Technischer Bericht 2012/05

Towards Optimized Public Sensing Systems using Data-driven Models

Damian Philipp, Jarosław Stachowiak,
Frank Dürr, Kurt Rothermel

Institut für Parallele und Verteilte Systeme
Universität Stuttgart

Universitätsstraße 38
70569 Stuttgart

August 2012

Towards Optimized Public Sensing Systems using Data-driven Models

Damian Philipp, Jarosław Stachowiak, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
Email: firstname.lastname@ipvs.uni-stuttgart.de

***Abstract*—The proliferation of modern smartphones has given rise to Public Sensing, a new paradigm for data acquisition systems utilizing smartphones of mobile users. In this paper, we present DrOPS, a system for improving the efficiency of data acquisition in Public Sensing Systems. DrOPS utilizes a model-driven approach, where the number of required readings from mobile smartphones is reduced by inferring readings from the model. Furthermore, the model can be used to infer readings for positions where no sensor is available. The model is directly constructed from the observed phenomenon in an online fashion. Using such models together with a client-specified quality bound, we can significantly reduce the effort for data acquisition while still reporting data of required quality to the client. To this effect, we develop a set of online learning and control algorithms to create and validate the model of the observed phenomenon and present a sensing task execution system utilizing our algorithms in this paper. Our evaluations show that we obtain models in a matter of just hours or even minutes. Using the model-driven approach for optimizing the data acquisition, we can save up to 38 % of energy for communication and provide inferred temperature readings for uncovered positions matching an error-bound of 1 $^\circ C$ up to 99 % of the time.**

I. Introduction

The proliferation of modern smartphones such as the Apple iPhone or Google Android Phones has given rise to Public Sensing (PS), a new paradigm for data acquisition systems utilizing smartphones of mobile users [6], [8]. These devices are equipped with various sensors such as accelerometers, cameras, light sensor, and positioning sensors like GPS. Together with their capability of processing and communicating captured sensor data, smartphones have become powerful networked sensor platforms that can be used to obtain sensor data without the cost of managing a dedicated fixed sensor network.

However, due to the mobility of users, capturing sensor data at certain points of interests (POI) is more challenging than for fixed sensor networks where a sensor could simply be installed at each POI. In [22], Philipp et al. introduced the concept of *virtual sensors* as a mobility-transparent abstraction of the PS system. Similar to the sensors of a fixed sensor network, virtual sensors can be placed at POIs and report readings with a certain client-defined sampling rate. The PS system is responsible for selecting suitable mobile devices to capture data for virtual sensors such that the client-defined quality specifications are fulfilled.

Although the concept of virtual sensors provides a user-friendly abstraction, an open problem is how to deal with virtual sensors without a mobile device in their vicinity (unavailable virtual sensors). In particular for lower densities of mobile devices and higher requested sampling rates of virtual sensors, this becomes and inherent problem of PS.

Another inherent problem of PS is the energy required for sensing, processing, and communicating data. Since PS utilizes mobile devices of users, the energy spent for PS must be kept to a minimum or users might not be willing to support PS. In other words, unnecessary sensor readings should be avoided.

In this paper, we tackle both of these problems by proposing a *model-driven approach for PS*. The basic idea of model-driven PS is to infer readings of virtual sensors from data captured by mobile devices, based on a previously learned model. On the one hand, using a model-driven approach, we can alleviate the problem of unavailable virtual sensors by inferring missing readings. On the other hand, we can avoid unnecessary readings by actively omitting readings of virtual sensors whose values can be inferred.

Model-driven sensing has already proven to increase the performance of fixed sensor networks and actuated sensing systems [7], [12], [27]. These approaches target long-running sensing tasks, and use either expert knowledge or a pilot deployment to obtain the model a priori. Although these are reasonable approaches for long-running tasks, PS requires a different approach to support queries for virtual sensors over shorter periods of time that can be defined spontaneously by the client. Therefore, we propose an approach for obtaining models *online* within short time. Following the general goal of saving energy, we additionally optimize the acquisition of models for low energy consumption by keeping valid models as long as possible.

In more detail, the contributions of this paper are:

1) An online learning algorithm for models of spatially distributed phenomenons that can obtain training data from running queries and quickly creates a model suitable for model-driven sensing. To minimize energy as well as the time to learn a model, we try to re-use historic readings and only update portions of the model if possible.
2) An online model validity check algorithm (MOCHA) that verifies whether a model still accurately reflects the underlying phenomenon. In order to minimize the en-

ergy spent for checking the validity, MOCHA uses only a small fraction of additional control readings, which are continuously monitored and compared to inferred readings. Moreover, MOCHA avoids abandoning models too quickly by applying smoothing techniques.

Our evaluations show that we obtain models in a matter of just hours or minutes. Using our model-driven approach, we can save up to 38 % of energy for communication and provide inferred readings for unavailable positions matching an error-bound of $1\,^{\circ}C$ up to 99 % of the time.

The remainder of this paper is structured as follows. We present the system model and problem statement in Section II before discussing the basic operation of the DrOPS system in Section III. Section IV details the optimized operation and gives a brief introduction to the spatial models used in DrOPS, while Section V presents our online learning and validation algorithms. We evaluated the DrOPS system in a simulated environment, whose methodology and results are discussed in Section VI. Related work is discussed in Section VII, before we conclude this paper with a short summary in Section VIII.

II. System Model and Goals

Before introducing the system model, we first define some terminology. We will refer to smartphones carried by users as *mobile nodes*. A *virtual sensor v* is a point in space for which our system should provide a measurement. Each virtual sensor has a coverage area *area(v)*, comprising a set of positions. *area(v)* can be defined arbitrarily, as long as the coverage areas of all virtual sensors are pairwise non-overlapping to ensure a unique mapping of mobile nodes to virtual sensors. If there is at least one mobile node located in *area(v)* to capture sensor readings for *v*, we say that *v* is *available*. Otherwise, *v* is *unavailable*. The set of all unavailable virtual sensors is denoted as Unavailable*(V)*.

For the sake of simplicity, we assume a circular model based on Euclidean distance $\delta()$ for the coverage area of virtual sensors, using $v.\delta_{max}$ as the radius of the coverage area. So, for a set of virtual sensors *V* we define individual coverage areas as $\forall v \in V : pos \in area(v) \Leftrightarrow \delta(pos, v) \leq v.\delta_{max}$.

Virtual sensors provide measurements as virtual readings, which can either be *effective readings* or *inferred readings*. An effective reading *r* is taken by a mobile node at position $pos(r) \in area(v)$, whereas an inferred reading is computed using a model of the observed phenomenon.

In order to define which virtual sensors should report effective readings and for which virtual sensors readings should be inferred, we split a set of virtual sensors *V* into subsets V_{eff} and V_{inf}, where $V = V_{eff} \cup V_{inf}$, $V_{eff} \cap V_{inf} = \emptyset$. It should be noted that virtual sensors $v \in V_{eff}$ may turn out to be unavailable. In this case, no effective reading will be available for *v*. We therefore define $V^{-}_{eff} = V_{eff} \setminus \text{Unavailable}(V)$ as the (reduced) set of effective sensors that are actually available. $V^{+}_{inf} = V_{inf} \cup \text{Unavailable}(V)$ denotes the extended set of virtual sensors returning inferred readings either intentionally (set V_{inf}) or due to being unavailable.

The DrOPS system consists of a gateway service and a set of mobile nodes. The gateway service serves as an interface to submit queries to the system and to store, browse, and return historic data. To ensure the scalability of the system, the geographical area for which the system can service requests for data is partitioned into a set of service areas. Each service area is serviced by a gateway server, which is responsible for tracking which nodes are located in its service area, and for communicating with these nodes.

Mobile nodes can determine their position using a positioning system such as GPS, and use a set of environmental sensors (light, sound, temperature, air pollution, etc.) to observe their environment. We assume that all nodes are equipped with the appropriate sensor hardware to provide readings for all data requests occurring in the system and that readings are provided instantly upon accessing a sensor. Furthermore, nodes are equipped with a 3G interface allowing for communication with the gateway server whose service area contains the mobile nodes position. DrOPS does not make any assumption about the mobility of nodes, which is in general uncontrollable in a PS system.

The DrOPS system accepts queries *R = (V,y,p,Q)* issued by clients, where *V* represents the set of virtual sensors, *y* denotes the type of the reading to be taken, e.g., light, temperature, and *p* is the sampling period. *Q* is a set of quality parameters required by the client. The content of *Q* depends on the selected algorithms and, therefore, will be explained in detail in the corresponding sections. At the end of every sampling period, a result set FR_V is sent to the client. FR_V contains all effective readings for virtual sensors in V^{-}_{eff}. Depending on the selected algorithm, it may also contain inferred readings for virtual sensors in V^{+}_{inf}.

Our goal is to create a system for obtaining data on spatially distributed environmental phenomena according to several quality and optimization objectives. First, the system should maximize the number of virtual sensors for which readings are reported, i.e., compensate for unavailable virtual sensors by inferring readings using a model of the observed phenomenon. Second, it should optimize the data acquisition process, i.e., increase efficiency by minimizing the number of effective readings while still returning data that meets application-defined quality constraints, given as part of the quality parameters *Q*.

More precisely, to optimize data acquisition we will select a set of virtual sensors $V_{eff} \subseteq V$, minimizing $|V_{eff}|$ under the constraint that $\forall u \in V_{inf} : |i_u - r_u| \leq Q.T$. Here, r_u represents the real value at virtual sensor *u* while i_u is the inferred reading for *u*. Note that we cannot guarantee that the quality constraints are always met, for instance, if certain areas are not covered by mobile nodes. However, we aim to minimize the number of violations of the above constraint.

III. Basic Sensing Algorithm

Next, we explain the basic execution of queries, before we describe the optimized model-driven operation and the process of model maintenance in subsequent sections. The

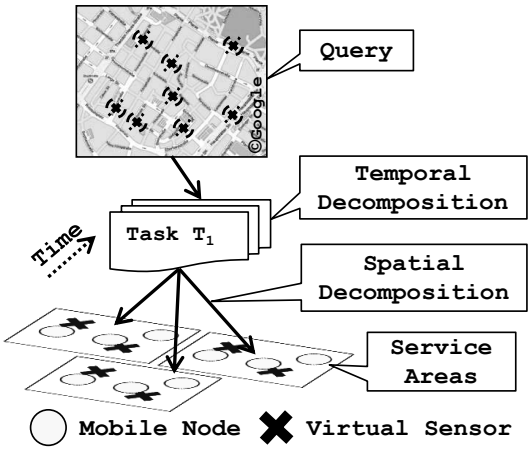


Figure 1: Overview of Sensing Task Execution

basic sensing algorithm is used to illustrate the operation of DrOPS and will later serve as a baseline for comparing the performance of our optimized approaches.

To issue a query *R = (V,y,p,Q)*, the client submits it to the gateway service. The gateway then performs a *temporal decomposition* of the query by generating a new task $T = (V, y, Q)$ every p seconds at the beginning of each sampling period until the client cancels the query. Note that for the basic algorithm, $V = V_{eff}$ and $V_{inf} = \emptyset$, i.e., all sensors should return effective readings, since the basic algorithm does not use inference to compensate for unavailable sensors or for optimization. For each T, the gateway performs a *spatial decomposition* by creating one sub-task $s_A = (V^A_{eff}, y, Q)$ per service area *A*, where $V^A_{eff} \subseteq V$ denotes the virtual sensors located in A for which effective readings should be taken. Each s_A is then sent to all mobile nodes in A. This process is illustrated in Fig. 1. Note that we do temporal decomposition first to later allow for a global optimization of task execution over all service areas.

On receiving a task, each node determines whether it is located within the coverage area of any virtual sensor $v \in V'$. If so, it takes an effective reading *r* and returns $(v, r, pos(r))$ to the gateway. Otherwise, it discards the query. In case there are multiple effective readings taken for *v*, the gateway only selects the reading $\overline{r}$ with minimum $\delta(pos(\overline{r}), v)$. The gateway stores all selected readings together with the starting time of the corresponding sampling period. At the end of each sampling period, the effective readings for virtual sensors $v \in V^-_{eff}$ is returned to the client as result FR_V. No data is returned for other virtual sensors.

In the following sections, we extend this basic sensing algorithm with our model-driven approach to compensate for unavailable virtual sensors and minimize the number of necessary effective readings.

IV. Model-Driven Optimization

The basic sensing algorithm presented in the previous section has two problems, which we solve with our model-driven approach. First, the basic sensing algorithm cannot obtain readings for unavailable virtual sensors. In these cases, DrOPS can use its model of the observed phenomenon to infer readings for these virtual sensors. Second, we can use the same model to identify strongly correlated virtual sensors. We use this to minimize the size of V_{eff}, thus, reducing the effort required for task execution.

Since the focus of DrOPS is to obtain data for spatially distributed environmental phenomena, we need to select an appropriate model for this data. One kind of mathematical model that has been identified as suitable for general spatially distributed environmental phenomena is a multivariate Gaussian distribution (MGD). Note that capturing other phenomena, e.g., discrete environmental phenomena such as individual events (e.g., lightning strikes), may require a different model or a different optimization approach altogether. We will briefly introduce the structure of an MGD and show how to apply MGDs to infer and optimize readings in principle, before we present an extended version of DrOPS based on these principles. For more in-depth information about the application of MGDs, we refer to [7], [12], [17].

A. Multivariate Gaussian Distribution

In an MGD, each virtual sensor v is modeled using a one-dimensional Gaussian distribution with mean μ_v and variance σ^2_v. Individual distributions for virtual sensors v, v' are dependent on each other. This dependency is expressed in a covariance matrix $\Sigma_{V,V}$ where each entry $\sigma_{v,v'}$, $v \neq v'$ signifies the covariance of the values for virtual sensors v and v'. Consequently, the entries on the diagonal $\sigma_{v,v} = \sigma^2_v$ represent the variance of each individual distribution. We define M_V to be the vector of mean values of each individual distribution and write a multivariate Gaussian distribution over the virtual sensors in V as $MGD_V = (M_V, \Sigma_{V,V})$.

Given an incomplete vector of effective readings P_W from virtual sensors $e \in W \subset V$, we can use MGD_V to infer the missing values for virtual sensors $u \in U = V \setminus W$ based on the following equations:

$$\mu_{u|P_W} = \mu_u + \Sigma_{u,W}\Sigma^{-1}_{W,W}(P_W - M_W) \quad (1)$$

$$\sigma^2_{u|P_W} = \sigma^2_u - \Sigma_{u,W}\Sigma^{-1}_{W,W}\Sigma_{W,u} \quad (2)$$

In these equations, $\Sigma_{U,W}$ denotes the covariance matrix reduced to the entries corresponding to $\sigma_{u,w}, u \in U, w \in W$.

The output of this inference are not exact individual values but rather one-dimensional Gaussian distributions encoding the estimate about the current value at each u. Note that both the inferred mean $\mu_{u|P_W}$ and the variance $\sigma^2_{u|P_W}$ depend on the set of virtual sensors *W*, but only $\mu_{u|P_W}$ depends on the actual values in P_W.

In the following, we will also refer to this process as function Infer(U, MGD_V, P_W).

B. Near-Optimal Sensor Selection

Using Infer, we can infer readings for unavailable virtual sensors, on the one hand. On the other hand, we can also use an MGD model to optimize the data acquisition process by explicitly moving virtual sensors from V_{eff} to V_{inf} and

Algorithm 1 Modified GREEDY algorithm for selecting V_{eff} [12].

Require: $T = (V, y, Q)$, $MGD_V = (M_V, \Sigma_{V,V})$
$V_{eff} \leftarrow \emptyset, V_{inf} \leftarrow V$
while $(\exists v \in V_{inf} : \sigma^2_{v|V_{eff}} > Q.\sigma^2_{max}) \wedge (V_{inf} \neq \emptyset)$ **do**
  $I_{max} \leftarrow 0$
  $w \leftarrow \perp$
  for all $u \in V \setminus V_{eff}$ **do**
    $I_u \leftarrow$ mutualInformation$(u, \Sigma_{V,V}, V_{eff})$
    if $I_u > I_{max}$ **then**
      $w \leftarrow u$
      $I_{max} \leftarrow I_u$
    end if
  end for
  $V_{eff} \leftarrow V_{eff} \cup \{w\}, V_{inf} \leftarrow V_{inf} \setminus \{w\}$
end while
return V_{eff}

then inferring their values. The rationale behind this is that for sets of strongly correlated virtual sensors, it is sufficient to request effective readings for a small subset of these to still yield accurate inferred readings (i.e., readings with a variance below a given threshold σ^2_{max}) for all virtual sensors.

As stated before, we strive to minimize the size of set V_{eff} in order to achieve the best optimization. Guestrin et al. have shown that this problem is NP hard [12] and have developed a set of near-optimal heuristic algorithms to address this problem. We use a slightly modified version of their GREEDY algorithm, presented in Alg. 1. Given a task *T=(V,y,Q)* and a covariance matrix $\Sigma_{V,V}$, the GREEDY algorithm selects $V_{eff} \subseteq V$. Note that the gateway is not aware which virtual sensors are available. Therefore, selecting V_{eff} is done in an optimistic fashion, assuming that they will be available.

Our variant of GREEDY selects effective sensors as follows. Initially, $V_{eff} = \emptyset$ and $V_{inf} = V$. Sensors are moved to V_{eff} in sequence, where each time the sensor with the highest mutual information I_{max} is selected, i.e., the sensor that most significantly reduces the uncertainty about the inferred readings for virtual sensors remaining in V_{inf}. For a formal definition of the mutual information criterion, see the original publication [12].

Whereas the original algorithm limited the set of virtual sensors to a fixed size, we use a target maximum variance $Q.\sigma^2_{max}$ provided as a quality parameter. The modified sensor selection algorithm will then add as many sensors as necessary to V_{eff} to ensure that the variance of any virtual sensor is less or equal to $Q.\sigma^2_{max}$.

C. Optimized Sensing Algorithm

Given an MGD and a sensing task $T = (V, y, Q)$, we modify the operations of DrOPS to use the model-driven optimization as shown in Alg. 2. At the beginning of each sampling period, we use the modified GREEDY-Algorithm to select a set $V_{eff} \subseteq V$ of virtual sensors (line 2).

Algorithm 2 Model-driven sensing task execution

Require: $R = (V, y, p, Q)$, $MGD_V = (M_V, \Sigma_{V,V})$
for all $task \in$ temporalDecomposition(p) **do**
2: $V_{eff} \leftarrow$ GREEDY$(V, \Sigma_{V,V}, Q.\sigma^2_{max})$
  $task.V \leftarrow V_{eff}$; $readings \leftarrow \emptyset$
4: **for all** $subtask \in$ spatialDecomposition$(task)$ **do**
    $readings \leftarrow readings \cup$ execute$(subtask)$
6: **end for**
  $FR_V \leftarrow readings \cup$ INFER$(V^+_{inf}, MGD_V, readings)$
8: **return** Final Result FR_V
end for

The task is then executed in lines 3 to 6 as in the basic algorithm. At the end of the sampling period, after collecting all effective readings at the gateway, the gateway performs the inference step. Using INFER, readings for virtual sensors in V^+_{inf} are inferred from the available effective readings in line 7. For mimicking a classic sensor network, we output the inferred mean values for each $v \in V^+_{inf}$ as an inferred reading. If a client demands additional information, we also include the variances in the final result FR_V.

V. MODEL MANAGEMENT

To use the optimized sensing algorithm, an accurate model of the observed phenomenon must be available. As already motivated in Section I, PS requires such a model to be available quickly whereas existing approaches use long-term training periods to learn a model. For instance, consider a temperature monitoring application. Intuitively, the temperature will rise in the morning and drop in the afternoon. Existing approaches take several days of training data to create a model that accurately reflects this shift in temperature over the day. Such a long training period is much too long for a PS application that spontaneously requests current temperature values for virtual sensors at certain POIs.

Therefore, the basic idea of our approach is to quickly derive a model of the observed phenomenon on demand that is sufficiently accurate for the near future using an *online learning algorithm*. Since such a model might not be able to reflect changes happening over a longer time period—such as the temperature rising in the morning and falling in the evening in the previous example—, we continuously monitor model accuracy using an *online model validity check algorithm (MOCHA)* to learn a new model when the old one becomes too inaccurate.

Next, we start by presenting our online model validity check algorithm MOCHA, before we present our online learning algorithms.

A. MOCHA

The goal of MOCHA is to check a model for correctness. Intuitively, a model is correct if the Gaussian distributions of inferred readings fit the real data, i.e., inferred values from virtual sensors in V^+_{inf} center around the true mean value and the variance matches the true variance. However,

Algorithm 3 MOCHA algorithm. $\mathcal{C}$ denotes the set of virtual sensors used for control readings.

Require: Final Result FR_V, Control Readings $C_{V_{ctl}^-}$, Threshold $Q.T$, Acceptable Violations $Q.violations$

RMSE $\leftarrow 0$
for all $c \in V_{ctl}^-$ **do**
  RMSE $\leftarrow$ RMSE $+ (FR_c - C_c)^2$
end for
RMSE $\leftarrow \sqrt{\frac{\text{RMSE}}{|V_{ctl}^-|}}$
if RMSE $\geq Q.T$ **then**
  Add "violation" to window
else
  Add "no violation" to window
end if
if Number of violations in window $\leq Q.violations$ **then**
  return "Model Valid"
else
  return "Model Invalid"
end if

checking this property for every virtual sensor in V_{inf}^+ is inefficient or impossible. On the one hand, this would require constant sampling of all virtual sensors, i.e., $V_{eff} = V$, which would render the whole optimization useless. On the other hand, inferred distributions are not constant since they depend on values and sources of effective readings, which may be different in every sampling period.

Therefore, we take a different approach for MOCHA (see Algorithm 3). At the beginning of each sampling period, we randomly choose a set of control sensors $V_{ctl} \subseteq V_{inf}$ of size *Q.control*. In addition to the virtual sensors V_{eff} selected by the GREEDY algorithm, we request effective readings for virtual control sensors in V_{ctl}. At the end of the corresponding sampling period, only effective readings from virtual sensors in V_{eff}^- are used as input for the inference algorithm. We then compare the inferred readings for available virtual control sensors in $V_{ctl}^- = V_{ctl} \setminus \text{Unavailable}(V)$ to their corresponding effective control readings by computing the root mean squared error (RMSE, lines 1 to 5) for each virtual sensor. If the RMSE is greater than a predefined threshold $Q.T$, which is part of the extended quality specification Q of a sensing task, we consider the model to be inaccurate. $Q.T$ defines the average error for inferred readings that is still acceptable to the client. Thus, we can define RMSE $> Q.T$ to indicate an inacceptable model accuracy.

Using the RMSE, we avoid the problem of comparing individual samples to inferred distributions since we can compare absolute values directly. Furthermore, by adjusting the size of V_{ctl}, we can trade off the costs for effective sampling and the probability of detecting inaccurate models (model quality).

Immediately discarding a model when RMSE $> Q.T$ might lead to abandoning an accurate model in case the observed error was a single outlier. We therefore use a sliding window approach (lines 6 to 15) to dampen the reactivity of MOCHA. We define a model to be inaccurate if there are *Q.violations* (RMSE > Q.T) within the last *Q.window* samples. For example, for $Q.window = 2$ and $Q.violations = 1$, we will abandon the model only when observing RMSE $> Q.T$ in two consecutive sampling periods.

Using MOCHA, we next introduce our online learning algorithms.

B. Simple Online Learning Algorithm

The *simple online learning algorithm (S-OL)* creates a MGD model for the optimized sensing algorithm while a task is executed. It uses MOCHA to monitor model accuracy and creates a new model whenever the current model is discarded by MOCHA.

The runtime of a query is divided into learning phases and optimization phases. The duration of learning phases is configured to a constant time using the quality parameter *Q.learnTime*, e.g., 1 hour. During this time, queries are executed using the basic sensing algorithm, i.e., $V_{eff} = V$. At the end of the learning phase, we run an existing offline learning algorithm on the available data to create a new MGD. This includes data from the current learning phase and all previous phases (both learning and optimization) for the same task up to a certain age given in *Q.maxAge*. Note that reducing the maximum age of data reduces the runtime of the learning algorithm. However, when setting *Q.maxAge* too short, the learned model will forget global shifts in the observed phenomenon that have been encountered before. Therefore, if periodic shifts can be expected, e.g., temperature shifts in day/night cycles, *Q.maxAge* should be set to a multiple of the cycle duration.

The number of effective readings required to learn an operational MGD depends on the learning algorithm. We used the approach by Schwaighofer et al. [24], which can be configured to work with as little as two samples per virtual sensor. If the number of effective readings available for a virtual sensor *v* at the end of the learning phase is insufficient, e.g., for the approach of Schwaighofer et al. at most one effective reading newer than *Q.maxAge* is available, *v* is excluded from the model. Thus, rather than not performing any optimization at all, we set $v \in V_{eff}$ in every sampling period and no inference is done when the virtual sensor is unavailable.

After $Q.learnTime$, we switch to the optimization phase, where we use the MGD as described in Section IV to reduce the number of effective readings taken. Furthermore, we continuously run MOCHA to monitor the accuracy of the current MGD. When MOCHA considers the current MGD to be inaccurate, we switch back to the next learning phase.

C. Improved Online Learning Algorithm

The *adaptive online learning algorithm (A-OL)* improves the learning phase of S-OL w.r.t. two issues of S-OL. First, the learning phase in S-OL is of constant duration. However, the optimum duration is dependent on the availability of virtual sensors. Ideally, the learning phase is terminated when a sufficient number of samples has been gathered. Therefore,

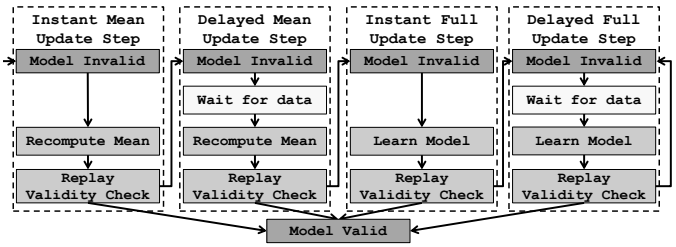


Figure 2: Optimized learning phase of A-OL

Algorithm 4 Instant Mean Update Step

Require: $R = (V, y, p, Q)$, $MGD_V = (M_V, \Sigma_{V,V})$
$\forall v \in V : M'_V[v] = \perp$
2: $W \leftarrow \emptyset$
for all $v \in V$ **do**
4: $data \leftarrow$ getHistoricData$(v, [Q.maxAge, \text{now}()]))$
if $|data| > 0$ **then**
6: $M'_V[v] \leftarrow$ mean$(data)$
else
8: $W \leftarrow W \cup \{v\}$
end if
10: **end for**
$M'_V[W] \leftarrow$ INFER(W, MGD_V, M'_V)
12: **return** $MGD'_V = (M'_V, \Sigma_{V,V})$

A-OL adapts the length of the learning phase to the number of gathered effective readings.

Second, in S-OL the complete model is recomputed from scratch in every learning phase. However, in many cases the covariance matrix is still correct while only the mean vector is inaccurate due to some global shift that has not been observed before, e.g., temperature rising in the morning while a query was started the night before and has only seen low night-time temperatures. Therefore, A-OL uses a model update mechanism that adjusts only the mean vector whenever this is sufficient.

The learning phase in A-OL is subdivided into four steps (cf. Fig. 2). At the end of every step, we replay the whole set of GREEDY, INFER, and MOCHA on data from the last *Q.window* sampling periods to compare the performance of the updated model to that of the previous model. If the new model is considered valid by MOCHA, we immediately switch to the next optimization phase using this model.

In the **instante mean update step** (Alg. 4) we keep the covariance matrix and update only the mean vector from existing data, to avoid costly requests for effective readings from all virtual sensors. We compute the mean of all effective readings newer than *Q.maxAge* that we have seen for each virtual sensor (lines 3 to 10). If no effective reading has been reported within the considered history for a virtual sensor, e.g., if the virtual sensor was never available, we use the old model to infer a mean value for this virtual sensor (line 11). This might yield a greater error in inferred readings using the new mean vector, which, however, would be detected by MOCHA.

Algorithm 5 Delayed Mean Update Step

Require: $R = (V, y, p, Q)$, $MGD_V = (M_V, \Sigma_{V,V})$
$waitStart \leftarrow$ now()
2: **repeat**
wait for next sampling period
4: $V_{eff} \leftarrow V$
request data
6: **until** $|\{v \in V || \text{getHistoricData}(v, [waitStart, \text{now}()])| \geq Q.minReadings\}| \geq Q.minSensors$ or *totalWaitTime* $>$ *Q.learnTime*
return INSTANTMEANUPDATE(R, MGD_V)

Algorithm 6 Instant Full Update Step

Require: $R = (V, y, p, Q)$
$data \leftarrow$ getHistoricData$(V, [Q.maxAge, \text{now}()])$
2: **return** learnModel$(data)$

If the new model is still invalid, we start the **delayed mean update step** (Alg. 5). In the delayed mean update step, we issue queries to all virtual sensors for effective readings and update only the mean vector. Unlike in S-OL, the time spent on obtaining fresh data is not fixed. We use an adaptive criterion, where readings are requested until at least *Q.minReadings* effective readings have been received for at least *Q.minSensors* virtual sensors each (line 6). The mean vector is then updated as in the instant mean update step (line 7). Waiting for more readings or more sensors, respectively, will yield a more accurate model but degrades the overall efficiency since more time is spent in the learning phase.

If the model is still considered invalid after the delayed mean update step, we move to updating the entire model (mean vector and covariance matrix) in the **instant full update step** (Alg. 6) using the same learning algorithm as used in the learning phase of S-OL. As input for the learning algorithm, we use the effective readings obtained in the delayed mean update step along with the available historic data. So no new data has to be gathered in this step.

If the new model remains invalid, we continue with the **delayed full update step** (Alg. 7). In this step, we again try to rebuild the complete model based on fresh data. To this end, we again request fresh effective readings from all virtual sensors. When sufficient data has arrived, we execute the learning algorithm. If the model remains invalid, we repeat this step until a valid model has been created.

There are two issues that can degrade the performance of the A-OL algorithm, the first being unavailable virtual sensors. If a fraction larger than $|V| - Q.minSensors$ virtual sensors do not report a sufficient number of effective readings, the waiting time will be infinite. The second issue is that the replay of sensor data may cause a model to be falsely considered invalid. To tackle these issues, we set *Q.learnTime* as a hard limit on the total time spent in these four steps (*totalWaitTime*). When this limit is reached, a new full model is constructed from the available data and considered to be valid without further

Algorithm 7 Delayed Full Update Step

Require: $R = (V, y, p, Q)$
$waitStart \leftarrow$ now()
repeat
  wait for next sampling period
  $V_{eff} \leftarrow V$
  request data
until $|\{v \in V || \text{getHistoricData}(v, [waitStart, \text{now}()])| \geq Q.minReadings\}| \geq Q.minSensors$ or *totalWaitTime* $>$ *Q.learnTime*
$data \leftarrow \text{getHistoricData}(V, [Q.maxAge, \text{now}()])$
return learnModel($data$)

checking. Should the new model be invalid, this is detected by MOCHA after at most *Q.window* sampling periods. Thus, the learning phase in A-OL cannot last longer than that of S-OL, but A-OL has the opportunity to return to the optimization phase much quicker.

VI. Evaluation

In this section, we briefly evaluate the performance of the DrOPS system. We will first discuss the setup of our simulation environment. Then we look at preliminary results for the performance and energy consumption of data acquisition using S-OL and A-OL.

A. Simulation Setup

We evaluated the DrOPS system using two real-world datasets: Ten days from the Intel Lab data set [10] and seven non-consecutive weeks ("epochs") of data from the Lausanne Urban Canopy Experiment (LUCE) of the SensorScope project [21]. Both data sets contain environmental readings, e.g., temperature. Data tuples (*Sensor ID*, *Timestamp*, *Value*) are reported by a set of fixed sensors. The Intel Lab data contains about 50 fixed sensors in an indoor deployment while the LUCE data contains around 100 fixed sensors in an outdoor setting. Positions of the fixed sensors are available for each dataset. Sensors report data at individual, irregular intervals. In the Intel Lab data, all intervals are a multiple of 10, 20, or 30 s. In the LUCE data, the intervals of individual sensors are not synchronized, therefore we aggregated the data into 30 s intervals. Fig. 3a and 4 show an excerpt of the temperature data from each data set, where *Real Mean* denotes the mean temperature of all virtual sensors in each sampling period.

We generate queries by placing a virtual sensor for temperature data at the position of every real sensor in each data set, basically requesting a temperature map for the service area. Thus, we can directly use the values from the underlying data sets instead of performing some interpolation. Additionally, we can directly compare inferred readings to ground truth. The sampling period is adapted to match the interval at which data is provided by the data set to get as many tasks as possible out of the data. Note that in both data sets, the data in each sampling period is incomplete. Thus, in each sampling period there is a varying number of sensors that do not report readings. In the Intel Lab data, there are an average of 20 out of 50 virtual sensors with available data in each sampling period, thus, there are relatively few effective readings per sampling period available. In the LUCE data, there are always more than 70 out of about 100 virtual sensors with available data in each sampling period.

Parameter		Intel Lab	LUCE
Error Threshold	*Q.T*	$1\,^{\circ}C$	$1\,^{\circ}C$
Max. Variance	$Q.\sigma^2_{max}$	0.3	0.1
Window Size	*Q.window*	8	1
Violations	*Q.violations*	5	0
Control Readings	*Q.control*	2	1
Learning Time	*Q.learnTime*	1 hour	1 hour
Maximum Age	*Q.maxAge*	3 days	3 days
A-OL Paramters	*Q.minReadings*	5	5
	Q.minSensors	49	98

Table I: Quality parameters used in the simulation

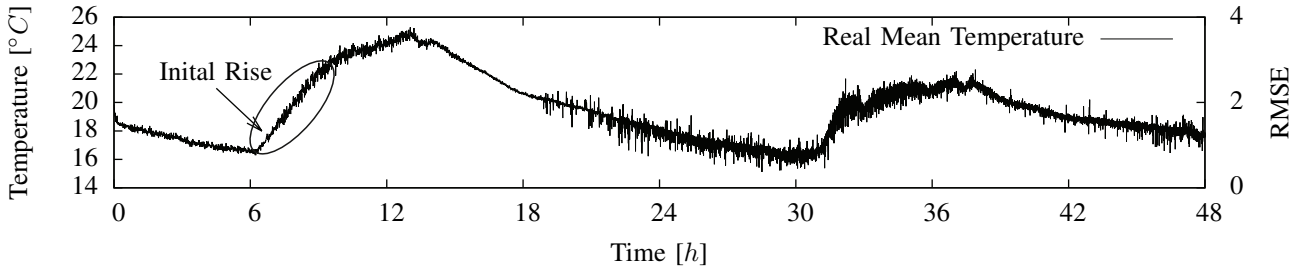
To integrate node mobility and communication, we used two simulation environments implemented in the network simulator Omnet++ [28] that differ w.r.t. the level of detail and the computational complexity. The *detailed* simulation includes fine-grained models for mobility and communications. However, it has a huge computational complexity, prohibiting its use for parameter studies. Therefore, we created the *fast* simulation, built for low computational complexity. It is used for parameter tuning and initial comparison of our algorithms. The use of the detailed simulation environment is limited to the final energy evaluation.

In the *fast simulation environment*, no communication is simulated and node mobility is taken directly from the underlying data sets: Virtual sensors are available iff there is a reading available in the underlying data set. These abstractions allow us to run each simulation for the complete duration (10 days and 1 week, respectively) of the underlying real-world data sets. We introduced a single query for all virtual sensors, lasting for the entire simulated time, in each simulation run.

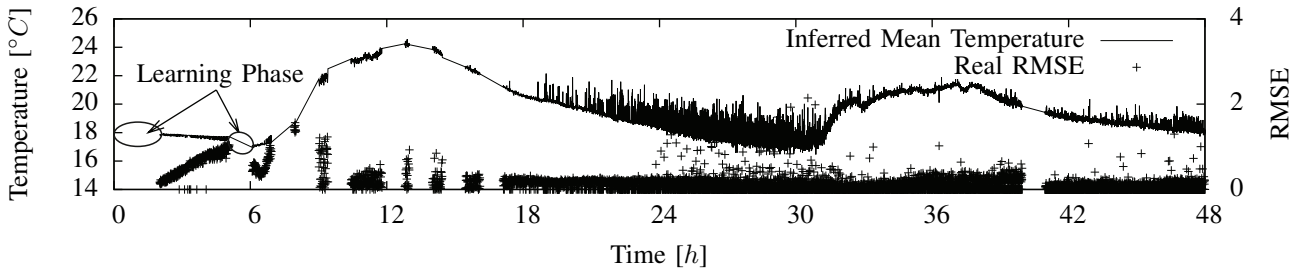
In the *detailed simulation environment*, we use the INET-MANET extension of Omnet++ and a custom 3G model to create a detailed implementation of DrOPS. Energy consumption is quantified using an empirical energy model [3].

Note that the energy model includes neither energy spent on position fixes nor energy used for sampling sensors. If we were to include energy usage for position fixes, we would merely see a constant offset to our results. For the energy consumption of environmental sensors, no consistent energy model is available. Energy usage may vary strongly between built-in sensors of different types or external sensors, e.g., connected via Bluetooth. For very cheap sensors, the energy required for sampling is insignificant compared to the communication energy. For very expensive sensors, our evaluation underestimates the possible energy savings, as we reduce the number of sensor samplings by replacing effective readings with inferred readings. Therefore, the absolute energy savings would be even higher for expensive sensors.

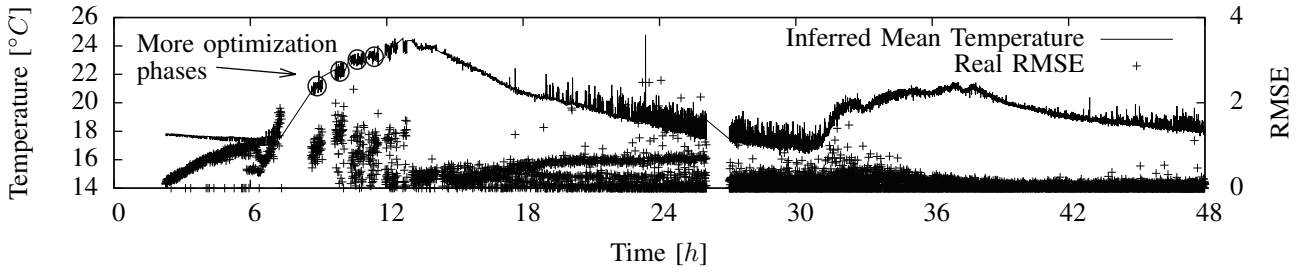
To generate node mobility for the Intel Lab data, we placed 70 mobile nodes on an abstract representation of the labs'



(a) True mean of sensor readings (ground truth)



(b) S-OL



(c) A-OL

Figure 3: Excerpt of data from the Intel Lab data set (fast simulation environment).

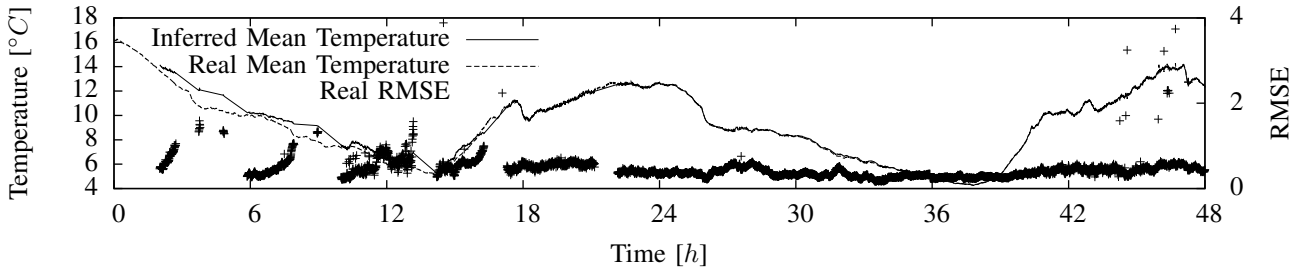


Figure 4: Excerpt of data from one epoch in the LUCE data set (S-OL only, fast simulation environment).

floor plan. Nodes move around randomly along the available paths in the lab. For the LUCE data, we used a road graph of the deployment area extracted from OpenStreetMap [13] as input for the CanuMobiSim mobility simulator [26] to generate random mobility traces for 150 nodes.

Due to the level of detail, the detailed simulations are several orders of magnitude slower than the fast simulations, making it infeasible to simulate the entire length of each data set. Instead of running one long simulation, we ran a set of short simulations in parallel. Simulations began every 3 hours from the beginning of the data file and continued for 6 simulated hours each. For the Intel Lab data set this was sufficient to allow for running the detailed simulations. The LUCE data, however, consists of several epochs and more virtual sensors, requiring more mobile nodes. Therefore, for the LUCE data, we further limited the number of simulations by only including simulations beginning within the first 9 hours of each epoch. As with the fast environment, we introduced a single query for all virtual sensors in each simulation run.

The detailed environment has a drawback causing the simu-

lation to systematically underestimate the performance of our system. In a real system, we require only that a virtual sensor is available to obtain an effective reading. However, since we are using preexisting data sets, besides there being a node in the coverage area of the virtual sensor, there also has to be a sample for that virtual sensor in the data set. Thus, the chance of obtaining effective readings is significantly reduced compared to a real system. Therefore, the amount of data available for learning a new model or inferring readings is reduced, which then degrades the accuracy of inferred readings in the detailed environment.

B. Performance Evaluation

Using the optimized parameters determined in the previous section, we now evaluate quality and efficiency of A-OL. Fig. 3 and 4 show example executions of the DrOPS system on temperature data from the Intel Lab data and LUCE data, respectively. For the Intel Lab data, the mean temperature from the underlying data set is depicted in Fig. 3a while 3b and 3c show the mean temperature as output by DrOPS along with the real RMSE of each task in an optimization phase. For the LUCE data, we limit the presentation to an example execution using S-OL and can thus show all graphs in a single plot. Note that when there are no values for real RMSE, the system is in a learning phase, otherwise it is in a optimization phase.

We can see in Fig. 3 that during the initial rise of the mean temperature, A-OL manages to rebuild the model more quickly than S-OL and thus spends more time in the optimization phases overall. Thus we will limit further analysis to A-OL. Fig. 5 shows the quality and efficiency achieved for each data set by A-OL. For the LUCE data, quality is better than for the Intel Lab data. We attribute this to the fact that for the LUCE data, no smoothing step in MOCHA is necessary (cf. Table I). Therefore, whenever inferred results start deviating from real values, DrOPS immediately switches to a learning phase. For the case of the Intel Lab data, smoothing was necessary, as many individual outliers occurred. In total, these outliers as well as readings taken during the time it takes to fill the smoothing window degrade the quality.

When looking at the plot for efficiency, we can see that for the LUCE data, efficiency is lower than for the Intel Lab data. This is likely caused by the fact that in the LUCE data the temperature values are shifting very quickly, sometimes irregularly, whereas the values in the Intel Lab data are comparatively smooth. Thus, models for the LUCE data are valid for shorter time periods, i.e., more time is spent in learning phases.

C. Energy Evaluation

We analyze the energy consumption of task execution in the DrOPS system using the detailed simulation environment. Fig. 6 shows the result of our simulations. The energy consumption of each node was normalized to the average energy consumption of nodes using the basic approach. We can see that DrOPS greatly increases the energy efficiency of data acquisition.

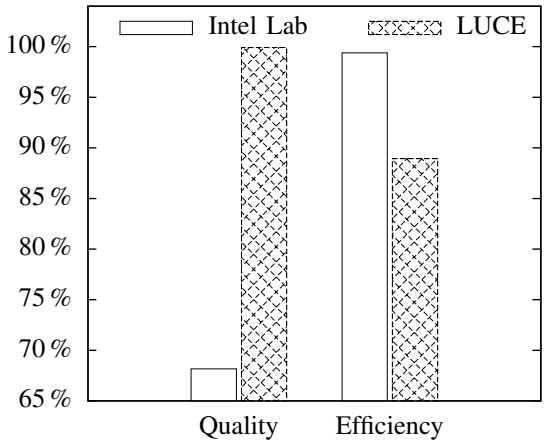


Figure 5: Performance of A-OL under both Intel Lab data and LUCE data.

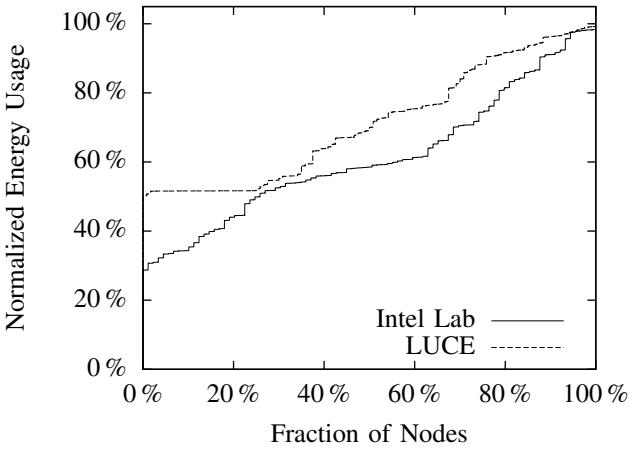


Figure 6: Normalized cumulated energy usage of nodes

To quantify the energy savings, we calculated the total energy used by all nodes in all simulation runs for each approach. Compared to the basic approach, A-OL requires 70 % of energy for the LUCE data and only 62 % for the Intel Lab data. Thus, we can save up to one third of the energy compared to the basic approach.

VII. Related Work

Research interest in Public Sensing has been growing over the last few years [6], [8]. Several prototype systems and system architectures have been proposed. CenceMe [20] uses activity recognition techniques to share data about the current state of a participating person to social networks. MobGeoSen [15] is a system architecture for environmental monitoring applications. CarTel [14] focuses on car-based sensor networks making observations about traffic and road conditions. MetroSense [5] provides a system for arbitrary sensing tasks. However, all of these approaches focus on general hardware and software systems challenges and neither discusses possible

optimizations nor challenges due to node mobility.

Several simple prototype systems aimed at monitoring environmental variables have been developed [19], [25]. They require inefficient constant sampling from each participating mobile node.

Previous approaches for optimizing public sensing focused on nodes that read fixed sensors using RFID [29]. Mobile nodes coordinate to sample each sensor at most once in every sampling period. Further optimizations are presented by Weinschrott et al. for sensors taking continuous measurements along street segments [30]. Nodes coordinate to achieve k-coverage, however, no additional optimization to completely eliminate readings is done. In the MapCorrect approach [2] for creating and validating road maps from GPS traces this coordination is extended by mobility prediction to improve the duty cycle of individual devices.

Lu et al. present another [18] approach for location-centric sensing task execution. However, their system only focuses on individual points of interest rather than observing a larger area.

To determine mobile nodes most suitable for executing a sensing task, Reddy et al. developed a reputation system [23]. The reputation is built over a long time from geographical and temporal availability and willingness to participate in task execution. The output of the system is however designed to aid a human operator in node selection rather than providing automatic selection.

Several works exist in optimizing the placement and scheduling of sensors in traditional sensor networks. In coverage-based approaches, sensor nodes are placed to minimize the overlap of their respective coverage areas to yield full coverage of an observed area with a minimum number of sensors [1], [31]. More closely related to our work, model-driven approaches determine the information value of individual sensors and limit data acquisition to those sensors [9]–[11], [16]. However, none of these systems have to deal with either online learning of models or unavailability of sensors.

Using mobile sensors, works in actuated sensing also focus on model-driven optimizations for data acquisition [4], [27]. In these approaches, optimal paths for mobile sensors are computed to take the most interesting readings in a given area within a given time limit. The underlying assumption is that the mobility of nodes can be controlled by the system, which does not hold in public sensing.

VIII. Conclusion

In this work, we presented the DrOPS system for monitoring environmental values using public sensing. DrOPS uses a model-driven sensing approach based on multivariate Gaussian distributions to infer readings, in order to reduce the set of mobile nodes that are queried for effective readings to reduce the energy consumption. Moreover, we can compensate for missing readings due to unavailable virtual sensors. Furthermore, we introduced an online learning algorithm to learn multivariate Gaussian distributions over short time periods, and MOCHA, an online model validity check algorithm to determine whether a given multivariate Gaussian distribution fits current sensor readings.

Our evaluations show that we obtain optimization models in a matter of just hours or minutes. Using the model-driven approach for optimizing the data acquisition, we can save up to 38 % of energy for communication and provide inferred readings for uncovered positions matching an error-bound of $1\,^{\circ}C$ up to 99 % of the time.

In current work, we are building a prototype implementation of the DrOPS system to evaluate the system performance in a real-world setup. Furthermore, we plan to introduce an energy efficient hybrid 3G/WiFi routing algorithm to further reduce the energy cost for communication.

IX. Acknowledgments

This work was partially funded by the SpoVNet project of Baden-Württemberg Stiftung gGmbH.

References

[1] Z. Abrams, A. Goel, and S. Plotkin. Set K-Cover Algorithms for Energy Efficient Monitoring in Wireless Sensor Networks. In *IPSN '04: Proc. of the 3rd intl. Symp. on Inform. Process. in Sensor Networks*, pages 424–432, New York, NY, USA, 2004. ACM.

[2] P. Baier, H. Weinschrott, F. Dürr, and K. Rothermel. MapCorrect: automatic correction and validation of road maps using public sensing. In *36th Annu. IEEE Conf. on Local Comput. Networks (LCN 2011)*, Bonn, Germany, Oct. 2011.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of the 9th ACM SIGCOMM Conf. on Internet Measurement*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[4] D. Budzik, A. Singh, M. Batalin, and W. Kaiser. Multiscale Sensing with Stochastic Modeling. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Syst., IROS 2009.*, pages 4637–4643, 2009.

[5] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson. People-Centric Urban Sensing. In *WICON'06: Proc. of the 2nd Annu. Intl. Work. on Wireless Internet*, New York, NY, USA, 2006. ACM.

[6] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, 12(4):12–21, 2008.

[7] N. A. C. Cressie. *Statistics for Spatial Data*. Wiley-Interscience, Oct. 1993.

[8] D. Cuff, M. Hansen, and J. Kang. Urban Sensing: Out of the Woods. *Commun. ACM*, 51(3):24–33, Mar. 2008.

[9] A. Das and D. Kempe. Sensor Selection for Minimizing Worst-Case Prediction Error. In *IPSN '08: Proc. of the 7th Int. Conf. on Inform. Process. in Sensor Networks*, pages 97–108, Washington, DC, USA, Apr. 2008. IEEE Computer Society.

[10] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB '04: Proc. of the 30th Int. Conf. on Very large data bases*, pages 588–599. VLDB Endowment, 2004.

[11] H. González-Banos. A Randomized Art-Gallery Algorithm for Sensor Placement. In *SCG '01: Proc. of the 17th Annu. Symp. on Computational Geometry*, pages 232–240, New York, NY, USA, June 2001. ACM.

[12] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *Proc. of the 22nd Int. Conf. on Machine learning*, ICML '05, pages 265–272, New York, NY, USA, 2005. ACM.

[13] M. M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, Oct. 2008.

[14] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. Cartel: a distributed mobile sensor computing system. In *SenSys'06: Proc. of the 4th Int. Conf. on Embedded Networked Sensor Syst.*, pages 125–138, New York, NY, USA, 2006. ACM.

[15] E. Kanjo, S. Benford, M. Paxton, A. Chamberlain, D. S. Fraser, D. Woodgate, D. Crellin, and A. Woolard. MobGeoSen: Facilitating Personal Geosensor Data Collection and Visualization Using Mobile Phones. *Personal and Ubiquitous Computing*, 12:599–607, 2008.

[16] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward Community Sensing. In *IPSN '08: Proc. of the 7th Int. Conf. on Inform. Process. in Sensor Networks*, pages 481–492, Washington, DC, USA, 2008. IEEE Computer Society.

[17] D. G. Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *J. of the Chem., Metal. and Mining Soc. of South Africa*, 52(6):119–139, 1951.

[18] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell. Bubble-sensing: Binding Sensing Tasks to the Physical World. *Pervasive and Mobile Computing*, 6(1):58 – 71, 2009.

[19] N. Maisonneuve, M. Stevens, M. E. Niessen, P. Hanappe, and L. Steels. Citizen Noise Pollution Monitoring. In *Proc. of the 10th Annu. Int. Conf. on Digital Government Research*, dg.o '09, pages 96–103. Digital Government Society of North America, 2009.

[20] E. Miluzzo, N. D. Lane, S. B. Eisenman, and A. T. Campbell. CenceMe – Injecting Sensing Presence into Social Networking Applications. In G. Kortuem, J. Finney, R. Lea, and V. Sundramoorthy, editors, *Smart Sensing and Context*, volume 4793 of *Lecture Notes in Computer Science*, pages 1–28. Springer Berlin / Heidelberg, 2007.

[21] D. Nadeau, W. Brutsaert, M. Parlange, E. Bou-Zeid, G. Barrenetxea, O. Couach, M.-O. Boldi, J. Selker, and M. Vetterli. Estimation of urban sensible heat flux using a dense wireless network of observations. *Environmental Fluid Mechanics*, 9:635–653, 2009.

[22] D. Philipp, F. Dürr, and K. Rothermel. A sensor network abstraction for flexible public sensing systems. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th Int. Conf. on*, pages 460–469, Oct. 2011.

[23] S. Reddy, D. Estrin, and M. B. Srivastava. Recruitment Framework for Participatory Sensing Data Collections. In *Int. Conf. on Pervasive Computing (Pervasive)*, page 18, May 2010.

[24] A. Schwaighofer, V. Tresp, and K. Yu. Learning gaussian process kernels via hierarchical bayes. In *Adv. in Neural Information Processing Systems (NIPS)*, number 17, pages 1209–1216. MIT Press, 2005.

[25] A. Steed and R. Milton. Using Tracked Mobile Sensors to Make Maps of Environmental Effects. *Personal and Ubiquitous Computing*, 12(4):331–342, 2008.

[26] I. Stepanov. CANU Mobility Simulation Environment (CanuMobiSim). Online.

[27] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm. In *IJCAI'09: Proc. of the 21st Int. joint Cinf. on Artifical Intell.*, pages 299–304, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[28] A. Varga. The OMNeT++ Simulator. Online.

[29] H. Weinschrott, F. Dürr, and K. Rothermel. Efficient Capturing of Environmental Data with Mobile RFID Readers. In *MDM '09: Proc. of the 10th Int. Conf. on Mobile Data Manage.: Syst., Services and Middleware*, pages 41–51, Washington, DC, USA, 2009. IEEE Computer Society.

[30] H. Weinschrott, F. Dürr, and K. Rothermel. StreamShaper: Coordination Algorithms for Participatory Mobile Urban Sensing. In *Proc. of the 7th IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS'10)*, pages 1–10, San Francisco, CA, USA, Nov. 2010. IEEE.

[31] H. Zhang and J. Hou. Maintaining Sensing Coverage and Connectivity in Large Sensor Networks. *Ad Hoc & Sensor Wireless Networks*, 1(1-2), 2005.