Mixed-Initiative Coordination for Disaster Response in the Real-World

Paper XXX

ABSTRACT

The problem of allocating emergency responders to rescue tasks is a key application area for agent-based coordination algorithms. However, to date, none of the proposed approaches take into account the uncertainty predominant in disaster scenarios and, crucially, none have been deployed in the real-world in order to understand how humans perform when instructed by an agent. Hence, in this paper, we propose a novel algorithm, using Multi-agent Markov Decision Processes to coordinate emergency responders. More importantly, we deploy this algorithm in a mixed-reality game to help an agent guide human players to complete rescue tasks. In our field trials, our algorithm is shown to improve human performance and our results allow us to elucidate some of the key challenges faced when deploying of mixed-initiative team formation algorithms.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Multi-Agent Systems

General Terms

Design, Human Factors, Algorithms

Keywords

Human-Agent Interaction, Coordination, Decision under Uncertainty, Adjustable Autonomy

1. INTRODUCTION

The coordination of teams of field responders in search and rescue missions is regarded as one of the grand challenges for multi-agent systems research [**?**]. In such settings, responders with different capabilities (e.g., fire extinguishing, digging, or life support) have to form teams in order to perform rescue tasks (e.g., extinguishing a fire or digging civilians out of rubble or both) to minimise costs (e.g., time or money) and maximise the number of lives and buildings saved. Thus, responders have to plan their paths to the tasks (as these may be distributed in space) and form specific teams to complete some tasks. These teams, in turn, may need to disband and reform other teams to complete other tasks requiring different capabilities, taking into account the status (e.g., health or building fire) of the tasks and the environment (e.g., if a fire or radioactive cloud is spreading). Furthermore, uncertainty in the environment (e.g., wind direction or speed) or in the responders' abilities to complete tasks (e.g., some may be tired or get hurt) means that plans may need to change depending on the state of the players and the environment.

To address these challenges, in recent years, a number of algorithms and mechanisms have been developed to create teams and allocate tasks. For example, [**?**, **?**, **?**] and [**?**, **?**], developed centralised and decentralised optimisation algorithms respectively to allocate rescue tasks efficiently to teams of field responders with different capabilities. However, none of these approaches considered the uncertainty in the environment or in the field responders' abilities. Crucially, to date, while all of these algorithms have been shown to perform well in simulations (assuming agents as computational entities), none of them have been *deployed* to guide *real* human responders (amateur or expert) in real-time rescue missions. Thus, it is still unclear whether these algorithms will cope with real-world uncertainties (e.g., communication breakdowns or change in wind direction), be acceptable to humans (i.e., agent-computed plans are not confusing and take into account human capabilities), and do help humans perform better than on their own.

Against this background, in this paper we develop a novel algorithm for team coordination under uncertainty and evaluate it within a real-world mixed-reality game that embodies the simulation of team coordination in disaster response settings. In more detail, we consider scenario involving rescue tasks distributed in a disaster space over which a radioactive cloud is spreading. Tasks need to be completed by the responders before the area is completely covered by the cloud (as responders will die from radiation exposure) which is spreading according to varying wind speed and direction. Our algorithm captures the uncertainty in the scenario (i.e., in terms of environment and player states) and is able to compute a policy to allocate responders to tasks to minimise the time to complete all tasks without them being exposed to significant radiation. The algorithm is then used by an agent to guide human responders based on their perceived states. This agent is then implemented in our deployed platform, AtomicOrchid, that structures the interaction between human responders, a human coordinator, and the agent in a mixed-reality location-based game. By so doing, we are able to study, both quantitatively and qualitatively, the perfor-

Appears in: *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), Lomuscio, Scerri, Bazzan, Huhns (eds.), May, 5–9, 2014, Paris, France.*

mance of a mixed-initiative team (i.e., a human team under human and agent guidance) and the interactions between the different actors in the system. Thus, this paper advances the state of the art in the following ways:

1. We develop a novel approximate algorithm for team formation under uncertainty using a Multi-agent Markov Decision Process (MMDP) paradigm, and show how it accounts for real-world uncertainties.

2. We present AtomicOrchid, a novel platform to evaluate team formation under uncertainty using the concept of mixed-reality games. AtomicOrchid allows an agent, using our team formation algorithm, to coordinate, in real-time, human players using mobile phone-based messaging, to complete rescue tasks efficiently.

3. We provide the first real-world evaluation of a team formation agent in a disaster response setting in field trials and present both quantitative and qualitative results. Our results allow us to elucidate some of the challenges for the formation of human-agent collectives, that is, mixed-initiative teams where control can be passed between agents and humans in flexible ways.

When taken together, our results show, for the first time, how agent-based coordination algorithms for disaster response can be validated in the real-world. Moreover, these results allow us to derive a methodology and guidelines to evaluate human-agent interaction in real-world settings.

The rest of this paper is structured as follows. Section **??** formalises the disaster response problem as an MMDP. Section **??** then describes the algorithm to solve the path planning and task allocation problems presented by the MMDP while Section **??** describes the AtomicOrchid platform. Section **??** presents our pilot study and the evaluation of the system in a number of field trials. Finally, Section **??** concludes.

2. THE DISASTER SCENARIO

We consider a disaster scenario involving a satellite, powered radioactive fuel, that has crashed in a sub-urban area (see Section **??** to see how this helps implement a mixed-reality game). While debris is strewn around a large area, damaging buildings and causing accidents and injuring civilians, radioactive particles discharged in the air, from the debris, are gradually spreading over the area, threatening to contaminate food reserves and people. Hence, emergency services, voluntary organisations, and the military are deployed to help evacuate the casualties and resources before these are engulfed by a radioactive cloud. In what follows, we model this scenario formally and then describe the optimisation problem faced by the actors (i.e., including emergency services, volunteers, medics, and soldiers) in trying to save as many lives and resources as possible. We then propose an algorithm to solve this optimisation problem (in Section **??**). In Section **??**, we demonstrate how this algorithm can be used by a software agent (in our mixed-reality game) in a mixed-initiative process to coordinate field responders.

2.1 Formal Model

Let G denote a grid overlaid on top of the disaster space, and the satellite and actors are located at various coordinates $(x, y) \in G$ in this grid. The radioactive cloud induces a radioactivity level $l \in [0, 100]$ at every point it covers in the grid (100 corresponds to maximum radiation). While the exact radiation levels can be measured by responders on the ground (at every grid coordinate) using their geiger counter, it is assumed that some information is available from existing sensors in the area. However, this information is uncertain due to the poor positioning of the sensors and the variations in wind speed and direction (and we show how this uncertainty is captured in the next section). A number of safe zones $G' \subseteq G$ are defined where the responders can drop off assets and casualties. Let the set of field responders be denoted as $i_1, \cdots, i_n \in I$ and the set of rescue tasks as $t_1, \cdots, t_m \in T$. As responders enact tasks, they may become tired, get injured, or receive radiation doses that may, at worst, be life threatening. Hence, we assign each responder a health level $h_i \in [0, 100]$ that decreases based on their radiation dose and assume that their decision to perform the task allocated to them is liable to some uncertainty (e.g., they may not want to do a task because they are tired or don't believe it is the right one to do). Moreover, each responder has a specific role $r \in Roles$ (e.g., fire brigade, soldier, or medic) and this will determine the capabilities he or she has and therefore the tasks he or she can perform. We denote as $Roles(i)$ the role of responder i. In turn, to complete a given task t, a set of responders $I' \subseteq I$ with specific roles $R_t \subseteq R$ is required. Thus, a task can only be completed by a team of responders I' if $\{Roles(i)|i \in I'\} = R_t$. In our

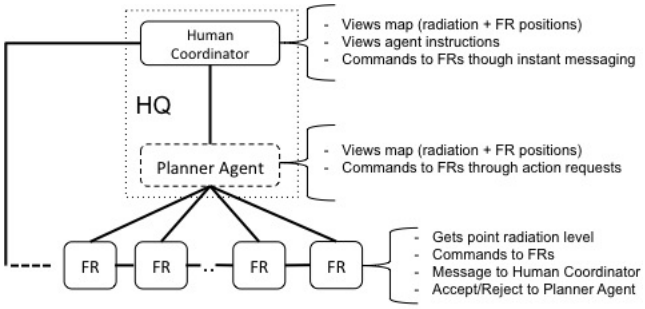


Figure 1: The interactions between different actors in the disaster scenario. Lines represent communication links. Planner agent and coordinator sit in the headquarters (HQ). Field responders (FRs) can communicate with all actors directly.

scenario, we assume that the field responders are coordinated by the headquarters headed by a human coordinator H but assisted by a planner agent PA that can receive input from the human coordinator or the field responders. While the human coordinator H communicates its instructions directly to the responders (e.g., using an instant messaging client or walkie talkie), the planning agent PA can compute an allocation of tasks for the responders to complete. This is communicated to them in terms of simple "Do task X at position Y". The responders may not want to do some tasks (for reasons outlined above) and may therefore simply accept or reject the received instruction. These interactions are depicted in Figure 2.1.

It is important to note that our model implements different types of control: (i) agent-based: when the agent instructs the responders (ii) human-based: when respon-

ders work with the coordinator or between themselves. Our model also captures different modes of control: (i) centralised: when responders respond to the planning agent or human coordinator (ii) decentralised: if responders coordinate between themselves. Crucially, this scenario allows for flexible levels of human and agent autonomy. For example, field responders may simply implement the plan given to them by the planner agent but can also feedback their constraints to the planner agent (as we demonstrate later) by rejecting some instructions and requesting new instructions.

Given this model, we next formulate the optimisation problem faced by the responders and solved by the planning agent (later in Section **??**). To this end, we propose a Multi-Agent Markov Decision Process (MMDP) [**?**] that captures the uncertainties of the radioactive cloud and the responders' behaviours. Specifically, we model the spreading of the radiative cloud as a random process over the disaster space and allow the actions requested from the responders to fail (because they refuse to go to a task) or incur delays (because they are too slow) during the rescue process. This stands in contrast to previous work [**?**, **?**] that require the process of task executions to be deterministic and explicitly model the task deadlines as deterministic constraints (which are stochastic in our domain). Thus in the MMDP model, we represent task executions as stochastic processes of state transitions. Thus, the uncertainties of the radioactive cloud and the responders' behaviours can be easily captured with transition probabilities. Additionally, modelling the problem as a MMDP allows us to use many sophisticated algorithms such as MCTS [**?**] and RTDP [**?**] that have already been developed in the literature.

2.2 The Optimisation Problem

A Multi-agent Markov Decision Process (MMDP) is formally defined as a tuple $\langle I, S, \{A_i\}, P, R\rangle$, where:

- I is a set of n responders. Each responder is associated with a unique identifier number $i \in I$.
- $S = S_r \times S_{p_1} \times \cdots \times S_{p_n} \times S_{t_1} \times \cdots \times S_{t_m}$ is the state space. S_r is the state variable of the radiative cloud to specify the radiative level (between 0 and 100) of each grid. S_{p_i} is the state variable for responder i to specify his current health level and location. S_{t_j} is the state variable for task j to specify its status (picked up, dropped off, or idle) and location.
- A_i is a set of responder i's actions. Each responder can stay in his current grid, move to his 8 neighboring grids (north, northeast, east, southeast, south, southwest, west, northwest), or pickup/drop a task located in his current grid. A joint action is a list of actions, $\vec{a} = \langle a_1, \cdots, a_n\rangle$, one for each responder.
- $P = P_r \times P_{p_1} \times P_{p_n} \times P_{t_1} \times P_{t_n}$ is the transition function. $P_r(s'_r|s_r)$ is the probability for the radiative cloud to expand from state variable s_r to s'_r. $P_{p_i}(s'_{p_i}|s_{p_i}, a_i)$ is the probability for responder i to transit from state variable s_{p_i} to s'_{p_i} when executing action a_i (e.g., when a responder moves to north, his health level and location will be updated based on his previous health level and location). $P_{t_j}(s'_{t_j}|s_{t_j}, \vec{a})$ is the transition probability for task j and a task transits to a new state if and only if all the necessarily skilled responders are located in the same grid as the task and perform the "pickup/drop" actions at the same time. This implicitly model the requirements of tasks for teams of skilled responders.
- R is the reward function. If a task has been dropped off on any dropoff zone, a big reward is received. A huge penalty is given if a responder is killed. Each responder's action is associated with a small cost.

Algorithm 1: Team Coordination

Input: the MMDP model and the current state s.
Output: the best joint action $\vec{a}$.
`// The task planning`
$\{t_i\} \leftarrow$ compute the best task for each responder $i \in I$
foreach $i \in I$ **do**
  `// The path planning`
  $a_i \leftarrow$ compute the best path to task t_i
return $\vec{a}$

At each time step of the game, we assume the planning agent is fully observable of the current state. Thus, a plan (a.k.a policy) for the team is a mapping from states to joint actions, $\pi : S \rightarrow \vec{A}$. By given a plan, the responders know how to act in the field. The expected value of a plan π can be computed recursively by the Bellman equation:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s') \qquad (1)$$

where $\pi(s)$ is a joint action selected by the plan and $\gamma \in (0, 1]$ is the discount factor. The goal of solving the game is to find an optimal plan π^* that maximize the expected value given the initial state s^0, $\pi^* = \arg\max_{\pi} V^{\pi}(s^0)$.

3. TEAM COORDINATION ALGORITHM

Given a reasonable size of the game, the MMDP model can be huge. For example, with 8 players and 17 tasks in a 50×55 grid, the number of possible states is more than 2×10^{400}. Thus, it is computational intractable to find the optimal plan of the game. One useful observation is that, when making a decision, each player first selects a task to form a team and then move to the task location independently. In our planning algorithm, we use this hierarchical structure to decompose the decision making process into two level:

- In the higher level, task planning algorithm is run for the whole team to assign the best task to each players given the current state.
- In the lower level, by given a task, path planning algorithm is run for each player to find the best path to the task from his current location.

Given the huge state space, it is intractable to find the plan for all states offline. Therefore, we only compute plans for the queried states online. The main process of the online hierarchical planning algorithm is outlined in Algorithm 1. The following subsections will describe the algorithms in each level in more detail.

3.1 Task planning

As mentioned above, each player in the game owns a type of skill and each task requires players with a certain combination of the skills. The goal is to assign a task to each

player that maximize the overall performance given the current state. To do that, we first compute all possible coalitions $\{C_{jk}\}$ for each task j where a coalition $C_{jk} \subseteq I$ is a subset of the players with the required skills. Then, we solve the following optimization problem to find the best coalitions:

$$\begin{array}{ll} \max_{x_{jk}} & \sum_{j,k} x_{jk} \cdot v(C_{jk}) \\ \text{s.t.} & x_{jk} \in \{0,1\} \\ & \forall j, \sum_k x_{jk} \leq 1 \quad \text{(i)} \\ & \forall i, \sum_{j,k} \delta_i(C_{jk}) \leq 1 \quad \text{(ii)} \end{array} \tag{2}$$

where x_{jk} is the boolean variable to decide whether to select coalition C_{jk} for task j or not, $v(C_{jk})$ is the characteristic function for coalition C_{jk}, and $\delta_i(C_{jk}) = 1$ if $i \in C_{jk}$ and 0 otherwise. Constraint (i) ensures that a task j is allocated at most to only one coalition (a task does not need more than one group of players). Constraint (ii) ensures that a player i is assign to only one task (a player cannot do more than one task at the same time). In the optimization, we only consider the tasks that have not been done and the players that are still alive. Because the players do not change their skills and the requirements of the tasks are static during the game, the set of all possible coalitions for each task can be computed offline before the game.

The key challenge of the optimisation problem in Equation 2 is to compute the value of the characteristic function $v(C_{jk})$ for each coalition C_{jk}. This is the expected value when the players in C_{jk} are assigned to task j. In order to compute this value, we need to know the plan after the completion of task j because not all tasks can be completed in one shot. As aforementioned, computing the optimal plan is intractable given the huge state space. Thus, we estimate the value by simulations. The main process is outlined in Algorithm 2.

In each simulation, we first assign the responders in C_{jk} for task k and run the simulator for this assignment starting from the current state s. After task k is completed, we collect the reward r and the new state s'. Then, we estimate the expected value of s' by Monte-Carlo Tree Search (MCTS), which has been shown to be efficient for many large problems [**?**]. The basic idea of MCTS is to maintain a search tree where each node is associated with a state and each branch is an assignment for the responders.

In the task planning level, "completing a task by a responder" is treated as a macro action, assuming that each responder can find his or her best way to the task and back to the closest dropoff zone (computed by Section 3.2). Thus, the main step of MCTS is to compute an assignment for the free responders at each node of the search tree. We use Equation 2 to compute the assignment and use the UCT heuristic [**?**] to estimate the coalition value:

$$v(C_{jk}) = \overline{v(C_{jk})} + c\sqrt{\frac{2N(s)}{N(s, C_{jk})}} \tag{3}$$

where $\overline{v(C_{jk})}$ is the averaged value of coalition C_{jk} at state s, c is a constant, $N(s)$ is the visiting frequency of state s, and $N(s, C_{jk})$ is the frequency that coalition C_{jk} has been selected at state s. Intuitively, if a coalition C_{jk} has large averaged value or is rarely selected, it gets high chance to be chosen in the next visit of the node. Once a coalition C_{jk} has been selected by the heuristic, the responders in C_{jk} will be assigned to the corresponding task, i.e., task k. The constant c is a tradeoff of the exploitation and exploration for the UCT heuristic.

Algorithm 2: Task Planning

Input: the current state s, a set of unfinished tasks T, and a set of free responders I.
Output: a task assignment for all responders.
$\{C_{jk}\} \leftarrow$ compute all possible coalitions of I for T
foreach $C_{jk} \in \{C_{jk}\}$ **do**
  `// The` N `trial simulations`
  for $i = 1$ **to** N **do**
    $(r, s') \leftarrow$ simulate the process with the starting state s until task k is completed by the responders in C_{jk}
    $V(s') \leftarrow$ estimate the value of s' with MCTS
    $v_i(C_{jk}) \leftarrow r + \gamma V(s')$
  $v(C_{jk}) \leftarrow \frac{1}{N} \sum_{i=1}^{N} v_i(C_{jk})$
return the task assignment computed by Equation 2

3.2 Path planning

In the path planning, we compute the best path for a player given the location of his assigned task. This is a single-agent problem that can be modeled as a MDP, $\langle S_i, A_i, P_i, R_i \rangle$, where:

- $S_i = S_r \times S_{p_i}$ is the state space. Player i only need to consider his own state variable (location and health level) and the state variable of the radiation cloud.
- A_i is the set of actions i. Player i only need to consider the actions of staying in the same grid or moving to the 8 neighboring grids.
- $P_i = P_r \times P_{p_i}$ is the transition function. Player i only need to consider the expanding of the radiation cloud and the change of his location and health level when moving in the grid map.
- R_i is the reward function. Player i has a small cost for moving around and a large penalty for being killed by entering the radiation cloud.

This process terminates when the location of the assigned task is reached or the player is killed (the health level is 0) by the radiation cloud. This is a typical MDP and can be solved by many state-of-the-art MDP solvers [**?**]. The key challenge is that the state space will blow up very quickly given a large map. To address this, we adopt Real-Time Dynamic Programming (RTDP) [**?**]. The main process is outline in Algorithm 3.

Instead of considering the whole state space, RTDP only visits the states that are reachable from the starting state s^0. If the loop cannot terminate in a fixed number of iterations, we assume there is not valid path between the two locations (either there are obstacle between them or the responder is killed by the radiative cloud on the path). There are several techniques we used to speed up the convergency of RTDP. In our problem, the map is static. Thus, we can initialize $V(s)$ by the cost value of the map, computed offline without considering the radiative cloud. This can be done by Value Iteration [**?**] assuming that any location on the map is the goal. The benefit of so doing is to help RTDP

Algorithm 3: Path Planning

Input: the starting state s^0 and the goal state s^g.
Output: a path from the starting location to the goal.
$s \leftarrow s^0$
repeat
  foreach $a \in A_i$ **do**
    $Q(s,a) \leftarrow R_i(s,a) + \sum_{s' \in S_i} P_i(s'|s,a)V(s')$
  $a \leftarrow \arg\max_{a' \in A_i} Q(s,a')$
  $V(s) \leftarrow Q(s,a)$
  $s' \sim P_i(s'|s,a)$
  $s \leftarrow s'$
until $s = s^g$
return Q

quickly navigate among the obstacles (e.g., buildings, water pools, blocked roads) without getting trapped in dead ends. Another technique is that, when traversing the reachable states ($s' \in S_i$ in Algorithm 3), we only consider the responder's current location and his or her 8 neighboring grids since $P_i(s'|s,a) = 0$ for other locations. This will further speed up the algorithm because, as mentioned above, the main challenge of path planning in our problem is the huge state space.

4. THE ATOMIC ORCHID PLATFORM

Joel and Wenchao

1. explain the main components and how agent is integrated
2. explain the instructions given to participants and how it mimics the disaster response problem detailed above.

4.1 Game scenario

AtomicOrchid is a location-based mobile game based on the fictitious scenario of radioactive explosions creating expanding and moving radioactive clouds that pose a threat to responders on the ground (the field players), and the targets to be rescued around the game area. Field responders are assigned a specific role (e.g. 'medic', 'transporter', 'soldier', 'ambulance') and targets have specific role requirements, so that only certain teams of responders can pick up certain targets. For example, an 'injured person' can only be picked up by an 'ambulance' and a 'medic' together. To pick up targets, the team must be collocated in the immediate proximity of the geofenced target. Furthermore, field responders must not expose themselves to radioactivity from the cloud for too long, else they risk becoming 'incapacitated'.

In their mission to rescue all the targets from the radioactive zone, the field responders are supported by (at least one) person in a centrally located HQ room, and the planning agent that sends the next task to the team of field responders [assuming the agent will have been described in detail already].

4.2 Player interfaces

Field responders are equipped with a 'mobile responder tool' providing sensing and awareness capabilities in three tabs (geiger counter, map, messaging and tasks; see figure XX). One tab shows a reading of radioactivity, player health level (based on exposure), and a GPS-enabled map of the game area to locate fellow responders, the targets to be rescued and the drop off zones for the targets. Another tab provides a broadcast messaging interface to communicate with fellow responders (field responders and HQ). Another tab shows the team and task allocation dynamically provided by the agent. Notifications are used to alert both to new messages and task allocations.

HQ is manned by at least one player who has at their disposal an 'HQ dashboard' that provides an overview of the game area, including real-time information of the players' locations (see figure XX). The dashboard provides a broadcast messaging widget, and a player status widget so that the responders' exposure and health levels can be monitored. HQ can further monitor the current team and task allocations by the agent. Importantly, only HQ has a view of the radioactive cloud, depicted as a heatmap. 'Hotter' zones correspond with higher levels of radioactivity.

4.3 Planning agent

[Wenchao. Describe how the agent works (not implementation detail, add that in subsection below), i.e., when it is polled, what information is being exchanged, and how the team/task allocation is being constructed from that and sent.]

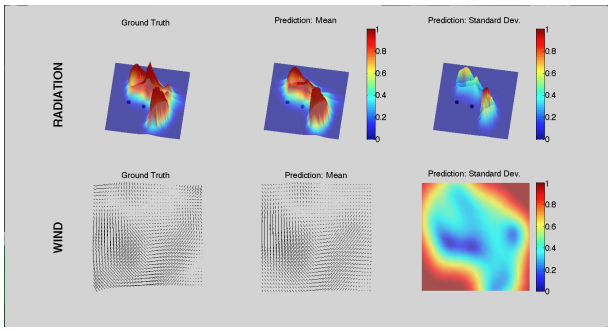
4.4 Radiation Cloud Modelling

The radiation cloud diffusion process is modelled as a non-linear Markov field stochastic differential equation (which assumes the cloud intensity is Gaussian distributed in log-space). The cloud is driven by wind forces which vary both spatially and temporally. The wind velocity is modelled by two a priori independent Gaussian processes, one GP for each Cartesian coordinate axis. The GP captures both the spatial distribution of wind velocity and also the dynamic process resulting from shifting wind patterns such as short term gusts and longer term variations. In our simulation the each spatial wind velocity component is modelled by a squared-exponential GP covariance function, K, with fixed input and output scales over time (although any covariance function can be substituted).

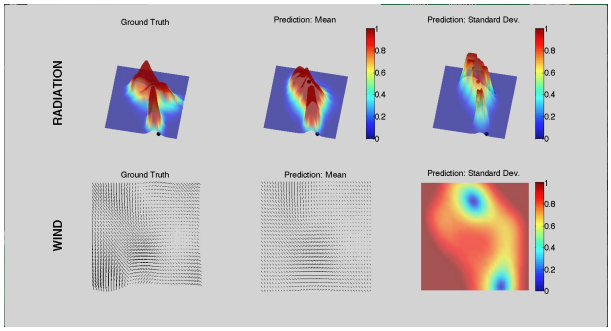
Both the radiation cloud and wind model priors are combined into a single joint model called a *latent force model* (LFM) [**?**] and predictions of the radiation cloud intensity are inferred using the extended Kalman filter (EKF). The EKF provides both an estimate of the radiation cloud and wind conditions as well as an indication of uncertainty in these estimates arising through sparse observations of the dynamic radiation intensity and wind conditions. The EKF state $S(t) = (\underline{R}(t)\underline{V}_x(t)\underline{V}_y(t))$ represents both the wind velocities, $V_x(t)$ and $V_y(t)$, and the log-radiation cloud density, $R(t)$, on a regular $N \times M$ grid defined across the environment. The temporal component of the wind GP models is assumed Markovian and thus, the wind dynamics are incorporated with the EKF as per the KFGP [**?**]. For example, the $N \times M$ x-component of the wind velocity prediction at time-step $t+1$ is $V_x(t+1) = FV_x(t) + \nu_t$, where the process model $F = \rho I$ and Gaussian process noise $\nu_t \sim \mathbb{N}(0, (1-\rho^2)K)$ for some time-step correlation ρ. When $\rho = 1$ then the wind velocities are constant. Values of ρ close to zero model gusty wind conditions.

The cloud intensity and wind velocity are measured by *monitor agents* equipped with geiger-counters and anemometers. These agents are directed to take measurements with

greatest information gain in the radiation cloud intensity. The measurements are folded into the EKF and this refines estimates of the radiation cloud across the grid. Figure 2 shows example cloud simulations for slow varying (i.e. $\rho = 0.99$) and gusty (i.e. $\rho = 0.90$) wind conditions. Figure 2(a) shows slow varying conditions in which case the radiation cloud can be interpolated accurately using sparse sensor measurements and the LFM model. Alternatively, during gusty conditions Figure 2(b) the radiation cloud model is more uncertain far from the positions where measurements are taken.



(a) Slow varying wind conditions



(b) Gusty wind conditions

Figure 2: Radiation and wind simulation ground truth and estimates obtained from monitor agents (black dots). Left most panes are ground truth, middle panes are estimates and right most panes are estimate uncertainties. (a) Slow varying wind conditions and (b) gusty wind conditions.

4.5 System architecture

[Wenchao: adapt this to version 2.0] AtomicOrchid is based on the open-sourced geo-fencing game MapAttack[1] that has been iteratively developed for a responsive, (relatively) scalable experience. The location-based game is realized by client-server architecture, relying on real-time data streaming between client and server.

The client-server architecture is depicted in figure XX. Client-side requests for for less dynamic content use HTTP. Frequent events, such as location updates and radiation exposure, are streamed to clients to avoid the overhead of HTTP. In this way, field responders are kept informed in near real-time.

[1]http://mapattack.org

The planning agent agent ... [add implementation detail]

The platform is built using the geoloqi platform, Sinatra for Ruby, and state-of-the-art web technologies such as socket.io, node.js, redis and Synchrony for Sinatra, and the Google Maps API. Open source mobile client apps that are part native, part browser based exist for iPhone and Android; we adapted an Android app to build the mobile responder app.

5. PILOT STUDY

Joel and Wenchao

1. Explain setup of experiment - area of interest + setup of tasks
2. Explain evaluation = quantitative and qualitative.

Metrics.

- Comparisons between with/without agent versions for the below:
- Performance of FR: number of tasks completed, time on task?, number of messages sent, number of teams formed and disbanded, time on team, acknowledgements of tasks
- Messages: classification
- Health
- Distance travelled
- HQ: number of agent monitoring actions (clicks), number of 'supporting'/related messages (e.g., enforcement, contradictions/overriding)
- Agent performance: number of instructions, number of replanning steps, replanning robustness (diversion of task allocation compared to previous step)
- Following instructions ('obedience'): number of instructions followed vs. not followed (incl. number of HQ interventions/overriding agent allocation), instruction handling diagram
-

5.1 Conclusions

6. REFERENCES

[1] A. Mauricio, D. Luengo, N. D. Lawrence, et al. Latent force models. In *International Conference on Artificial Intelligence and Statistics*, pages 9–16, 2009.

[2] S. Reece and S. Roberts. An introduction to gaussian processes for the kalman filter expert. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–9. IEEE, 2010.