight $w_t^{r_j^i} = w_t^i = p(y_t^i | x_t^i)$ to each component r_j^i ;

End for

For each component r_k^i , $k=1$ to m

create a component set R_k so that:

$$R_k = \{r_k^1, r_k^2, r_k^3, r_k^4, \dots, r_k^N\}$$

End For

For $i=1$ to N

For $k=1$ to m

select a component from R_k according to components' weights

$$\hat{r}_k^i = \text{select}(R_k);$$

End for

construct the new particle $\tilde{x}_t^i = \langle \tilde{r}_1^i, \tilde{r}_2^i, \tilde{r}_3^i, \tilde{r}_4^i, \dots, \tilde{r}_m^i \rangle$
End for

The component set resampling diversifies the sample space even when a limited number of particles are used. However, it also brings a problem: when dividing a state vector into components, a “good” particle with the right combination of components may be broken during the resampling procedure. Furthermore, when different components of the system state have dependencies over each other, a newly constructed particle may not hold the dependencies among the components. To ensure the robustness and correctness of the particle filtering algorithm, we propose to combine the component-set resampling and the standard resampling to solve the above problem and to take advantage of both. Specifically, during the resampling procedure, we re-sample one portion of the total particles using the standard resampling and re-sample the rest of the particles using the component set resampling. The goal of the standard resampling is to ensure the robustness and correctness of the algorithm, and the goal of the component set resampling is to diversify the sample space with limited number of particles. In our work, we have set that 20% of the particles are re-sampled from the standard resampling and 80% of the particles are re-sampled from the component set resampling. We note that these percentage numbers are empirically determined based on experiments for the agent-based model. In our application, the dependencies among the different agents are relatively weak and thus the majority particles are re-sampled from the component set resampling. We expect that for other applications these percentage values need to be changed in order to achieve best performance. A guideline about what percentage values to use can be established based on the level of dependency among the components, and that is considered as future work.

The final resampling procedure is named as the *mixed component set resampling* and is described below, where we assume the total number of particles is N , among which U particles are re-sampled from the standard resampling and V particles are re-sampled from the component set resampling. We note that the standard resampling procedure can be thought as a special case of the mixed component set resampling when $U=N$.

Algorithm III: Mixed Component Set Resampling

Mixed Component Set Resampling

Resample particle set with $N=U+V$ particles

1. **For** particle $x_t^1 \dots x_t^V$
Generate particles using the **Standard Resampling** described in Algorithm I.
End for
2. **For** particle $x_t^{V+1} \dots x_t^N$
Generate particles using **Component Set Resampling** described in Algorithm II.
End for

4.5. The Data Assimilation Algorithm

The overall data assimilation procedure for the agent-based simulation of smart environments is performed as follows. During the initialization stage, each particle is initialized to have the same number of agents as the number of occupants in the real system. For each particle, agents’ initial positions and destinations are randomly assigned over the 2D space. After the initialization, the data assimilation is carried out in an iterative manner. In each iteration, for each particle the sampling step uses the state transition function defined in section 4.2 to draw a new sample. This is done by starting from the particle’s state and running the agent-based simulation model for a period of time τ , where τ is the time to the next data assimilation time point (e.g., 20 minutes later). Afterwards particles’ weights are calculated based on the observation model and sensor data as described in section 4.3. Then the particles are re-sampled using the mixed component set resampling as described in section 4.4, and then the next data assimilation iteration starts. This algorithm is summarized as follows:

Algorithm IV: Data Assimilation based on Particle Filter

Data Assimilation based on Particle Filter

1. Initialization. $t = 0$
For particle $i=1, \dots, N$,
randomly assigning agents’ positions and destinations in particle i ;
End For

```

2. Sampling
   For particle  $i = 1, \dots, N$ 
       Start from the particle's state and run the agent-based simulation for a period of time  $\tau$ .
   End For
   For particle  $i = 1, \dots, N$ 
       Calculate particles' weights based on the observation model and sensor data.
   End For
3. Resampling
   Resample with replacement  $N$  particles using the mixed component set resampling
   Set  $t \leftarrow t + 1$  and go to step. 2

```

5. Experiment Results

To evaluate the effectiveness of the proposed method, we conducted a series of experiments with different number of agents and particles. We set up these experiments within the context of occupancy estimation, i.e., utilizing sensor data to estimate occupants' locations in the smart environment. Generally speaking, there are two types of problems associated with occupancy estimation. One is the target tracking problem that tracks each individual. This typically applies to a small number of occupants. Another is to estimate the overall spatial distribution (and density) of the environment without tracking each individuals. This typically applies to a large number of occupants, where tracking each individual is impractical (especially when using low quality sensors such as binary proximity sensors). In this paper, we consider the problem of tracking individual occupants and use a relatively small number of occupants. Our main goal is to demonstrate the developed data assimilation framework and the component set resampling algorithm. We use the identical twin experiment to evaluate our method, which is commonly used in data assimilation research. In the identical twin experiment, we first run a simulation and record the corresponding data. These simulation results are considered as "real", therefore, the observation data obtained here are regarded as coming from the "real" system. Consequently, we estimate the system state from the observation data using particle filters and then check whether these estimated results are close to the "real" results. This paper does not assimilate real world sensor data, which is considered as future work. Since all the experiments are based on simulations, we do not explicitly define the units of distance and time in the following descriptions, and refer to them as distance unit and time unit, respectively.

In all the experiments, the floor structure and sensor deployment are shown in Fig 4(a), where the space is an 800×300 2D space (in distance unit); the blue bold lines represent the walls and yellow concentric circles represent the sensors' locations and detection areas. The diameter of sensor's detection area is set to be 50 distance units. The error rate of the sensors is set to 5%, which means the sensor has 5% probability of reporting 1 when it is supposed to report 0 and vice versa. Agents move at a constant speed of 1.5 (distance unit/time unit). Sensor data are assimilated in every 4 time units. In the initialization stage of the particle filtering algorithm, agents' positions and destinations are distributed uniformly in space (except the locations occupied by walls). Fig 4(b) shows an illustration of data assimilation results of tracking two agents, where the red dots are different estimations of the two agents' locations from all particles, and the two blue dots (inside the many red dots) are the real locations of the two agents. The sensors that report 1 are displayed in blue (the blue concentric circles); otherwise they are displayed in yellow (the yellow concentric circles). The top left sensor is displayed in blue even though no agent is within its detection range at that time. This is due to the 5% error rate of the sensors. As can be seen, the estimations from the majority particles are close to the locations of the real agents. Note that in this research the posterior is represented by the aggregation of all of the particles instead of a single particle with the highest weight.

(a) Floor structure of the smart environment

(b) Result illustration (with 2 agents)

Fig 4. Experiment Setup and Result Illustration

Table 1 summarizes the experiment design and expected result for different experiments, where N_p is the total number of particles, m is the number of agents and Q represents whether the mixed component set resampling is used or not.

Table 1. Experiment Designs

Experiment	Purpose of Experiment	Parameters	Expected Result
Experiment 5.1	Justification of the component set resampling	$N_p=2000$ $m=2, 3, 4$ $Q=false, true$	As the number of agents increases, the number of particles that have the correct combination of all component values decreases when using the standard resampling. The component set resampling alleviates this problem.
Experiment 5.2	Evaluate the robustness of data assimilation using the standard resampling for a single agent	$N_p=800$ $m=1$ $Q=false$	Estimation error (between real position and estimated position) of the single agent reduces as data assimilation proceeds.
Experiment 5.3.1	Evaluate performance of the standard resampling for 2 agents	$N_p=1200, 1600, 2000$ $m=2$ $Q=false$	Compared to Experiment 5.2, the data assimilation performance decreases because two agents need to be estimated. Increasing the number of particles improves the performance.
Experiment 5.3.2	Evaluate performance of the mixed component set resampling for multiple agents	$N_p=1200, 1600, 2000$ $m=2, 4, 6$ $Q=true$	Compared to Experiment 5.3.1, the mixed component set resampling increases the data assimilation performance (reducing estimation errors)

In our work, the system state includes agents' locations and destinations. For simplicity, in the following description we only consider agents' locations when comparing a particles' state and the real system state. In all experiments, we employ a pairing procedure to pair the agents of a particle and the agents in the real system based on the distance between them so that the results can be compared. The pairing procedure is needed because sensors carry no identity information thus we need a way to associate an agent in the particle with the corresponding agent in the real system for comparison purpose. The pairing procedure works in the following way. Given a particle with agents $\langle a_1 a_2 \dots a_m \rangle$ and the real system with agents $\langle b_1 b_2 \dots b_m \rangle$, we first calculate the distances between all possible pairs $\langle a_i, b_j \rangle$ for all $i = 1$ to m and $j=1$ to m . We then sort the pairs $\langle a_i, b_j \rangle$ in ascending order based on the distance between a_i and b_j and store the sorted pairs in a list L . After that, we recursively select the first element $\langle a_x, b_y \rangle$ in L (which is a pairing of a and b that has the smallest distance), store $\langle a_x, b_y \rangle$ in a list T (referred to as the *pairing result list*), and then delete all the pairs in the list L that contains a_x or b_y . This continues until L is empty. The pairing result is stored in list T , where each element of T is a unique pairing of elements in $\langle a_1 a_2 \dots a_m \rangle$ and elements in $\langle b_1 b_2 \dots b_m \rangle$.

5.1. Justification of the Component Set Resampling

This experiment explores the impact of the system state dimension, i.e., number of system state variables that needs to be estimated, on the resampling result to justify and demonstrate the need of component set resampling. In our work, the number of system state variables is proportional to the number of agents. The state variables of each agent are considered as a component of the overall system state. As the number of agents increases, the number of possible combinations of the system state component values increases too. Consequently, more particles are needed in order to sufficiently cover the possible combinations and to effectively estimate the system state. In this experiment, we refer to a particle whose component values match all the component values of the real state as a particle with correct combination. As the number of agents increases, we expect to see the number of particles with correct combinations decreases dramatically when using the standard resampling. Meanwhile, even though a particle may not have the correct combination for all component values, it may have some component values that match

those in the real state. From the whole system state point of view, such a particle is not “good” because it does not completely match the real state. When using the standard resampling, such a particle is likely to be eliminated because of its low weight. However, the particle still has “good” components that carry useful information and can be utilized, which gives rise to the component set resampling. This experiment aims to demonstrate this aspect.

We use two different methods to measure the degree of match between a particle’s state and the real system state. In both methods, we use the pairing procedure described above to pair the agent in the real simulation and the agent in the particle. After the pairing procedure, for each pair of agents we calculate the location difference (i.e., the distance) between the two agents’ locations. If the distance between the two agents is small (smaller than a threshold; in our experiment, since the diameter of sensor’s detection area is 50 distance units, we set this threshold to be 50), we consider it as a *correct match*. We name the first method of measuring the degree of match as the “complete match” method. In this method, we count how many particles whose states completely match the real state, which means all the pairs in T have the correct match. Formally, let $\langle a, b \rangle$ represent a pair in a pairing result list T , l_a and l_b represent the location of agent a and agent b for the pair $\langle a, b \rangle$, $dist(l_a, l_b)$ is the function returning the distance between l_a and l_b , N is the number of particles, P is the set of all particles, T^i is the pairing result list between particle p^i and the real state. Then the measurement of complete match $E_{complete}$ is calculated as follows:

$$E_{complete} = \sum_{p^i \in P} g(T^i), \text{ where}$$

$$g(T^i) = \begin{cases} 1 & \text{if } \forall \langle a, b \rangle \in T^i, dist(l_a, l_b) < 50 \\ 0 & \text{otherwise} \end{cases}$$

We name the second method of measuring the degree of match as the “percentage match” method. In this method, we calculate the ratio of “good” components in the overall system state for each particle (which means the ratio of pairs in T that has the correct match), and then compute the average for all particles. Following the above notation, and also let m be the total number of agents, the measurement of percentage match $E_{percentage}$ is calculated as follows:

$$E_{percentage} = \frac{\sum_{p^i \in P} (\sum_{\langle a, b \rangle \in T^i} h(a, b)) / m}{N}, \text{ where}$$

$$h(a, b) = \begin{cases} 1 & \text{if } dist(l_a, l_b) < 50 \\ 0 & \text{otherwise} \end{cases}$$

The two methods measure the degree that the particles match the real state at different granularity levels. The complete match measurement represents the match at the whole state level – a particle is a complete match only when its whole state matches the real state. The percentage match measurement represents the match at the component (sub-state) level – a particle contributes to the percentage as long as it has one or more “good” components.

Fig 5. The route configurations

In this set of experiments, we utilize 2000 particles and the total simulation time is 350. Since data are assimilated in every 4 time units, there is a total of 88 data assimilation steps. The agents follow the route shown in fig 5 (the route shown in Fig 5(d) is not used in this experiment; it is used in the experiment described in Section 5.3.2). We vary the number of agents from 2, to 3 and then to 4, and use both the standard resampling and the mixed component set resampling in all the cases. Note that when there is only one agent, the number of components in the system state is 1. In this case, the mixed component set resampling performs in the same way as the standard resampling. Thus we do not perform experiments using the mixed component set resampling for the single agent case. Each experiment is repeated 5 times and the average result is presented.

Fig. 6 shows the results of complete match and percentage match based on the measurement methods described above. The results are calculated by averaging the results at each data assimilation step. Fig 6(a) shows the complete match result, where the y axis represents the average number of particles that completely match the real state and the x axis shows the three different cases with different number of agents. Fig 6(b) shows the percentage match result, where the y axis represents the average ratio of “good” components and the x axis represents the three cases. Fig 6(a) shows that for both the standard resampling and the mixed component set resampling, the number of complete matches decreases as the number of agents increases. This is expected because as the number of agents increases, it is more difficult for particles to have the correct combinations of system state. Fig 6(a) also shows that for the standard resampling the number of complete matches decreases dramatically as the number of agents increases. When there are 4 agents, there is zero number of complete match for the standard resampling. Nevertheless, from Fig 6(b) one can see that even though there is zero complete match in the case of 4 agents, the standard resampling still has a relative high ratio of “good” components (about 37%). The existence of such “good” components is one of the major motivations for us to develop the component set resampling, which divides the state space into sub-components and reconstruct particles at the component level. By doing this, the good “components” of all particles can be utilized for constructing new particles, and thus leading to improved results. The effect of the mixed component set resampling is demonstrated in both Fig 6(a) and Fig 6(b). In Fig 6(a), it is shown that the mixed component set resampling leads to more complete matches in all the three cases of 2, 3 and 4 agents. Even for the 4-agent case, the number of complete matches reaches as high as 160 using the mixed component set resampling (compared to 0 for the standard resampling). In Fig 6(b), the ratio of correct match of the components increases too when using the mixed component set resampling. These results justify the need and demonstrate the effect of the component set resampling.

Fig 6. Experiment results averaged from all data assimilation steps

Fig 7 provides more details about the percentage match of both resampling methods by showing the value of percentage match over time, where the x axis represents the data assimilation steps. As can be seen, in both Fig 7(a) and Fig 7(b), the percentage match value increases over time. This indicates as a general trend the particles gradually converge to the posterior of the true state. In both the two figures, there are oscillations due to the fact that sensors are sparsely deployed: the peaks correspond to situations when the real agents move into the sensor detection areas; the valleys correspond to situations when the real agents move out of the sensor detection areas. Furthermore, the figures show that as the number of agents increases, the percentage match values decrease. with the same results are seen in Fig 6. Comparing Fig 7(a) and Fig 7(b), we can see that the mixed component set resampling has better performance than the the standard resampling, because the percentage match values of the mixed component set resampling are higher than those of the standard resampling in all the three cases.

Fig 7. Percentage match results over time

5.2. Data Assimilation for Single Agent

This set of experiments assimilate sensor data to estimate the position of a single moving agent. The goal is to show that the state estimation converges to the “true” value, that is, the error between the true location and the estimated location of the agent remains below a certain threshold as the data assimilation proceeds. In this experiment, we use 2000 particles, and the total simulation time is 1000. Since sensor data are assimilated in every 4 time units, the total number of data assimilation steps is 250.

To evaluate the performance of the data assimilation, in each step of the data assimilation we calculate the error between the agent’s real location and the estimated locations from particles, and use the averaged result from all particles in each data assimilation step to measure the estimation accuracy. Formally, the averaged location error at time t is calculated as below:

$$E(t) = \frac{\sum_{p^i \in P} \text{dist}(l_t^i, l_t^{\text{real}})}{N}$$

, where N is the number of particles, N_p is the set of all particles, l_t^i is the location of the agent in particle p^i at time t , l_t^{real} is the location of the agent in the real system at time t , and $\text{dist}(l_t^i, l_t^{\text{real}})$ is the function returning the distance between l_t^i and l_t^{real} .

We evaluate the data assimilation for two cases where the agent has different moving patterns. In the first case, the real agent moves forward without turning back. The route configuration is shown in Fig 8(a). In the second case, the real agent moves forward and then turns back. The route configuration for the second case is shown in Fig 8(b). The two cases share the same initial location for the agents. Fig 9 shows the estimation errors of the two experiments, where the result at each data assimilation step is calculated by averaging results from 10 experiment runs.

Fig 8. Configurations of the routes

Fig 9. Estimation error for the move-forward route and turn-back route

In Fig 9, the vertical axis represents the estimation error and the horizontal axis represents the data assimilation steps. In both cases, there are oscillations for the estimation errors. This is due to the sparse deployment of sensors as explained in Section 5.1. As a general trend, when time increases the estimation error decreases and eventually maintains at a low level. This is expected and shows that the particle filter algorithm is working. Note that for the move-forward case, the estimation error slightly increases after step 190 because the agent moves across several 3-way intersections at the end of the route. These 3-way intersections result in more alternatives for the agent's movement, and thus make it more difficult for particles to converge to the "true" state of the agent. Fig 9 also shows that in the turn-back route configuration, the estimation error increases considerably between step 154 and 208. This is because the agent is turning to an opposite direction and is staying in an area where no sensor is deployed. After the agent enters the detection area of sensors, the error decreases and then maintains at a low level.

5.3. Data Assimilation for Multiple-Agent

5.3.1. Data Assimilation for Multiple-Agent using the Standard Resampling

This set of experiments investigate the impact of number of particles on the estimation result. We carry out these experiments using the standard resampling for two agents. The routes of the two agents are set as Fig 5(a). In general, we expect to see improved data assimilation result when the number of particles increases. In other words, the estimation error decreases as the number of particles increases. The estimation error is calculated as follows:

$$E(t) = \frac{\sum_{k=1}^N \frac{1}{m} \sum_{j=1}^m \text{dist}(l_t^{(j)k}, l_t^{(j)real})}{N}$$

, where N represents the number of particles and m represents the number of agents in the experiments; $l_t^{(j)k}$ represents the location of agent j in particle k at time t and $l_t^{(j)real}$ represents the location of the corresponding pairing agent in the real system at time t . In all the experiments, the total simulation time is 350, and there are 88 data assimilation steps. The number of agent m is 2, and the number of particles N varies from 1200, to 1600 and then to 2000. Fig. 10 shows the experiment results that are averaged from 20 experiment runs.

Fig 10. Estimation error using different number of particles

In Fig 10(a), the vertical axis represents the estimation error by averaging the error for all data assimilation steps. The horizontal axis represents the number of particles used in data assimilation. It shows that as the number of particles increases, the average estimation error decreases. Fig 10(b) shows the estimation errors over time for the three cases. In general, it can be seen that as the number of particles increases, the estimation error per time step decreases. This is because when the number of particles increases, they can more effectively cover the state space and thus it is more likely that there exist samples whose states are close to the “true” system state. Below we show the performance of the mixed component set resampling and compare it with the standard resampling.

5.3.2. Data Assimilation for Multiple-Agent using the Mixed Component Set Resampling

In this section, we conduct experiments using the mixed component set resampling and compare their results with the standard resampling. The purpose is to demonstrate the advantage of the mixed component set resampling over the standard resampling when the system state has multiple components (that is, there are multiple agents). We first consider the case of 2 agents. The experiment configurations and measurements are the same as in Section 5.3.1. We vary the number of particles from 1200, to 1600 and then to 2000, and show their estimation errors in Fig 11(a), 11(b), and 11(c), respectively.

Fig 11. Comparison of component set and standard resampling algorithm

In Fig 11, the vertical axis represents the estimation errors and the horizontal axis represents data assimilation steps. Fig 11 shows that the estimation errors of the mixed component set resampling are lower than those of the standard resampling in all three cases of using the different number of particles. This confirms the advantage of the component set resampling.

To further show the advantage of the component set resampling, we conduct two other experiments where 4 and 6 agents are used, respectively. The routes for the 4 agents and 6 agents are shown in Fig 5(c) and Fig 5(d) respectively. We use 1200 particles and employ both the standard resampling and the mix component set resampling and compare their results. Fig. 12 shows the results that are averaged from 20 experiment runs. It can be seen that the mixed component set resampling has lower estimation error than the standard resampling in both cases. These results confirm that the mixed component set resampling increases the performance of particle filter-based data assimilation for multiple agents. We note that in this work we use binary proximity sensors, which provide no

identity information of agents and thus it is impossible to distinguish which agent (or multiple agents) triggers which sensor data. As the number of agents increases, it is more difficult for particles to converge to the “true” system state, and even if they converge, they may not be able to maintain the convergence due to the dynamics of the system and the limited information provided by the sensors. In the 6-agent case as shown in Fig 12(b), the estimation error decreases to a relatively low level in the beginning and then increases towards the end. This is because the agents were separated from each other in the beginning, and then after data assimilation step 65, several agents move together across a crowded area (in the middle of the map), making it difficult to distinguish the agents and their moving directions based on the binary proximity sensor data. As the number of agents increase further, estimating the state of each individual agent using 1600 particles becomes even more difficult. See the next section for some discussions on this.

Fig 12. Comparison of component set and standard resampling algorithm

6. Discussions

In our experiments, we use a sparsely populated environment with small number of agents to demonstrate the data assimilation framework. Due to the limited information provided by the binary proximity sensors (especially that the sensors provide no identity information about agents), tracking each individual for a large number of agents in a crowded environment is infeasible. In fact, as the number of agents increases, the nature of the occupancy estimation problem changes from tracking each individual to estimating occupants’ overall spatial distribution (and density) in the environment. Advanced techniques such as clustering agents to help counting the number of agents and distinguishing different agents (see e.g., [31, 32]) can be incorporated to improve the tracking results. This paper does not focus on the tracking problem itself because its main focus is on the data assimilation framework and the component set resampling method.

Incorporating real time data in modeling and simulation has been recognized as an important topic in computer modeling and simulation. There could be multiple ways to utilize real time sensor data for serving different purposes. In this work, real time sensor data is assimilated into a simulation model to provide estimations of the real system state. The estimations of system state are then used to provide initial conditions for running simulations to more accurately simulate/predict the future dynamics of the system. To achieve this goal, we frame the problem as a state estimation problem from observation data and adopt the same principles of data assimilation that have achieved fruitful results in other research fields such as geosciences and meteorology. It is important to note that our work is different from the work of model calibration, which aims to calibrate a simulation model to match the data of a real system. However, using data assimilation it is possible to treat some of the model parameters as state variables and then dynamically estimate/calibrate those parameters based on observation data (see our previous work [34] as an example).

This paper presents data assimilation within the context of agent-based simulation of smart environments, where real time sensor data are available. The data assimilation framework and methods are all developed based on this application context. Nevertheless, we note that the principles of the developed framework can be adapted to other agent-based simulation applications, such as agent-based traffic simulation, where real time sensor data are available. This is because the data assimilation is framed as a general state estimation problem, where the state transition function is defined based on the agent-based simulation model. As sensors are increasingly used in various applications, the topic of how to assimilate real time sensor data into simulations becomes more and more important.

This work represents one of the first efforts in data assimilation for agent-based simulations. It is our hope that it would inspire more works from others to make progress in this field.

7. Conclusions

In this paper, we develop a particle filter-based data assimilation framework for agent-based simulation of smart environments. An agent-based simulation model is described and then the data assimilation framework is presented. To improve data assimilation for multiple agents, we develop a novel resampling method called component set resampling that breaks the overall system state into sub-components and then re-sample particles at the component level. This resampling method increases the diversity of the samples and therefore can represent a larger state space using the same number of particles. Several experiments are carried out to demonstrate the data assimilation framework, and results show that the component set resampling has better performance than the standard resampling in data assimilation for multiple agents. Future work will include developing methods to support occupancy estimation involving large number agents, carrying our experiments using real world sensor data, and evaluating the data assimilation framework using more agent-based simulation models.

Acknowledgements

This work is partially supported by NSF grants CNS-0841170 and CNS-0941432.

References

- [1] M. Altaweel, L. Alessa, A. Kliskey, C. Bone, A framework to structure agent-based modeling data for social-ecological systems, in: Struct. Dynam. eJ. Anthro. Rel. Sci. 2010.
- [2] R. Aylett, M. Cavazza, Intelligent Virtual Environments: a State-of-the-Art Report, in: In: Eurographics 2001, STAR Reports volume. (2001), 87-109.
- [3] O. Baqueiro, Y. Wang, P. McBurney, F. Coenen, Integrating Data Mining and Agent Based Modeling and Simulation, in: P. Perner (Ed.) Advances in Data Mining. Applications and Theoretical Aspects. (Springer Berlin Heidelberg, 2009), 220-231.
- [4] T. Bengtsson, P. Bickel, B. Li, Curse of dimensionality revisited: Collapse of the particle filter in very large scale systems, in: D. Nolan, T. Speed (Eds.) IMS Lecture Notes - Monograph Series in Probability and Statistics: Essays in Honor of David A. Freedman. Institute of Mathematical Sciences, 2008, 316-334.
- [5] D.G. Brown, R. Riolo, D.T. Robinson, M. North, W. R, Spatial process and data models: Toward integration of agent-based models and GIS, Journal of Geographical Systems, (2005) 25-47.
- [6] D.J. Carter, B. Whitehead, A study of pedestrian movement in a multi-storey office building, Building and Environment, 11 (1976) 239-247.
- [7] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math., 1 (1959) 269-271.
- [8] A. Doucet, N. De Freitas, N. Gordon, Sequential Monte Carlo method in practice, (Springer, New York, 2001).
- [9] D. Fox, KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization, in: In Advances in Neural Information Processing Systems. 2001.
- [10] W.R. Gilks, C. Berzuini, Following a moving target—Monte Carlo inference for dynamic Bayesian models, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63 (2001) 127-146.
- [11] S. Hassan, L. Antunes, J. Pavon, N. Gilbert, Stepping on earth: A roadmap for data-driven agent-based modelling, in: The Centre for Research in Social Simulation. Paper 5. (2008).
- [12] S. Hassan, J. Pavón, L. Antunes, N. Gilbert, Injecting Data into Agent-Based Simulation, in: K. Takadama, C. Cioffi-Revilla, G. Deffuant (Eds.) Simulating Interacting Agents and Social Phenomena. (Springer Japan, 2010), 177-191.
- [13] C. Liao, P. Barooah, An integrated approach to occupancy modeling and estimation in commercial buildings, in: American Control Conference (ACC), (2010), 3130-3135.
- [14] S. Meyn, A. Surana, Y. Lin, S.M. Oggianu, S. Narayanan, T.A. Frewen, A sensor-utility-network method for estimation of occupancy in buildings, in: Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on. (2009), 1494-1500.
- [15] G.W. Pulford, Taxonomy of multiple target tracking methods, Radar, Sonar and Navigation, IEE Proceedings -, 152 (2005) 291-304.

- [16] S. Särkkä, A. Vehtari, J. Lampinen, Rao-Blackwellized particle filter for multiple target tracking, *Information Fusion*, 8 (2007) 2-15.
- [17] N. Shrivastava, Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms, in: *SenSys '06: Proc. 4th Internat. Conf. on Embedded Networked Sensor Systems*. (ACM press, 2006), 251-264.
- [18] J. Singh, U. Madhow, R. Kumar, S. Suri, R. Cagley, Tracking Multiple Targets Using Binary Proximity Sensors, in: *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*. (2007), 529-538.
- [19] R. Tomastik, S. Narayanan, A. Banaszuk, S. Meyn, Model-Based Real-Time Estimation of Building Occupancy During Emergency Egress, in: W.W.F. Klingsch, C. Rogsch, A. Schadschneider, M. Schreckenberg (Eds.) *Pedestrian and Evacuation Dynamics 2008*. (Springer Berlin Heidelberg, 2010), 215-224.
- [20] F. Klügl, G. Rindsfuser, Large-Scale Agent-Based Pedestrian Simulation, *Multiagent System Technologies*(2007) pp145-156.
- [21] L. Yang, N. Gilbert, Getting away from numbers: Using qualitative observation for agent-based modelling, in: F. Amblard (Ed.) *ESSA'07: Fourth Conference of the European Social Simulation Association*. (Toulouse, France, 2007), 205-214.
- [22] H. Aydt, M. H. Lees, S. J. Turner, W. Cai, Toward Simulation-based Egress Optimization in Smart Buildings using Symbiotic Simulation, in: *PED 2012: 6th International Conference on Pedestrian and Evacuation Dynamics*(2012), 987-999.
- [23] B. Heath, R. Hill, F. Ciarallo, A Survey of Agent-Based Modeling Practices (January 1998 to July 2008), *Journal of Artificial Societies and Social Simulation* (2009) 12(4)9
- [24] J. Arroyo, S. Hassan, C. Gutierrez, J. Pavon, Re-thinking simulation: a methodological approach for the application of data mining in agent-based modelling, *Computational and Mathematical Organization Theory* (2010) 16(4) 416-435
- [25] P. Windrum, G. Fagiolo, A. Moneta, Empirical Validation of Agent-Based Models: Alternatives and Prospects, *Journal of Artificial Societies and Social Simulation* (2007) 12(2) 8.
- [26] S. Hassan, J. Arroyo, J. M. Galan, L. Antunes, J. Pavon, Asking the Oracle: Introducing Forecasting Principles into Agent-Based Modelling, *Journal of Artificial Societies and Social Simulation* (2013), 16(3) 13.
- [27] Z. Chen Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics* (2003) 182 (1) pp 1-69
- [28] C. Snyder, Bengtsson T, Bickel P, Anderson J. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*(2008), vol. 136, no. 12, pp.4629 – 4640.
- [29] F. Bouttier, and P. Courtier. 1999. Data assimilation concepts and methods. Edited by Meteorological Training Course Lecture Series. Reading, England: European Centre for Medium-Range Weather Forecasts.
- [30] R. H. Reichle, 2008. Data Assimilation Methods in The Earth Sciences. *Advances in Water Resources*, vol. 31, no. 11, pp.1411-1418, 2008.
- [31] J. Singh; Kumar, R.; Madhow, U.; Suri, S.; Cagley, R. Multiple-target tracking with binary proximity sensors. *ACM Trans. Sen. Netw.* 2011.
- [32] L. Song; Wang, Y. Multiple target counting and tracking using binary proximity sensors: Bounds, coloring, and filter. In *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Philadelphia, PA, USA, 11–14 August 2014; pp. 397–406.
- [33] N.J. Gordon; Salmond, D.J.; Smith, A.F.M., 1993. Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *Radar and Signal Processing, IEEE Proceedings F*, vol. 140, no. 2, 107 – 113,1993.
- [34] F. Bai, S. Guo, X. Hu, Towards parameter estimation in wildfire spread simulation based on Sequential Monte Carlo Methods, *Proc. 44th Annual Simulation Symposium (ANSS), 2011 Spring Simulation Multiconference (SpringSim'11)*, pp. 159-166, 2011