Scheduling Sensors for Monitoring Sentient Spaces using an Approximate POMDP Policy

Ronen Vaisenberg, Alessio Della Motta, Sharad Mehrotra and Deva Ramanan

The University of California, Irvine
ronen@ics.uci.edu, alessio.dellamotta@gmail.com, sharad@ics.uci.edu, dramanan@ics.uci.edu

***Abstract*—We present a framework for sensor actuation and control in sentient spaces, in which sensors are used to observe a physical phenomena. We focus on sentient spaces that enable pervasive computing applications, such as smart video surveillance and situational awareness in instrumented office environments. Our framework utilizes the spatio-temporal statistical properties of an observed phenomena, with the goal of maximizing an application-specified reward. Specifically, we define an observation of a phenomena by assigning it a discrete value (state) and we model its semantics as the transition between these values (states). This semantic model is used to predict the future states in which the phenomena is likely to be at, based on partially-observed past states. To accomplish real-time agility, we designed an approximate, adaptive-grid solution for Partially Observable Markov Decision Processes (POMDPs) that yields practically good results, and in some cases, guarantees on the quality of the approximation. We use our framework to control and actuate a large-scale camera network so as to maximize the number and type of captured events. To enable real-time control, we implement an action schedule using a table lookup and make use of a factored probability model to capture state semantics. To the best of our knowledge, we are the first to address the problem of actuating a large-scale sensor network based on a real-time POMDP formulation.**

I. Introduction

Recent advances in sensing technologies have enabled the infusion of information technology into physical processes. This offers unprecedented opportunities, the impact of which will rival (if not exceed) that of the WWW. Such sensor-enabled environments can realize sentient spaces that have the potential to revolutionize almost every aspect of our society. Sentient systems observe the state of the physical world, analyze and act based on it. Sentient systems enable a rich set of pervasive computing applications including smart video surveillance, situational awareness for emergency response and social interactions in instrumented office environments to name a few. Sentient systems present new forms of computational challenges, from modeling, control, and scheduling perspectives. One has to balance the needs of (possibly multiple) pervasive computing applications with physical constraints of sensors/devices being controlled, all while achieving situational awareness of the physical world being monitored. In this paper, we use additional information, namely the semantics and predictability of the monitored world, to improve the agility of sentient systems. We design real-time algorithms that use probabilistic semantic models to schedule sensor actuation in an "optimal" way. We evaluate our approach in an instrumented smart building, specifically using it to dynamically control a camera network so as to record the events of interest.

A crucial aspect of sentient spaces is that they often provide only a partial or noisy view of the world, due to physical constraints on the sensors themselves. In such cases, schedulers should balance the demands of an application with the need to better observe the world. We advocate the use of a Partially Observable Markov Decision Process (POMDP) to reason about sensor actuation with physically-realized, non-ideal sensors. We explore our framework in the context of large-scale camera networks. Specifically, we use POMDPs to control pan-zoom-tilt (PZT) parameters of cameras for a variety of pervasive computing applications such as as building occupancy estimation, identity recognition, people tracking, etc. Notably, our framework schedules actuations so as to simultaneously maximize the (possibly-conflicting) benefits of differing end applications.

The key technical contributions of this work are as follows: first we propose a general POMDP framework for sensor actuation problems that balances application need and situational awareness. Second, we introduce approximations that are required to scale POMDP solvers for *real-time* actuation of large-scale sensor networks. We further extend the approximation framework to balance the need of multiple monitoring applications. Third, we demonstrate the effectiveness of our system, by comparing it to commonly-used baseline scheduling algorithms, on a recording from a large scale camera network in a campus building. Our results suggest that sensor level challenges can be abstracted and effectively addressed by a middleware layer.

To the best of our knowledge, we are the first to address the problem of actuating a large-scale sensor network based on an approximated, real-time POMDP. We accomplish this by using a semantic model of building activities and approximating the expected utility of scheduler actions. We describe a semantic correlation model that is simple enough to be implemented in real-time, yet powerful enough

to capture meaningful semantics of typical behavior. Our action selection process is as fast as a table lookup in real-time. Though we focus on large-scale camera networks, our approach applies to the general problem of sensor selection and actuation in sentient spaces. We briefly discuss such extensions in Sec. VI. We believe that our approach will serve as a fundamental building block for building next-generation sentient systems.

II. Problem Formulation

A. Abstract Problem Formulation

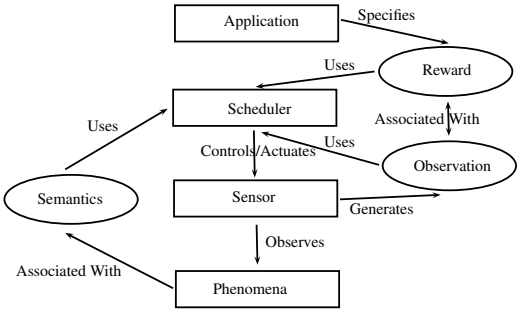


Figure 1. Sentient Spaces High-Level Concepts.

Fig. 1 outlines the high-level concepts of our sensor actuation framework. A pervasive application specifies a reward that is associated with an observation of a phenomena. The scheduler controls and actuates the sensor so as to better observe the phenomena and maximize the application-specific reward. To do so, the scheduler uses semantics that are associated with the phenomena.

Environment Model: The environment model is an abstract representation of the monitored environment. For instance, in a monitored building the environment model is used to refer to different regions such as "r_1 = south west hallway, second floor" and "r_2 = second floor kitchen".

Phenomena: The phenomena is any kind of feature property whose value is amenable to observation or estimation, including physical properties, existence and occurrence assessments, etc. The phenomena happens in the instrumented space and changes (states) over time. The phenomena has a measurable value instance (state) at a particular time, written as X_t. For instance, consider $\boldsymbol{N}$ spatial regions of an instrumented building, each of which can be empty or occupied by a person. In this case, there exist 2^N possible phenomena states.

Semantics: We make use of semantic rules that define how the phenomena evolves over time and space. Specifically, in the context of this paper, we model these semantics as $P(\{X_{t+1}, .., X_{t+T}\}|\{X_t, .., X_{t-M}\})$, which is a function that returns predictions of the next $\boldsymbol{T}$ timesteps into the future given the states from the past $\boldsymbol{M}$ timesteps.

Sensor: The sensor is the physical device that is used to observe a phenomena. The system has access to sensors (i.e., cameras) $s_1, .., s_n$ that are controlled by a scheduler.

Scheduler: The scheduler controls the capture and processing parameters at time t by choosing an action $A_t \in A$.

Observation: At any time t, given the phenomena state and scheduler action, the system generates an observation $O(X_t, A_t)$.

Applications: We assume that multiple pervasive computing applications wish to monitor phenomena of different types. The sentient system is responsible of generating a schedule such that sensor data captured by the system satisfies the application needs.

Rewards: We define $R(O(X_t, A_t)) \rightarrow \mathbb{R}$ as an application-specific reward that depends on the observation, which itself depends on the phenomena state and the selected action. For instance, in our building monitoring setting, an image containing entities will have a higher reward as compared to an image with none. We account for the general setting there may exist multiple applications with different reward functions.

B. Instantiating the Framework for a Camera Network

We focus on a active camera network, where the set of actions A cam be modeled with discrete PZT states of each camera. Each camera can be zoomed out, or zoomed into a particular region within the field of view. We denote the collection of N regions observed throughout the entire network as a vector X_t, where $X_t^r \in \{0, 1\}$ indicates if there exists a person in region r at time t. A is the actuation plan specifying the PZT state of each camera (either zoomed-out or the region being zoomed into). The state transition model P captures the "motion semantics" of how people move in the monitored space; e.g., people tend to appear in spatially adjacent regions of cameras as they walk. In our model, actions do not directly affect the phenomena but the observation of it; $P_{A_t \in A}(X_{t+1}|X_t) = P(X_{t+1}|X_t)$. We write $O^r(X_t, A_t)$ for the observation at region r at time t given world state X_t and actuation plan A_t. $R(X_t, A_t)$ is the reward associated with being at state X_t while taking action A_t; we assume this decomposes into a sum of per-region rewards $R(X_t, A_t) = \sum_{r=1}^{N} R(X_t^r, A_t)$.

In this setting, we would like a system which schedules future actuations given a *partial observation* of the world, provided by the actuated camera network. It must trade-off actions that satisfy the immediate needs of an application versus those that maintain an accurate representation of the world. Specifically, due to physical limitations, each camera can be actuated to exist at a single PZT state. A zoomed in state allows the system to collect high resolution images at the cost of a limited field of view, which limits the system's knowledge of the monitored space. On the other hand, a low resolution image, while not providing a enough detail for face recognition, may still be used to obtain a

coarse understanding of how people are moving within the monitored space, which in turn may help target the collection of high-resolution images in the near *future*.

We define a POMDP that naturally balances both long and short term benefits of the actuation problem. Our POMDP is a five-tuple (S, A, P, O, R) where S is the set of states of the physical system being monitored, A is the set of actions, P is the state transition model of the physical system, O is the state observation model, and R is the state-action reward function. Given the above POMDP, the goal is to find a policy π that specifies an action for each estimated world state so as to maximize the total expected reward over some finite horizon: $\sum_{t=1}^{T} E[R(X_t, A_t)]$. We refer to T as the "look-ahead" of the system; if $T = 1$, the system is myopic and will always actuate sensors to maximize its immediate reward. With larger T, an optimal policy will realize that it is better sometimes take actions to reduce uncertainty, as this will produce a more accurate estimate of world state which can help increase future rewards.

While discrete-state MDPs are solvable with dynamic programming, POMDPs are intractable [11]. One attractive approach is to cast a POMDP as a MDP whose state space becomes a "belief state", or the set of all probability distributions over world states. Unfortunately, one needs a exponentially-large number of gridded or quantized belief states. One of our primary contributions in this work is an adaptive gridding strategy that dynamically adds discrete states to an underlying belief-state MDP, so as to obtain a provably good approximation of the value (or expected reward) of each belief state and action.

III. Approach

The table below summarizes the different challenges and our approach to address each one:

* N regions, M past states, T seconds lookahead)

Challenge	Approach	Subsec.
Transition model is exponential $O(2^{NM})$.	Assume that people move independently and use Noisy OR model which is linear $O(N^2 M)$ (as in [14]).	III-A
Scheduler belief state requires 2^N dimensions.	Assume that the scheduler belief state can be factored into a belief state for each individual region which reduces the belief state to N dimensions.	III-B
Looking ahead by growing a search tree is exponential in time $O(N^t)$	Represent only certain scheduler states and use dynamic programming (value iteration) $O(NT)$.	III-C
Hard to select which scheduler states to represent apriori.	Based on the properties of our reward function we bound the error and dynamically add points to the representation if the error is not acceptable.	III-E
Scheduling decisions must be made at real-time.	Partitioning the space by cameras so that only regions that are in direct competition be in the same partition. Pre - compute an approximated state action grid for each camera with constant space requirements.	III-G

The rest of the section discusses in details each of the above challenges.

A. State transition model

Our monitored space is divided into N disjoint regions. At (discrete) time t the observed phenomena is assumed to be in some state X_t. X_t is represented as a binary vector of length N (for example, $X_t^i \in \{0, 1\}$ can be used to encode the presence of a person in region i at time t or lack of thereof.

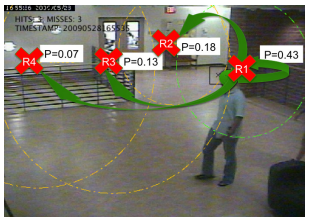


Figure 2. Illustrating the conditional probability transition model.

Fig. 2 illustrates the conditional transition probabilities of the face appearing in each of the regions in that camera given its current location. The number of possible world states is 2^N, meaning that naive state transition models will not scale. We adopt the factored transition model of [14] which assumes that entities will move independently in a monitored environment:

$$P(X_t^j | X_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{o=1}^{M} \prod_{i=1}^{N} (1 - \alpha_{ij}^o X_{t-o}^i) \quad (1)$$

where α_{ij}^o is the probability that a person moves to region j o-seconds later, given that they are currently at region i.

The above model can be derived from a noisy-OR assumption on spatio-temporal correlation. While simplistic, it captures many natural semantics about object movement (people tend to walk down hallways, wait at elevators, etc.) without explicit multi-object tracking, which can be difficult in unconstrained scenarios. [14] show that the factored-frontier algorithm [8], or one-pass belief-propagation, suffices for inference with such a model. The final algorithm can be intuitively thought of as a prediction filter (that predicts future states given the semantic model) followed by a correction stage (that re-weights predictions using new observations).

B. Approximating the POMDP Belief States

POMDP solvers maintain an internal belief state about the external world. Because the number of distinct world states is exponentially large (2^N), representing distributions over such spaces would require a vector of (2^N) dimensions. Fortunately, the factored inference algorithm of the previous section suggests that one can approximate the probability over states as a product of marginal probabilities for each of the N regions. Hence we represent a belief state $\hat{X}_t$ as a N dimensional vector where each entry $\hat{X_t^j}$ is the probability of an event taking place in region R_j at time t.

Scheduling based on the Estimated State $\hat{X}_t$ - Given our best estimate of the state of the world - $\hat{X}_t$ we face a

difficult problem: What should the course of action be for the next time step? One might suggest zooming in to the most likely region in terms of the computed probability. In this case, the scheduler will always be zoomed in, as there will always be a region which has the highest probability. Furthermore, by following this approach, the probabilities will be computed based on a partial view of the space, as each camera is limited to viewing only the region it is zoomed in to. In a camera network of 6 cameras where each camera has 7 regions, there will constantly be 6*6=36 regions out of the 42 for which the scheduler has high uncertainty. Optimally we should choose the action with the highest expected utility in the "long run", taking into account both the indirect reward associated with reduced uncertainty and the direct reward associated with satisfying the application requirements.

C. Value Iteration

In this section, we temporarily make the simplifying assumption that approximated belief states $\hat{X}_t$ can be represented as discrete elements $s \in S$ such that $|S| = n$. In this case, our POMDP can be written as a MDP with discrete states, and the optimal policy π can be represented as a table with $O(n)$ entries whose entries can be computed in $O(nT|A|)$ with dynamic programming (where T is the horizon and A is the set of possible actions). The full algorithm, described in Alg. 1, is known as value iteration. We refer the reader to [3], [12] for complete details, but we briefly review it here as our solution builds upon it.

Algorithm 1: Dynamic Programming Value Iteration

```
Data: follow(s,a)- Returns the set of states reachable from state s
      taking action a.
1  begin
2      /* Initialization */
3      for s ∈ S do
4          for t ∈ 1..T do
5              opt[s,t] ⟵ 0
6      for s ∈ S do
7          for t ∈ (T − 1)..1 do
8              for a ∈ actions do
9                  /* Compute the expected reward for a at state s
                      when t seconds remain. */
10                 a_R ← 0
11                 for s_{t+1} ∈ follow(s,a) do
12                     a_R ← a_R +
                       p(s_{t+1}|s) * (R(s_{t+1}|a) + opt[s_{t+1}, t + 1])
13                 opt[s,t] ← max(opt[s,t], a_R)
```

Alg. 1 computes the maximal expected reward by performing a "backward walk" in time. At time T, the time has already ended (we started from 0, and performed a look ahead of T seconds). Thus the maximal expected utility is zero. At time $t \leq T-1$ the process computes the expected reward of state s for each action $a \in A$ by taking into account all possible scheduler belief states s_{t+1} that can follow from s when action a is taken and looking up their expected reward in the dynamic programming table. The immediate reward of being in state state s_{t+1} while taking action a is $R(s_{t+1}|a)$. The value stored in the table is that of the action with the highest expected utility $E_{max}[R(s)] = opt(s)$. The following subsections until III-G present our approach to discretize S, and find an approximate solution with quality guarantees.

D. Grid Creation

In order to apply the above approach using a discretized set of beliefs states S, we start by including the zero and unity vectors and creating an equally spaced grid points along probability vectors. However, future belief states s_{t+1} in line 11 in Alg. 1 will likely not be present in the finite set S. One can use the closest element in S, but this introduces errors that are compounded over time. To address this, we describe an adaptive strategy for iteratively adding discrete states to keep the error within some tolerance.

E. Adaptive Discretization with Bounded Error

We describe an algorithm for adaptively discretizing the state space S during Value Iteration such that the expected utility *opt* is computed within some user-provided tolerance. This bound is possible due to the additive and monotonic nature of the reward function in sensor actuation problems, which in turn lets us compute upper and lower bounds on *opt* during dynamic programming. We begin by defining two marginal belief state sets which will help us bound the error with a given state s:

$$FloorSet(s) = \{sf | \forall_j sf^j \leq s^j \ and \ s \in S\}$$

$$CeilingSet(s) = \{sc | \forall_j sc^j \geq s^j \ and \ s \in S\}$$

The ceiling set of a state s contains all the approximated marginal belief states that have a higher or equal probability of an event across all regions. Similarly the floor set has lower or equal probabilities.

Lemma: $\forall_{sf \in FloorSet(s)} E_{max}[R(sf)] \leq E_{max}[R(s)]$

Proof (sketch): We model the space as a vector X_t where the goal is to actuate sensors to observe as many $X_t^i = 1$ as possible. Our approximate belief state is represented as marginal probabilities, each element in s_i represents the probability of an activity taking place in region r_i. Thus, reducing the probabilities in s to those in sf reduces the probability of transitioning to any state which contains $X_t^i = 1$ due to the additive nature of our model (see eq. 1).

We compute the maximal expected utility of s based on the utility of $\hat{s} \in follow(s,a)$ which we repeatedly replace either with a state $\hat{s} \in FloorSet(s)$ if $\hat{s}$ is not present in the grid, or the exact value if $\hat{s}$ is present. In any case, the

expected utility that will be computed will be a lower limit on the true expected utility:

$$\forall_{sf \in FloorSet(s)} E_{max}[R(sf)] \leq E_{max}[R(s)]$$

■

Similar argument can be used to show that increasing the probabilities, results in an upper bound on the true expected utility:

$$\forall_{sc \in CeilingSet(s)} E_{max}[R(sc)] \geq E_{max}[R(s)]$$

This property allows us to bound the true utility:

$$opt[sf, k] \leq opt[s, k] \leq opt[sc, k]$$

Whenever we need to approximate state s_{t+1} we take the closest state in $FloorSet(s) - sf$ and the one in $CeilingSet(s) - sc$. Each entry in the dynamic programming table maintains four values: the expected utility when consistently approximating states using their floor set as well as the ceiling set and the actions that were used to achieve the highest utility for the floor and ceiling states.

Before the dynamic programming routine processes the entries for the next time step (as part of the for loop in line 7 of Alg. 1) we verify that the error rates in the entries we just computed meet the desired error level. This allows us to upper bound the error by the following expression: $\epsilon \leq opt[sc, k] - opt[sf, k]$ If the upper limit on the error rate is below the desired error, the process terminates and returns the computed table. In the case where the error is above the user specified threshold, we must add states to our approximation of the state space - S and restart the process.

F. Adding States to the Grid

If the error is above the desired threshold, we use the current grid to generate a new, finer grid by adding states based on their expected contribution to reducing the error. To accomplish this, we keep track of the states that were missing from the current grid and had to be approximated when s_{t+1} was computed (line 11 of Alg. 1). We maintain a priority queue with the top k states (in our experiments we used k = 5000) ordered based on their Euclidean distance from the closest state currently in S or the priority queue. The reason we want to limit the number of states that are added in each iteration is to ensure that we don't increase the state space unnecessarily. Although there might be more than k states that were needed, once we add the top k states we might meet the ϵ required error bound. This allows us to control the growth rate of the state space such that it is within k states of the minimal size needed. Without this iterative selection process the state space explodes with states which are very close to each other and cause the process to exceed memory limits. The prioritization routine allows the process to find solutions for much smaller ϵ values. Moreover this allows the user to specify a timeout condition or grid size limit (as well as ϵ) which will return the best solution (perhaps exceeding ϵ error) that could be found within the time and space constraints. Fig. 4(a) illustrates the number of states that are needed for different ϵ values to compute a policy for a single camera with four regions. As the tolerated error rate decreases the number of states increases. The problem of finding an optimal scheduler remains not tractable, as a small ϵ will require an unbounded sized state space but solutions are iteratively improving and the processing duration and required state space can be controlled by changing ϵ. Furthermore, in real settings, the number of states that are actually needed in order to get a relatively low error rates is manageable as illustrated by Fig. 4(a).

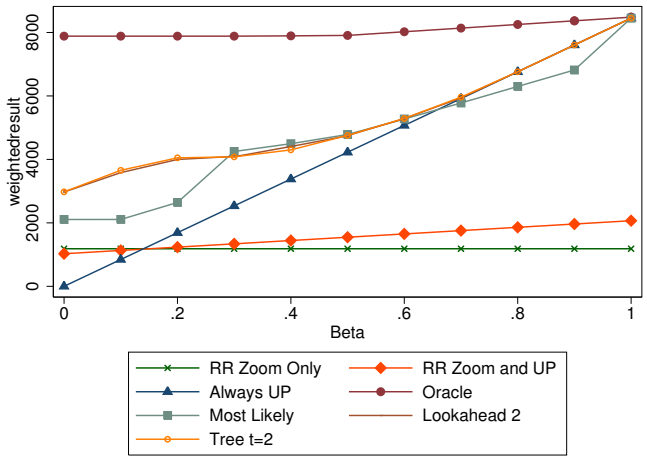
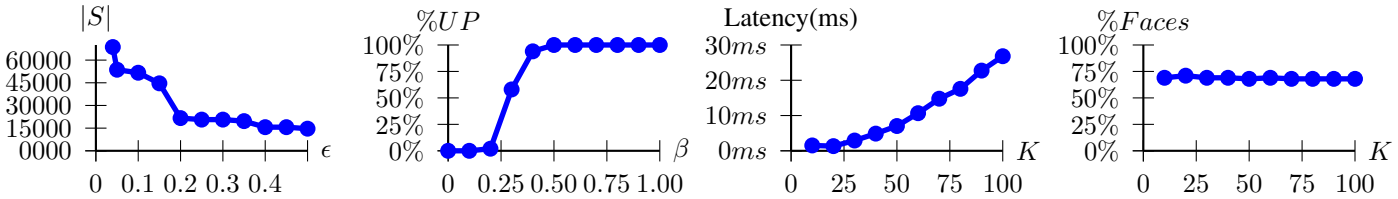


Figure 3. Total reward for different alternatives, 7 zoom regions.

An improved initial grid - We applied an approach which allows us to gauge the need in states so that ones that are more likely to be accessed are included in the grid as follows: we first perform an offline execution of a tree based approach, with lookahead of 2 seconds, computing the exact utility by growing a tree of all possible states and actions from a given scheduler state, for each camera. While the tree based approach is in execution we added points to the initial grid based on the same technique that we used to prioritize which states to add so that the error rates are within bounds. In this case, we used a priority queue to decide on which states to keep in the initial grid based on the states that are more likely to be accessed by the real-time scheduler. Saving all the states that could possibly be encountered is exponential, thus the priority queue approach allows us to effectively allocate our memory resources to where they are most needed. Note that the error bound guarantees are for points that are in the grid, thus, creating an initial grid this way reduces the errors introduced when points are looked up by the algorithm while at execution.

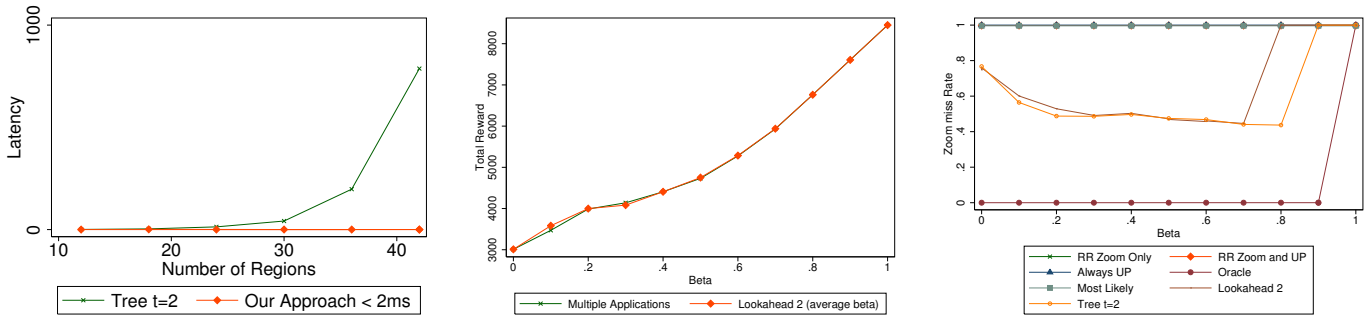


(a) Size of the State Grid as a function of ϵ

(b) % of states going "UP" as a function of β (when $\alpha = 1$)

(c) Latency of Scheduling as a function of number of cameras

(d) % of faces collected at high resolution as a function of the number of cameras



(e) "optimal solution" vs "approximate solution".

(f) Evaluation of our approach to address multiple concurrent pervasive computing applications.

(g) Rate of actuations not resulting in a facial image.

Figure 4. Illustrating the different aspects of the suggested approximated POMDP approach.

Efficiently Approximating s_{t+1} - Whenever a state is approximated we find its nearest neighbor in its floor and ceiling sets. This is done efficiently by using a range query over a KD-Tree[2], [7]. We start with S containing an equally spaced grid across all dimensions. For example, if we space each dimension according to $G = \{0, 0.5, 1\}$ of a four dimensional state space then $S = G \times G \times G \times G$. Whenever a state s_{t+1} is approximated we issue a range query between two points in the d-dimensional space - s_{t+1} and $Ceiling(s_{t+1})$. Where $Ceiling(s_{t+1})$ is the closest point to s_{t+1} that is also in $CeilingSet(s_{t+1})$. Such a point can be found efficiently by a binary search across each dimension. Since $Ceiling(s_{t+1}) \in S$ we are guaranteed a non empty result set in our range query, from which we select the nearest neighbor - sc. Similarly we select the closest point on the grid which is in s_{t+1}'s floor set sf. The closest of sc and sf is used to approximate s_{t+1} and $\epsilon \leq opt[sc, k] - opt[sf, k]$ is used to approximate the error associated with this approximation. This process is used to perform a lookup of a value in the grid both for the dynamic programming routine as well when an action is selected for a given state. KD-Range queries have time complexity of $O(\sqrt{|S|})$ and thus the process of approximating a given state to an existing grid point is extremely efficient. We report latency results as we increase the number of regions in each camera in Fig. 4(e) (discussed in details in the experiments section).

G. Distributed Real-Time Scheduling

At run-time, our scheduler updates its beliefs about the world using the **global** transition model - $\hat{X}_t = P(X_t^j | X_{(t-M):(t-1)}^{1:N})$. Beliefs are updated by standard prediction/correction equations (see Sec. III-A) for temporal models. Notably, this model is **global** in that correlations across cameras are taken into account. However, approximating the optimal policy table (π) based on the estimated state $\hat{X}_t$ with small ϵ is still intractable for networks with a large number of cameras. This is because n, the number of discretized states required to achieve a low error, empirically tends to grow exponentially in T (expected, as solving POMDP optimally is not tractable). To address this limitation, we adopted a factored scheduling approach, where the **global** scheduling table is partitioned into non-overlapping **local** scheduling tables. In our implemented system, cameras observe non-overlapping regions, and so it is natural to build a local policy table (or local scheduler) for each camera[1].

Overall, real-time scheduling proceeds as follows: region-specific beliefs $\hat{X}_t^r$ are updated using prediction/corection filtering, which require (ON^2M) multiplications. The updated beliefs are then used to query a camera-specific policy table to return a scheduled action state for each camera. This query requires a single table lookup per camera $O(N)$, making the overall scheduler extremely lightweight and fast.

[1]Further approximations for overlapping regions are discussed in p.109 [13]

H. Multiple Pervasive Computing Applications

Consider the scenario where multiple applications make use of different reward functions that may even change over time. For example, one may be interested in high and low-resolution during the day time, but interested in only high-resolution images at night (because low-resolution images under low light will be noisy). Assume we are given Q different applications. For any subset of active applications, one can compute the optimal policy using a reward function equal to the sum of each application-specific rewards. A naive approach would pre-compute an exponential (2^Q) number of policies corresponding to all possible subsets of active applications. We now describe an efficient algorithm for representing this set, making use of the additive property of our reward function. Let us augment our policy table from Alg.1 to store the expected utility of all actions for each state, and not just the best action. We then compute Q application-specific augmented policy tables. Given these Q tables, we compute policies for any subset of active applications by adding together "on-the-fly" the expected utility for each action-state, as described by Alg. 2.

Algorithm 2: Approximate approach to actuate sensors for concurrent applications

```
Data: E_app[s,t,a]- Returns the expected reward for app taking
      action a from state s when t seconds remaining.
begin
    best_action ← null
    best_action_reward ← 0
    for a ∈ actions do
        action_reward ← 0
        for app ∈ applications do
            /* Compute the expected utility for a at state s when
            app applications executing. */
            action_reward += E_app[s,t,a]
        /* Save the action with maximal total reward. */
        if best_action_reward < action_reward then
            best_action_reward ← action_reward
            best_action ← a
    return best_action
```

IV. Experimentation

The main parameters of the our approach are space-partitions (the number of regions in a camera and number of cameras), the reward function, and the error tolerance ϵ for our POMDP solver. In terms of evaluation, there are three important metrics (1) total reward after running the scheduler (2) latency of discovering events, and (3) number of events of interest detected versus missed (i.e., precision and recall). We split our experimental results in those that (a) compare to standard scheduling baselines (Fig. 3) and (2) diagnose the impact of various parameters (Fig. 4).

Baseline comparisons: We have implemented different baseline alternatives for comparison: 1) Consider a scheduler that for each camera, cycles between a zoomed out configuration and zooming into the regions covered by it, one by one in Round Robin. 2) Cycles in Round Robin between zoomed in configurations only in Round Robin. 3) Stays "UP" in all cameras at all times. 4) Select an action that has the highest expected utility based on the predicted probabilities. 5) Our approximated proposed approach with look ahead of two seconds. 6) A Tree based approach computing the exact utility by growing a tree of all possible states and actions from a given scheduler state, for each camera. This brute-force solution requires searching over an exponentially large space, and so is useful for analysis though not practice. 7) An "Oracle" approach which selects the best camera actuation taking into account events taking place in the future. Note that the performance of the "Oracle" approach might not even be achievable since we don't have access to future events.

Source code for all experiments in this section is available online [2] Our evaluation dataset contains recordings of two full days of unscripted human activity[3]. The data was recorded by a network of 6 cameras covering a single floor in the computer science building. For each algorithm we evaluated, the data recorded was replayed exactly as it happened in real-life, and the scheduler was given access only to data that it scheduled to collect and was evaluated based on its performance. We have divided each camera to $k = 7$ zoom regions. The more regions each camera has, the harder it is for the scheduler to select the right region to zoom in. The smaller the number of regions, the easier it is to "guess" a region that will contain a person. The reward for each region is defined as follows:

$$R(X_t^r, A_t) = \begin{cases} \alpha & \text{High-res face, } X_t^r = 1, r \in A_t \\ \beta & \text{Low-res face, } X_t^r = 1, r \notin A_t \\ 0 & Otherwise \end{cases}$$

This representation expresses a large family of reward functions, intuitively presenting the utility of a high resolution image vs. a low resolution one. The evaluation was done by replaying the events that took place in the observed environment and computing the total reward that each alternative was able to achieve. For example, a single point in our evaluation graphs would be the total reward by Round Robin, for $k = 7, \alpha = 1, \beta = 0$. Varying β gives us a way to see how RR performs with different reward functions, and doing the same for the "Tree" approach allows us to compare the two scheduling alternatives.

The choice of α and β defines the application utility associated with each observation. Consider the case where $\beta = 0$ and $\alpha = 1$. Collecting low resolution images serves no purpose to the application. However, low resolution

[2] https://code.google.com/p/sensoractuation/

[3] One day was used for training the model, and the other for evaluating the different schedulers.

images provide information about the state of the system. This, in turn, increases the likelihood of the scheduler to make the right call and satisfy the application needs. In Fig. 3 we compare different scheduling alternatives in our camera network where each camera has seven possible zoom regions. For different application utility functions in which we increase the utility of a low resolution image. When the utility of a low resolution image is relatively high (over 0.3) the alternatives which are sensitive to the value of β are equivalent, basically reducing to a simple scheduler which constantly actuates all cameras to zoom out. For the "hard" cases, where the utility of a low resolution image is low, looking ahead for 2 seconds out performs all other alternatives and accomplishes the highest total utility. Notice that our approximated approach performs very closely to the exact approach, but with real-time latencies[4].

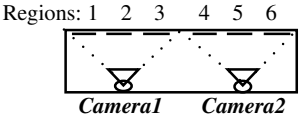
Diagnostic experiments: To illustrate the action selection process for different values of β where $\alpha = 1$, consider Fig. 4(b). We consider a single camera with 4 regions and solve the look ahead problem for 10 seconds ($\epsilon = 0.3$). We plot the fraction of states for which the best action is to zoom out (go "UP") as a function of β, the utility of a zoomed out event, where the utility of a zoomed in event is 1 (α=1). This table illustrates how the scheduler changes its behavior based on the characteristics of the reward function. The higher the utility of a low resolution image, the more likely the scheduler is to zoom out. The scheduler automatically selects the best action to take based on the specific $\alpha, \beta \in \mathbb{R}^+$ taking both long and short term benefits.

To illustrate the impact of computing the exact utility value on the latency of the scheduler, consider Fig. 4(e). We plot the latency in milliseconds of the total time that it takes the scheduler to actuate all cameras, when the decision process is performed by our approach and by a tree based approach (alternative 6 above). We vary the number of regions in each camera from 2 to 7 and thus the total number of regions in the camera network varies from 12 to 42. The latency of the tree based approach, even for a modest lookahead of 2 seconds, is not real-time and significantly higher than our approximated approach which takes less than 2 milliseconds for the largest case of 42 regions.

In Fig. 4(g) we plot the rate at which a high resolution image was collected but there was no entity in the region. We compute the ratio of high resolution images with a face over the total number of high resolution images collected. We have defined the rate to be 1 if no high resolution images were collected. This experiment captures the need for a low rate of "false-positives". Since the captured observations are later processed by a heavy operator (e.g., face recognition), we would like to minimize the number of frames that it has to process. The lookahead approach both collects a larger total of observations of interest to the application and reduces the number of false positives as compared to the other alternatives. Our approximated approach is comparable to the tree based approach and both lookahead alternatives vastly dominate other alternatives. As opposed to the tree based approach, our proposed approach is suitable for a sensor actuation setting, in which decision making has to be done in real-time latencies.

Multiple pervasive computing applications: To evaluate performance in the presence of multiple applications, we have created 11 scenarios in which there are 10 applications running in parallel each with a different reward function, as follows: 1) 10 apps with (α=1, β=0). 2) 9 apps (α=1, β=0) and 1 with (α=1, β=1). and so on until scenario 11) in which we have 10 apps with (α=1, β=1). The 11 cases are optimally solved by the following policies: 1) α=1, β=0. 2) α=1, β=0.1. and so on until 11) α=1, β=1. This enables us to compare two alternatives: Alternative 1: solve using a single optimal lookahead table for all apps (e.g., 9 apps with α=1, β=0 and 1 app with α=1, β=1 solved optimally with α=1, β=0.1). Alternative 2: as described by Alg. 2, approximate the solution by using the individual lookahead tables of each app (e.g., 9 tables with α=1, β=0 and 1 table with α=1, β=1) and choose the action with the maximal expected utility. These 11 application reward values have selected so that we could easily compare our approximation to a single optimal solution, in this case, alternative 1. In Fig. 4(f) we present our results. The results show that our approach is very close to the optimal solution, and thus validates that an approximated approach can be used to schedule multiple applications at the same time. Notice, that in the general case of the problem, simply reducing to a single lookahead table may not be possible and thus a principled approach to combine multiple applications is required.

Scalability in a large network: To examine performance as a function of network size, we have simulated a network of $K = 10$ to 100 cameras, monitoring a (very) long hallway. The figure below illustrates the case for $K = 2$ cameras.



Each camera has 3 zoom regions and people walk in a single direction from left to right. We have simulated the walk trajectories generated by 10 people who arrive at the entry point of the pipeline according to a Gaussian distribution with mean of 100 seconds and standard deviation of 25 seconds. Each person walks all the way to the end of the hallway at their own walk speed, specified as the time it takes the person to walk from region to region, which is also a Gaussian with mean of 4 seconds and variance of 2 seconds. Our scheduling algorithm automatically learnt the transition probabilities from a generated script used as training data.

[4]In expectation tree lookahead should be optimal for all values of β. However, we evaluate our scheduler on a large but finite set of data. This causes small fluctuations.

We have experimented with an application utility interested in high resolution facial images ($\beta = 0$, $\alpha = 1$). We plot the latency of reaching a scheduling decision as a function of K in Fig. 4(c) and the percentage of recalled faces in Fig. 4(d). The latency for a network of $K = 100$ cameras is less than 30 ms and linearly increasing with the size of the camera network, which makes the scheduler good fit for real-time applications. The recall of the faces remains high as the number of cameras increases. In the simulated results the recall of events is around 70% due to the fact that in about 1 of 3 cases the person will start moving and the scheduler will miss a single frame until it catches up in one of the following zoom regions.

V. Related Work

To the best of our knowledge, we are the first to address the problem of actuating a large scale camera network based on unscripted human activities. We accomplish this by using a semantic model of building activities and approximating the expected utility of actions. As opposed to problem formulation in which sensors are selected given processing and resource constraints[14] we focused on sensor actuation wherein the physical constraints of the sensor prevents capture and propose a general framework for sensor actuation.

Sensor actuation has been previously studied in specific settings. E.g., in the context of **people and object tracking** there exists a large body of work, we point the interested reader to the following surveys [18], [5]. The most relevant work to our is by [17] who suggests a POMDP approximation approach to address a single target tracking using a sensor network, where the goal is to conserve sensor battery life by querying only sensors that are likely to improve the location estimate of the target. Although the problem domain and formulation are similar, our approach differs from theirs in several key aspects: first, the scale of the problem in our case is different. [17] assumes that there is a single target in the system which makes the state space linear with the number of regions. In our case we track multiple targets not just one and address the exponential nature of the state of system as well as the state transition function. Second, [17] assumes that people walk according to a linear Gaussian model, while our formulation learns it directly. Third, in our case the policy is computed offline and actions are selected by an efficient lookup in a state-action map. In their case, the utility of action needs to be computed at real time which makes the scheduler less agile and less likely to meet real-time deadlines, even for a single target. In the context of sensor networks other ways for modeling the problem were proposed, for example for energy efficient data collection from sensor networks, recent work by [4] applies a Q-learning [16] technique to allow each sensor to self schedule its tasks an allocate its resources by learning their utility in any given state. The main advantage of Q-learning is that it does not require a model of the environment. In our case, we would like to utilize the motion characteristics of our monitored space. Furthermore, due to the size of the state space we are prevented from learning the utility of different states individually.

Unlike these previous papers, this paper takes a different view. Specifically, it addresses partially observability of the phenomena. POMDP has been also studied extensively in the literature, but the scale of the problems that it can solve is orders of magnitude smaller than the problem we have at hand.

The most relevant theory to our work is on the tractability of POMDP solutions. A number of exact value iteration algorithms have been proposed [3], [10]. None of these alternatives can be used in our context since they are limited to a very small state space. For our needs we are not interested in finding the exact solution. Our approach avoids the exponential graph in state space by approximating the scheduler belief space using a single probability value for each region of the space, instead of a probability value for each state of the world. The difference is linear (in our case) vs exponential. Second, policy search methods have also been used to optimize POMDP solutions [9], [1], Their strength lies in the fact that they restrict their optimization to reachable beliefs[10]. Much like the value iteration techniques, policy search methods require the belief state to be fully represented and for any reasonably-sized camera network the belief state space is too large. Third, approximate value iteration algorithms were presented. These techniques consist mainly of grid-based methods[6]. They can solve larger problems (90 states[10]) by updating only values at discrete grid points. Our approach is different in the way that it takes into account the specific characteristics of the monitored environment and the reward function that makes our approach practical for this problem setting.

A significant body of relevant work has recently emerged under the term "Network Distributed POMDP" [15]. ND-POMDP literature addresses the challenges that arise when two agents need to work in coordination. As an example, consider an application which is only interested in two frontal images of the same person from two different views at the same time (e.g., to reconstruct a 3D image). ND-POMDP can be used to find an optimal solution for this case. The state of the system as well as the system transition function is represented explicitly which renders DP-POMDP not tractable for the size of the problems that our approach addresses. As part of future work we are interested in addressing cameras with overlapping regions and joint camera reward functions. Applying techniques from ND-POMDP together with our approximation techniques seems like a promising direction.

VI. Conclusions and Future Work

We have developed a POMDP based framework to actuate sensors in a large scale sensor network. POMDP provides

an elegant way of representing and reasoning about the partial knowledge about the environment. However, modeling the scheduling problem in sensor networks using POMDP introduces a scalability challenge as the state space of the environment is significantly larger than typical problems with similar formulations. The heuristics and approximations that we followed made our scheduler tractable. The real time component is extremely lightweight and fast. And, most importantly there was significant improvement over alternative approaches. While our approach was evaluated in the context of camera based systems, the models that we have developed and the way that we use these models is much more general. For example, a very different type of sensor enabled application wherein loop sensors are used to measure traffic flows on various freeways/road network, and the data captured is used to build applications such as route planning, traffic jam determination, etc. We can apply the techniques described in this paper as follows: The phenomena that we monitor would be the traffic flow at each sensor discretized to a finite set of flow states, e.g., average speed rounded to the nearest value in 10 miles per hour intervals. The phenomena semantics capture the nature in which traffic flows are correlated between different locations. The techniques that we have developed apply to this setting and can be used in order to schedule the data collection from the different sensors in order to detect an event of interest - slow traffic flow (i.e., speed of less than 10 miles per hour). The data collection process would increase the number of slow traffic flow events collected. Furthermore, we can reason about the expected effects of actions such as controlling the delay in traffic light signals located at entry points to highways and actuate the traffic light to reduce the chance of a traffic jam.

As part of future work we would also like to consider a hybrid push/pull approach that will benefit from the advantages of computation at the sensors while meeting the deadlines of a real-time system and supporting the dynamic nature of a surveillance task.

REFERENCES

[1] J. Baxter, P.L. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(1):351–381, 2001.

[2] J.L. Bentley. K-d trees for semidynamic point sets. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197. ACM, 1990.

[3] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1023–1023, 1995.

[4] M. Di Francesco, K. Shah, M. Kumar, and G. Anastasi. An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks. *Wireless Sensor Networks*, pages 322–337, 2010.

[5] D.A. Forsyth, O. Arikan, L. Ikemoto, J. O'Brien, and D. Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Foundations and Trends® in Computer Graphics and Vision*, 1(2-3):77–254, 2005.

[6] W.S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operations research*, pages 162–175, 1991.

[7] A.W. Moore. An intoductory tutorial on kd-trees. *Extract from Andrew Moore's PhD Thesis: Effcient Memory based Learning for Robot Control*, 1991.

[8] Kevin P. Murphy and Yair Weiss. The factored frontier algorithm for approximate inference in dbns. In *UAI '01*, pages 378–385, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[9] A.Y. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[10] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International joint conference on artificial intelligence*, volume 18, pages 1025–1032, 2003.

[11] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

[12] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[13] R. Vaisenberg. Towards adaptation in sentient spaces. *PhD Thesis*, 2012.

[14] R. Vaisenberg, S. Mehrotra, and D. Ramanan. Exploiting semantics for scheduling data collection from sensors on real-time to maximize event detection. In *Proceedings of SPIE. SPIE*, 2009.

[15] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a spider on a network of pomdps: Generating quality guaranteed policies. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8. ACM, 2007.

[16] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[17] J.L. Williams, J.W. Fisher, and A.S. Willsky. Approximate dynamic programming for communication-constrained sensor network management. *Signal Processing, IEEE Transactions on*, 55(8):4300–4311, 2007.

[18] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 38(4):13, 2006.