SensoClean: Handling Noisy and Incomplete Data in Sensor Networks using Modeling

Yee Lin Tan **Vivek Sehgal** **Hamid Haidarian Shahri**
University of Maryland
{yeelin, viveks, hamid}@cs.umd.edu

Abstract: Sensor networks have shown tremendous growth in many domains such as environmental monitoring. The data captured from the physical world through these sensor devices, however, tend to be incomplete, noisy, and unreliable. Traditional data cleaning techniques cannot be applied to such data as they do not take into account the strong spatial and temporal correlations typically present in sensor data. Popular data modeling methods like Kalman filters and regression have shown good results in capturing spatio-temporal correlations. We implemented these methods in an extensible toolkit with graphical visualization, and explored their effectiveness in cleaning sensor data. We obtained good data cleaning results in our experiments using Kalman filters. Regression with a high-order polynomial also showed promising results, but worked poorly for data with high variability.

1. Introduction

Sensor networks are deployed in various domains to acquire information about different physical phenomena in real-time. The data acquired is typically not usable directly as it suffers from three problems, namely noise, missing data and incompleteness. There are several factors contributing to these problems. Noise usually occurs because of inaccuracy in hardware sensing and transmission. Unfavorable environmental conditions and limited battery power further exacerbates this problem. Missing values usually occur due to packet loss and node failure. Since sensors sample continuous physical phenomena at discrete time intervals, the data acquired is incomplete. Hence, incompleteness is an inherent problem with data acquisition in sensor networks, while missing values are accidental in nature. All these problems seriously impact the quality of data obtained from such networks.

Although advancements have been made in the manufacturing of sensors such as miniaturization, power efficiency, computing power, memory, etc., the problems of precision and accuracy still persist [5]. The aim of the industry is to manufacture tiny, cheap sensors that can be deployed everywhere and disposed when depleted. Consequently, noise, imprecision and inaccuracies are inevitable in these cheap sensors. It is extremely important that data from these sensors be reliable since actions are usually taken based on their readings. Dirty data can lead to detrimental effects since it may be used in critical decisions or in the activation of actuators.

Statistical and probabilistic modeling techniques have been used [4, 15] to solve the issues we discussed earlier. Modeling usually involves two phases: training and testing. In training, the parameters of the characteristic function representing the data are learned. Sometimes held-out data is used for validation to further improve the accuracy of the training process by preventing over-fitting. In the next phase, predictions are made about the testing data. Training is frequently done offline while testing can be done either offline or online.

The accuracy of statistical modeling is problem and data specific. Sensor data is temporal and spatial in nature. In general, a reading is usually of the format <*sensor-id, location, time, value*>. If the sensors are static, then the location field is usually omitted. Individual observations are assumed to be independent.

In our experiments we will examine the efficacy of the following methods:

1. **Kalman filter** [12, 16]: The Kalman filter is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements. An example of an application would be to provide accurate continuously-updated information about the position and velocity of an object given only a sequence of observations about its position, each of which includes some error. It is used in a wide range of engineering applications from radar and control systems to computer vision.
2. **Regression** [2, 13]: This usually involves fitting the best curve for a given set of points. In our case, since the data is time-varying and spatial, we use regression to find the best curve approximating the readings. This curve can be used not only to find missing or unknown data but also to reduce noise.

The focus of prior work in data cleaning has been primarily in the context of information integration and data-warehousing [1, 6, 9, 14]. However, the nature of sensor network data is inherently different, and previous approaches cannot always be applied directly in this domain. For instance, issues such as approximate duplicate elimination and schema matching are of great importance and well studied in that domain but are not relevant in sensor data. Little work has been done on data cleaning in the context of sensor network data. In this paper, we present a comparative study of two methods to improve the quality and reliability of data from sensor networks. We hope to obtain insights into which methods work better for certain datasets. We will build a toolkit implementing the statistical methods described above, which will allow researchers to apply and compare different data cleaning techniques on their sensor data.

Cleaning and query processing can be performed either at the individual sensors or at the base station. If cleaning is performed at the sensors, there would be significant communication cost in sending the parameters of the model to the individual sensors. Furthermore, there is a storage cost associated with storing the parameters. In addition to communication and storage costs, performing the actual cleaning at the sensors would incur a processing cost on the resource-constrained sensors. These problems do not arise if cleaning is done at the base station, given the typical processing power and storage capacity of base stations. Moreover, there would be huge savings by not having to communicate the model parameters to each individual sensor. Given that the lifetime of the sensors is heavily dependent on the amount of communication that they do, communication savings is very important. Having the data and the model at the base station is advantageous when it comes to query processing, as answers to user queries can be easily computed.

Performing the cleaning at the sensor level and query processing at the base station has no clear advantages. This is because communicating a single noisy reading to the base station and performing the cleaning work there incurs less communication cost than communicating all the model parameters over to the sensor itself. The latter, as we have mentioned earlier, imposes unnecessary processing and storage overhead on the sensors.

We begin by providing some background on the modeling techniques we use in the toolkit in Section 2. We then present a high-level design of our toolkit and the query processing module in Section 3. In Section 4, we describe specific details of our initial toolkit implementation, followed by an experimental study of the Intel Lab dataset using the two modeling techniques provided by our toolkit in Section 5. In Section 6, we discuss previous work related to data cleaning, and we conclude in Section 7.

2. Background

2.1. Kalman filter

The Kalman filter is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements. It is based on linear algebra and the hidden Markov model [12, 16]. The underlying dynamical system is modeled as a Markov chain built on linear operators perturbed by Gaussian noise. The state of the system is represented as a vector of real numbers. At each discrete time increment, a linear operator is applied to the current state to generate a new state, with some noise mixed in, and optionally, some information from the controls on the system if they are known. Then, another linear operator mixed with more noise generates the visible outputs from the hidden state.

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, the process has to be modeled in accordance with the framework of the Kalman filter as shown in the Figure 1. This means specifying the matrices F_k, H_k, Q_k, R_k, and B_k for each time-step k as described below.

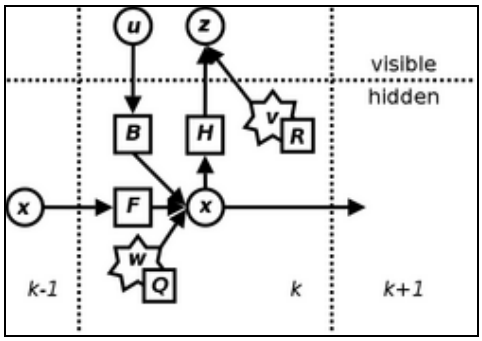

Figure 1: Model underlying the Kalman filter. Circles are vectors, squares are matrices, and stars represent Gaussian noise with the associated covariance matrix at the lower right.

The Kalman filter model assumes the true state at time k, x_k is evolved from the state at *(k-1)* according to

$$x_k = F_k x_{k-1} + B_k u_k + w_k$$

where

- F_k is the state transition model which is applied to the previous state x_{k-1};
- B_k is the control-input model which is applied to the control vector u_k;
- w_k is the process noise which is assumed to be drawn from a zero mean multivariate normal distribution with covariance Q_k

$$w_k \sim N(0, Q_k)$$

At time k an observation (or measurement) z_k of the true state x_k is made according to

$$z_k = H_k x_k + v_k$$

where H_k is the observation model which maps the true state space into the observed space and v_k is the observation noise which is assumed to be zero mean Gaussian white noise with covariance R_k.

$$v_k \sim N(0, R_k)$$

The initial state x_0 and the noise vectors at each step $\{w_1, \ldots, w_k, v_1, \ldots, v_k\}$ are all assumed to be mutually independent.

Since the Kalman filter is a recursive estimator, only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. The filter has two distinct phases: *predict* and *update* [12, 16]. The predict phase uses the estimate from the previous time step to produce an estimate of the current state. In the update phase, measurement information from the current time step is used to refine this prediction to arrive at a new, hopefully more accurate estimate.

2.2. Spatio-Temporal Regression

Regression is the process of fitting a curve to best define a set of real values. For the simplest case, a linear regression model can be used to approximate the points generated from a linear random distribution. The equation of the model is given by:

$$Y = WX + N$$

where N is Gaussian noise with zero mean (unbiased noise).

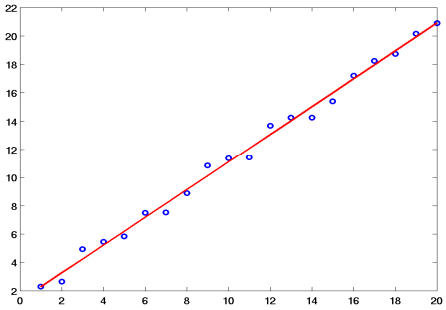

Figure 2: Linear regression

For more complex distributions, the regression model would be a polynomial in X. The best regression equation is usually found by iteratively adding more terms into the equation until there is negligible improvement in the estimate. This process is done offline before the training step using some test data. In the equation above we assumed the noise to follow a Gaussian distribution which might not be necessarily true in general.

In our case, we are dealing with data which has two components, namely time and space. One way of capturing the time dependency in this data is by fitting a time polynomial, such as $f(t) = w_0 + w_1 t + w_2 t^2$. Here 1, t, t^2 are the basis functions, while w_0, w_1 and w_2 are the corresponding weights to be estimated. In general, regression attempts to fit a set of basis functions $\{h_1, h_2, \ldots, h_k\}$ to the measurements using weights $\{w_1, w_2, \ldots, w_k\}$ such that:

$$f(t) = w_1 h_1(t) + w_2 h_2(t) + \ldots + w_k h_k(t)$$

When there are multiple measurements for different times, we need to define a basis matrix H with one column for each basis function and one row for each measurement. Similarly a measurement vector f is defined with one row for each measurement. Hence, the linear system of equations is as follows:

$$f = Hw$$

where

$$f = \begin{pmatrix} f(t_1) \\ f(t_2) \\ \dots \\ f(t_m) \end{pmatrix}$$

$$H = \begin{pmatrix} h_1(t_1) & h_2(t_1) & \dots & h_k(t_1) \\ h_1(t_2) & h_2(t_2) & \dots & h_k(t_2) \\ \dots & \dots & \dots & \dots \\ h_1(t_m) & h_2(t_m) & \dots & h_k(t_m) \end{pmatrix}$$

Note that f is a $m \times 1$ vector while H is a $m \times k$ matrix. Formally, there are m time slots and k basis vectors. We can find the optimal value of w by minimizing the error given by $\|Hw - f\|$. The solution for this optimization is found by setting the gradient of the quadratic objective function to zero:

$$w = (H^T H)^{-1} H^T f$$

The above regression model can be represented as a linear system $Aw = b$, where $A = H^T H$ and $b = H^T f$. This equation can be solved using methods available for linear systems, like Gaussian elimination [7].

The regression model thus far assumes the complete data corpus to be available before training. This is sometimes not true for sensor data, in which data arrives in a streaming fashion. In such a scenario, we may want to fit a regression model using a *sliding window*. Formally, we fit the basis functions with respect to the measurements performed in the last T minutes. Here if we increase the sliding window, the model would have more historical data to be trained upon and thus produce better estimates. But this will also make the process computationally expensive. Usually an optimal value for the sliding window is defined keeping in mind the above trade-off. In this scenario given the matrix A and the vector b for measurements at times $t_1, \dots, t_{m-1}$, these matrices can be updated using the measurement at time t_m [8]. With the updated values, the linear system $Aw = b$ can be solved again to obtain the new weight vector.

Sensors located close to each other typically show correlation in the values observed [8]. Thus, rather than building a regression model for each sensor in isolation, we can model them together to capture the spatial correlation as well. To do this, f and H have to be functions of both time and space (i.e. $f(x,y,t)$ and $H(x,y,t)$). The same regression model presented earlier can be used to solve for the optimal weight vector in this new framework. However, defining a good basis function H in this case is tricky. [8] uses space kernels to define the basis functions. But, in their approach, the kernels were found in an *ad hoc* manner and may not work well in other situations. We will explore basis functions and find the ones which work better in different settings.

3. Design Overview

The toolkit provides a platform for utilizing models to assist in data cleaning. More specifically, the toolkit is designed to clean raw sensor data by exploiting the spatio-temporal aspects of sensor data.

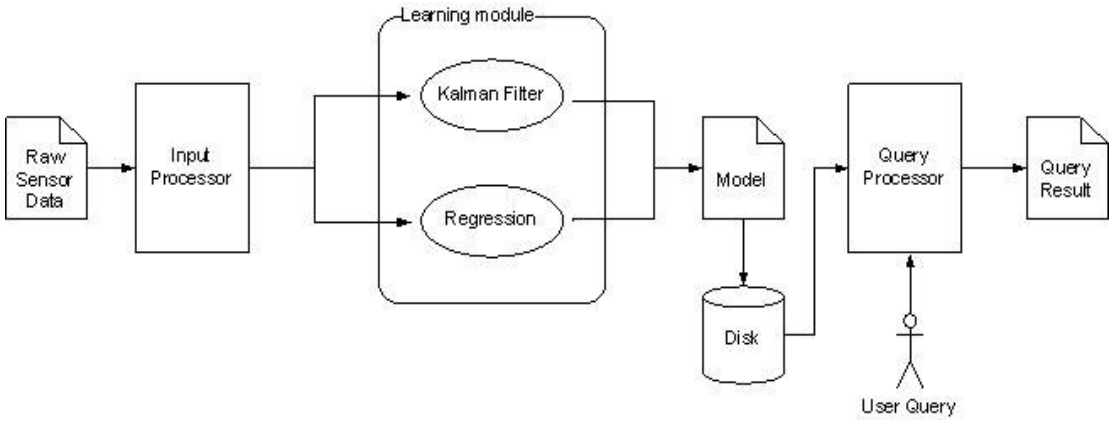

Figure 3: Toolkit architecture

As shown in Figure 3, the toolkit consists of three main components: input processor, learning module, and query processor. Raw sensor data is fed through an input processor which performs some initial preprocessing of the dataset before it is passed to the learning module. The learning module contains learning libraries that are implemented as pluggable components. This allows for new libraries to be added and increases the extensibility of our architecture. We have two learning libraries in our current implementation—Kalman filter and regression. The output of the learning module is a model that is saved to disk. The model is stored on disk so that the model need not be relearned for each user query. Moreover, when queries are presented to the query processor, the model can be easily retrieved and used to answer the user's queries. It also allows for comparisons between different models to be made.

In short, the data extracted by the input processor is modeled using a user-specified library with a given time quantization. Once the model has been trained, the user can pose queries for dataset cleaning, interpolation, and extrapolation to the system.

3.1. Learning Module

As mentioned above, the learning module contains learning libraries that are implemented as pluggable components. New libraries can be easily incorporated into our toolkit architecture. This can be done by modifying the input and output modules for the new library. The new library needs to accept the data structure provided by the input processor. As for the output, the new library needs to produce the true and estimated values in the format required by the query processing module.

3.2. Query Processing Module

Dataset Cleaning: The user can request for a dataset to be cleaned using a model that is trained on the data from the same environment. For example, a model trained on temperature data for a given time period (e.g. Nov-Dec) can be used to clean data for another time period (e.g. Jan). This will avoid repeating the computationally expensive training process. The toolkit prototype currently cleans the dataset that was used for training only.

The user can query the system to read a model from a file and then use it to clean the dataset. The output of such a query will be a dataset in which outliers have been removed and noisy readings are corrected. The output can either be saved into a file or visualized using a two dimensional plot (readings vs. time) for every sensor. For visualization, the user needs to specify the time interval and the sensors for which data is to be visualized. The user can also examine the data at various time steps.

Interpolation: Users can also query for missed readings of a particular sensor for a specified time interval. The user can interpolate the reading for a given sensor using a trained model for a specified time range.

The user can visualize the result on a graph to get a better understanding of how the sensor is behaving over time.

Extrapolation: Similarly, users can ask the model to predict the values of a particular sensor for a future time interval. In principle, this will work in a manner similar to interpolation but with a time interval in the future. The prototype toolkit does not support this functionality currently, but it can be easily extended.

4. Implementation

We have built an initial prototype of the toolkit implementing the architecture described above in Java. The toolkit uses Kalman filter and regression libraries to model sensor data. The Kalman filter library is modified from a learning tool by Welch et al. [17]. Their Kalman filter learning tool was originally designed to model the water level in a tank. We extended their implementation to model sensor data. The Drej [3] regression library was used in the toolkit. Drej is an open-source java library for linear and non-linear least-squares regression and classification. The user interface of the toolkit consists of the following four windows—main window, edit window, plot window, and a step window—which were adapted from the implementation in [17].

4.1. Main Window

The main window (Figure 4) serves to provide the user with choices that control the general execution of the toolkit. In this window, the user can specify the file name of the raw dataset, the total number of columns in the dataset, the number of features of the physical attributes in the dataset, the index of the column containing the feature to be modeled, and the threshold value for that feature. Setting a threshold for the feature being modeled helps in the removal of outliers in the preprocessing phase. In addition, the user is required to specify the desired time quantization for analyzing the data. The current implementation supports time quantization in hours and minutes. The user can choose to input model parameters, plot, and step through the training process by making the corresponding selections in the main window. The "Edit" button invokes the edit window, the "Plot" invokes the plot window, and "Step" invokes the stepper utility.

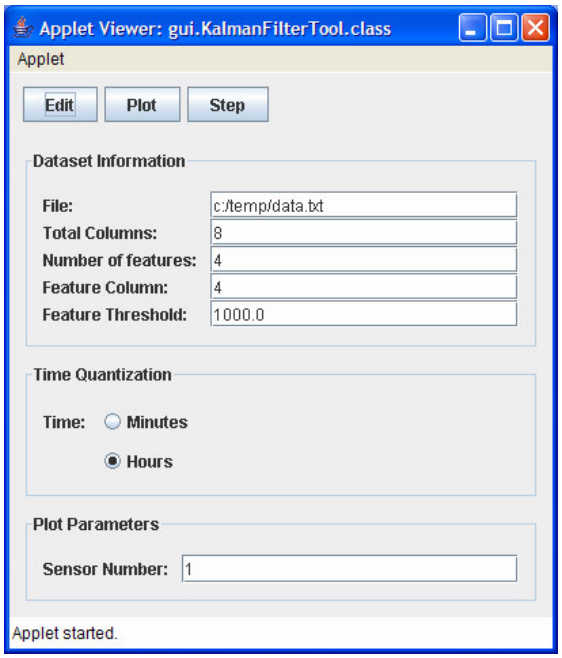

Figure 4: Main window

4.2. Edit Window

The Edit Window (Figure 5) allows the user to select the desired model, and to input the corresponding training parameters. For Kalman Filter, the training parameters are the initial values for the state estimate, x0, and the error covariance, P0. The values for x0 and P0 can be set to any arbitrary value if the user does not have any prior knowledge about the dataset. However, if the user knows that the physical attributed that is being modeled has value around 20, then setting x0 to 20 would help speed up the training process. The default values for x0 and P0 are 0.001 and 0.1, respectively.

For Regression, the training parameters are the kernel function, and the values for lambda and gamma. The kernel functions that are currently supported are linear, quadratic, cubic, multi-quadric, inverse multi-quadric and Gaussian. Lambda is basically a parameter that helps to control the fit, while gamma is used to initialize the multi-quadric, inverse multi-quadric and Gaussian kernels. The range for lambda is between 0.001 and 1.0. The default values for lambda and gamma are 0.5 and 1.0 respectively. The kernels supported by the toolkit are defined as follows:

a. **Linear Kernel:** $K(x1, x2) = x1 \cdot x2$

b. **Quadratic (d=2) and Cubic (d=3):** $K(x1,x2) = (x1 \cdot x2 + 1)^d$

c. **Multi-quadric:** $K(x_1, x_2) = -\sqrt{\left(\left\|x_1 - x_2\right\|^2 + \gamma^2\right)}$

d. **Inverse multi-quadric:** $K(x_1, x_2) = 1 \Big/ \sqrt{\left(\left\|x_1 - x_2\right\|^2 + \gamma^2\right)}$

e. **Gaussian:** $K(x_1, x_2) = \exp\left(-\gamma^2 \left\|x_1 - x_2\right\|^2\right)$

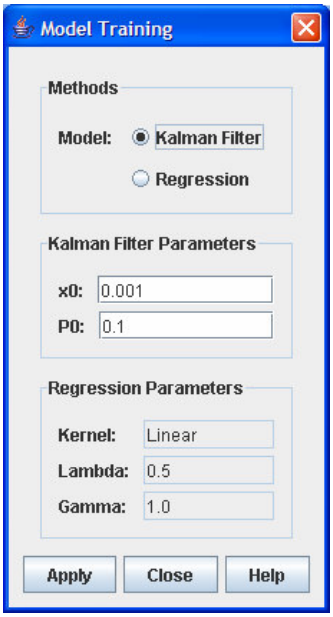

Figure 5: Edit window

4.3. Plot Window

Once the model is trained, the user can visualize the results of the training process given a specific sensor id. The plot window (Figure 6) displays three plots to the user. The first plot shows the actual and estimated values for each time step, while the second plot tracks the change in the error covariance over time. The third plot shows the residual, that is the difference between the actual and the estimated value, over the training duration. This plot is useful when comparing the efficacy of the different models.

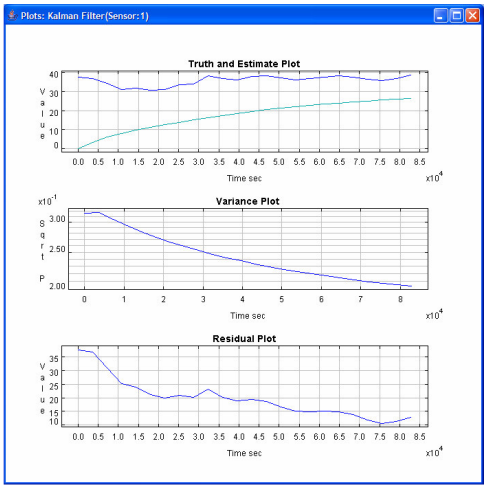

Figure 6: Plot Window

4.4. Step Window

The step window (Figure 7), which was adapted from the Kalman Filter Learning Tool by Welch et al [17], allows the user to step through the actual Kalman filter training process for each time step. The step utility is designed to allow users to study exactly how the Kalman Filter training works. The step function shows, for a selected time step, the values that determine the true state, the predicted state, the corrected state, the actual measurement, the predicted measurement, the Kalman gain, the predicted covariance, and the corrected covariance. These values are displayed for two adjacent time steps so that the user can see how these quantities change over time.

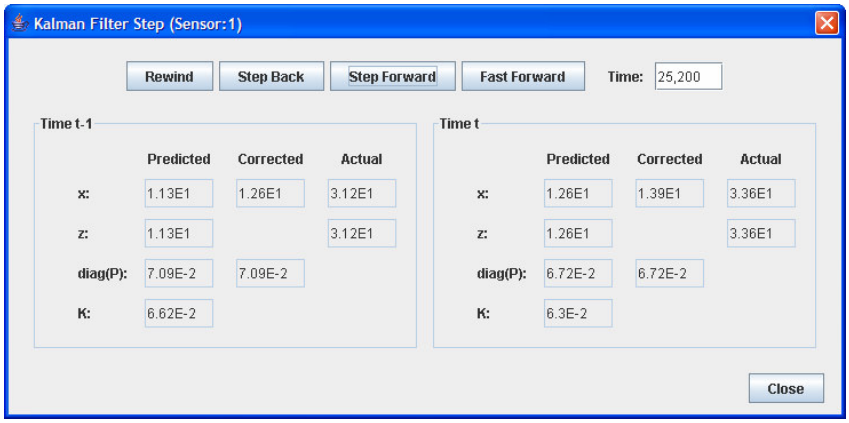

Figure 7: Step window

5. Experimental Results

In this section, we report the results of an empirical study of the Intel lab dataset using our toolkit. We present a comparative study of the Kalman filter and the regression models for each of the physical attributes (temperature, humidity, light and voltage) in the dataset.

Only three regression kernels were used in our experiments: linear, quadratic, and multi-quadric. For each of these experiments, we set the lambda and gamma parameters to 0.5 and 1.0, respectively. These values were selected based on their effectiveness. For Kalman filters, we ran the experiments with time quantized in both hours and minutes, while regression was done with time quantized in hours only. This is due to the high space complexity and memory requirements needed for matrix manipulations in regression.

Intel Dataset: For our study, we used the publicly available Intel Lab dataset [10] which contains data collected from 54 sensors deployed in the Intel Berkeley Research lab. Mica2Dot sensors with weather boards collected time-stamped topology information, along with humidity, temperature, light and voltage values once every 31 seconds. The data was collected using the TinyDB in-network query processing system. The dataset consists of approximately 2.3 million readings collected from these sensors. The format of the dataset is as follows: date, time, epoch, mote ID, temperature, humidity, light, and voltage.

The sensor ids range from 1-54. Figure 8 shows the placement of the sensors in the lab. Data from some sensor motes may be missing or truncated. Temperature is measured in degrees Celsius. Humidity is temperature corrected relative humidity, ranging from 0-100%. Light is in Lux (a value of 1 Lux corresponds to moonlight, 400 Lux to a bright office, and 100,000 Lux to full sunlight.) Voltage is expressed in volts, ranging from 2-3. The batteries that were used were lithium ion cells, which maintain a fairly constant voltage over their lifetime. Note that variations in voltage are highly correlated with temperature.

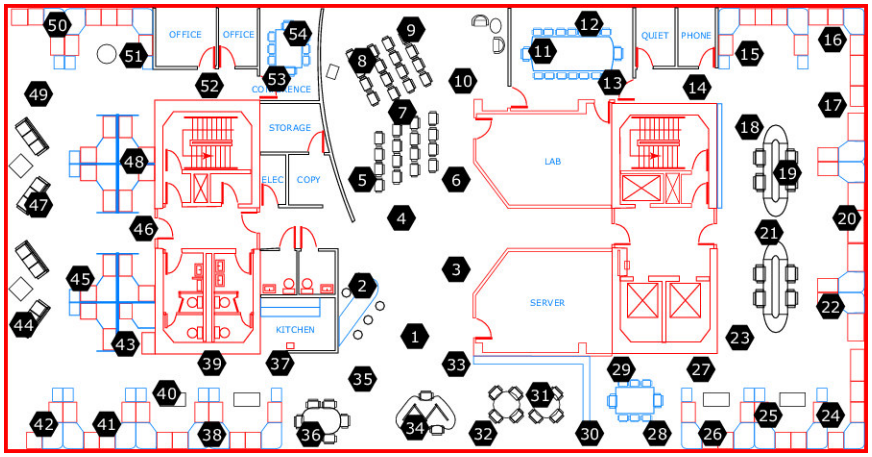

Figure 8: Sensor arrangement in the Intel Berkeley Research Lab

5.1. Temperature

The original temperature data in the lab dataset ranged up to 120 degrees Celsius, while the actual temperature was around 20-40 degrees Celsius. To eliminate the outliers, we set a threshold of 40 degrees Celsius which led to removal of 415,509 readings, which is approximately 18% of the total temperature data. Since the temperature in the lab was typically around 20 degrees Celsius, we set the Kalman filter parameter, x0, to 20 degrees Celsius.

Regression Model: Figure 10 shows that regression using a multi-quadric kernel function performed very well for predicting temperature using time, sensor ID, humidity, light and voltage as features. By using time and sensor ID in the input data, the model was able to capture temporal as well as spatial correlations.

For this kernel function, the average residual for one of the sensors is 0.12 ± 0.01 degrees Celsius, as shown in Table 1.

Using a simpler regression model like the linear kernel, an average residual of 2.08 ± 0.61 degrees Celsius (see table 1) was obtained, which is significantly higher than that obtained by the multi-quadric kernel. By comparing the truth vs. estimate plots for linear and multi-quadric kernels in Figures 11 and 10, we infer that temperature modeling requires a more complex kernel.

Features	Statistics	Kalman Filter (Hours)	Kalman Filter (Minutes)	Regression (Hours)		
				Linear	Quadratic	Multi-quadric
Temperature	**Avg**	2.32	0.59	2.08	0.90	0.12
	Stdev	0.69	0.02	0.61	0.23	0.01
Humidity	**Avg**	2.94	2.32	2.64	0.96	0.34
	Stdev	0.12	0.07	0.86	0.17	0.08
Light	**Avg**	131.75	21.84	154.02	205.94	71.54
	Stdev	43.59	0.62	86.09	13.93	11.24
Voltage	**Avg**	0.23	0.03	0.06	0.07	0.006
	Stdev	0.03	0.001	0.04	0.01	0.001

Table 1: Average residual and standard deviation when modeling temperature, humidity, light and voltage using Kalman filters and regression with various kernels. For Kalman filters, time was quantized in hours and minutes, while regression was done with time quantized in hours only.

Kalman Filter Model: When time quantization was in hours, the Kalman filter was trained for only 24 epochs which is insufficient to complete the training process (Figure 9), and therefore shows a relatively large average residual of 2.32 ± 0.69 degrees Celsius. We note that this average residual is higher than the result from the simple linear regression model. This shows that when the number of training samples is limited, regression outperforms Kalman filters.

When time was quantized in minutes instead, the number of training epochs increased 60 folds and the learning process for the Kalman filter improved significantly, as shown in Figure 12. The average residual was more encouraging at 0.59 ± 0.02 degrees Celsius.

Figure 9: Temperature truth and estimate plot using Kalman filter in hour

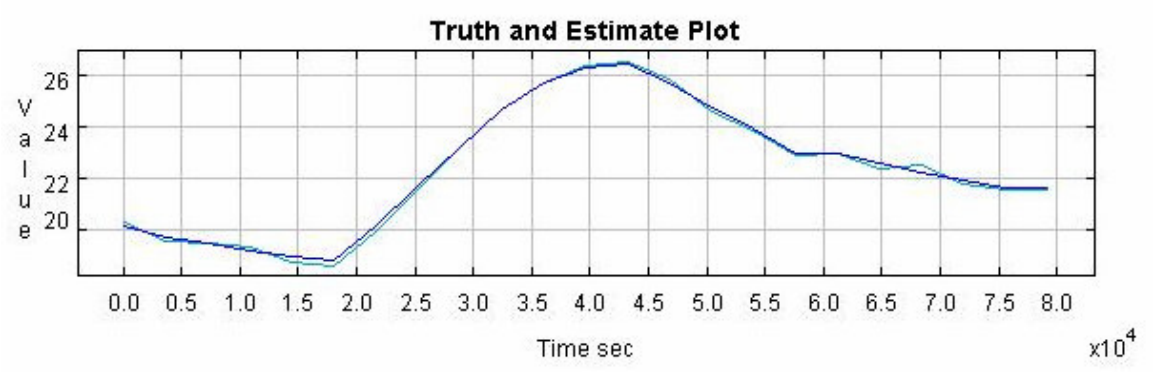

Figure 10: Temperature using regression with multi-quadric kernel

Figure 11: Temperature using regression with linear kernel

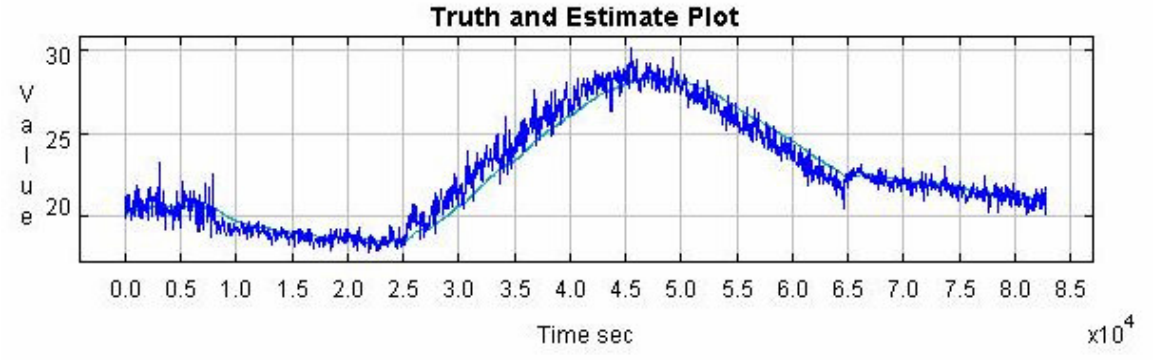

Figure 12: Temperature truth and estimate plot using Kalman filter in minutes

5.2. Humidity

Since humidity readings are known to lie within the 0-100% range, the experiments were run with the value of the feature threshold set to 100. This eliminated 3,901 outliers. Given that the indoor humidity in the lab was around 30%, the initial value of the Kalman filter parameter x0 was set to 30.

Regression Model: Similar to temperature, when predictions were made on humidity, regression using a multi-quadric kernel function was found to perform the best (Figure 13). The average residual was 0.34 ± 0.08 %, as shown in Table 1.

Regression with linear kernel, on the other hand, was not able to capture the humidity variable well (Figure 14). The average residual in this case was 2.64 ± 0.86 % (see Table 1). Like temperature, humidity requires a more complex kernel function for modeling.

Kalman Filter Model: Kalman filter was found to perform well for humidity too. Figure 16 shows Kalman filter trained over sensor data with time quantized in hours. The average residual for this model was 2.94 ± 0.12 %. Figure 15 shows the true and estimated humidity values for the Kalman filter trained with minute quantization. The average residual in this case was 2.32 ± 0.07 %.

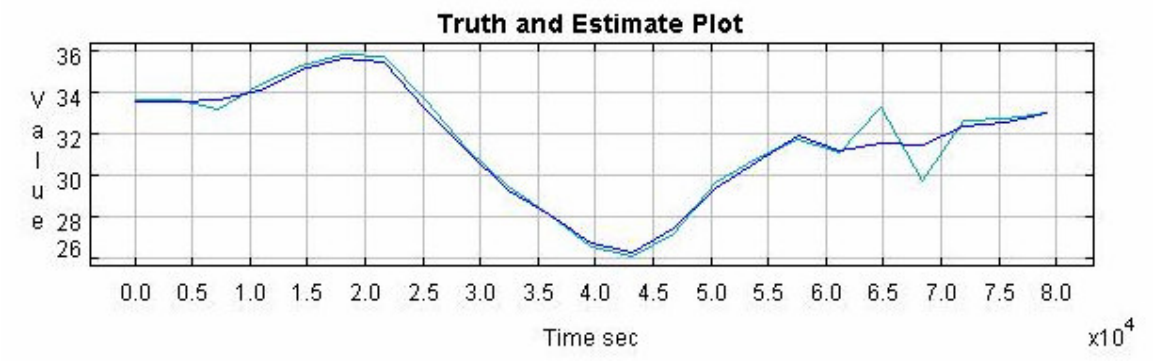

Figure 13: Humidity truth and estimate plot using regression with multi-quadric kernel

Figure 14: Humidity truth and estimate plot using regression with linear kernel

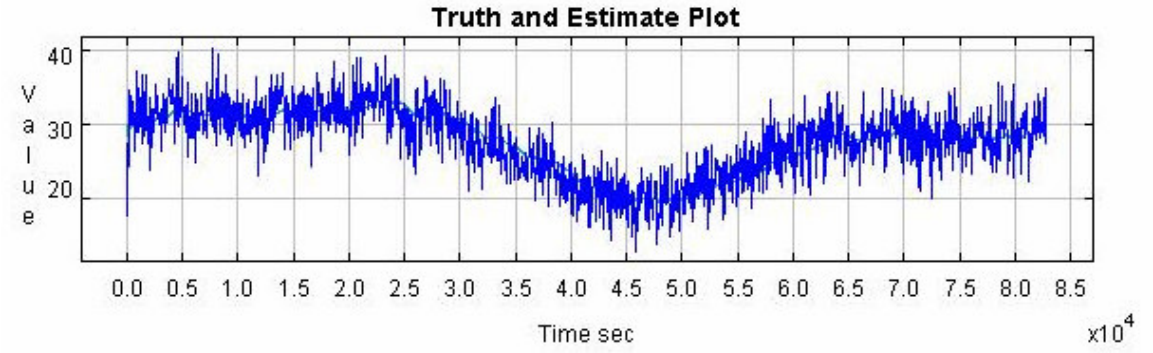

Figure 15: Humidity truth and estimate plot using Kalman filter in minutes

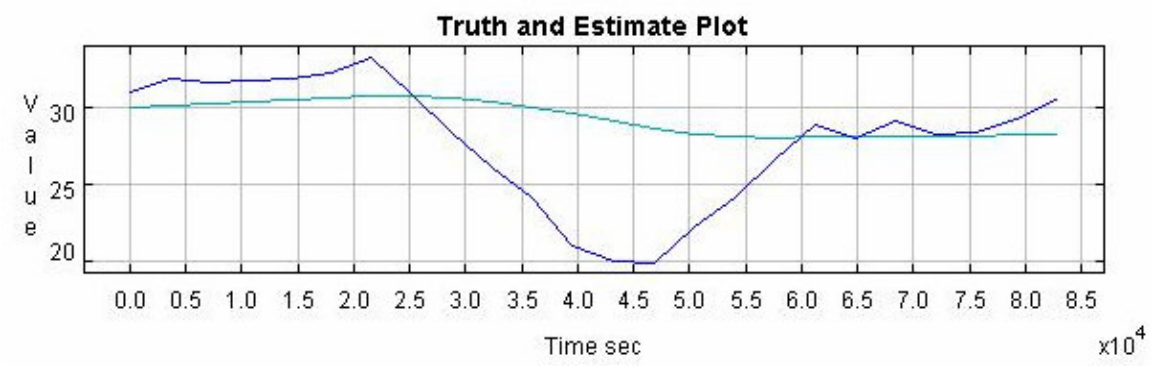

Figure 16: Humidity truth and estimate plot using Kalman filter in hours

5.3. Light

In view of the fact that light readings can range up to 100,000 Lux for sensors exposed to full sunlight, we set the feature threshold for the initial preprocessing phase to 100,000 Lux. This upper bound did not eliminate any light readings from the dataset. Since indoor lighting is subject to human intervention and could range from anywhere between 0-100,000 Lux, the Kalman filter parameter x0 was arbitrarily set to an initial value of 0.001.

Regression Model: Unlike temperature and humidity, regression performs poorly on the light data. Among the kernels, the multi-quadric kernel performed the best with an average residual of 71.54 ± 11.24 Lux (Table 1). See Figures 17 and 18 for the true vs. estimate plots from the multi-quadric and linear fits. With the large amount of variability in light, none of the regression kernels were able to fit the data properly.

Kalman Filter Model: Figure 20 shows that Kalman filter modeled with hour quantization performed better than linear and multi-quadric regression, with an average residual of only 131.75 ± 43.59 Lux. From Figure 19, we see that the Kalman filter model with minute quantization is able to learn the behavior of light very well. The average residual is 21.84 ± 0.62 Lux. In view of the possibility of light readings ranging from 1-100,000 Lux, this average residual value is quite impressive.

Figure 17: Light truth and estimate plot using regression with multi-quadric kernel

Figure 18: Light truth and estimate plot using regression with linear kernel

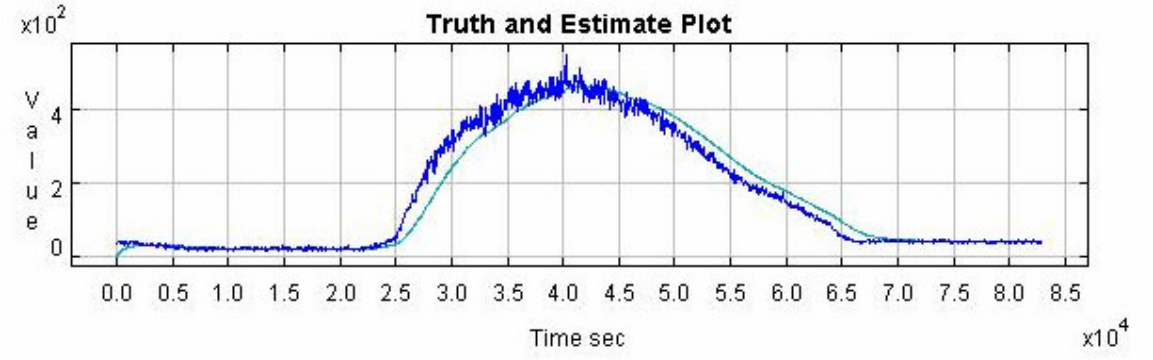

Figure 19: Light truth and estimate plot using Kalman filter in minutes

Figure 20: Light truth and estimate plot using Kalman filter in hours

5.4. Voltage

All voltage readings from the dataset were used in the experiments, as no outliers were discarded in the preprocessing phase. Since voltage is known to have values between 2-3 volts in this dataset, the Kalman filter parameter x0 was set to an initial value of 2 volts.

Regression Model: In contrast to light, all the regression kernels were found to fit the voltage data very closely. This is most likely due to the fact that the voltage values consistently ranged from 2-3 volts. From Table 1, the average residual for the multi-quadric kernel is 0.006 ± 0.001 volts, while the average residual for the linear kernel is 0.06 ± 0.04 volts. Figures 21 and 22 show the corresponding truth vs. estimate plots for both of these kernels.

Kalman Filter Model: The Kalman filter model learned the voltage data very quickly (within the first hour), as shown in the Figures 24 and 23. The average residual over the training process is only 0.23 ± 0.03 volts for hour quantization, and 0.03 ± 0.001 volts for minute quantization.

Figure 21: Voltage truth and estimate plot using regression with multi-quadric kernel

Figure 22: Voltage truth and estimate plot using regression with linear kernel

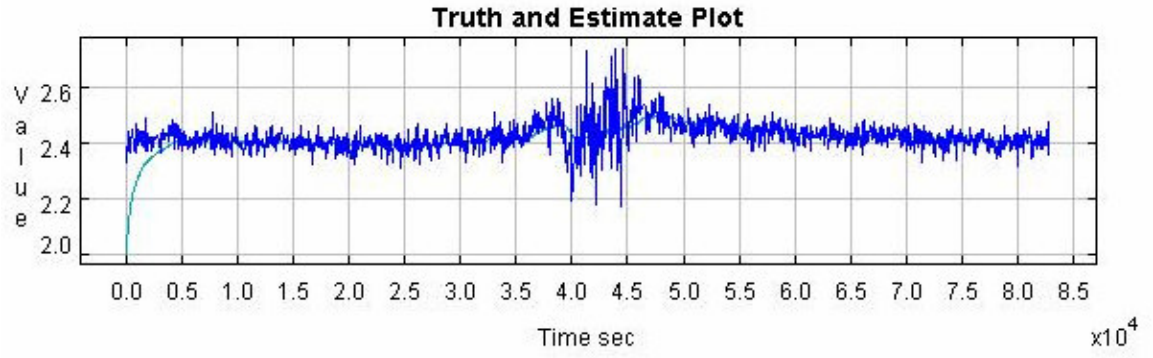

Figure 23: Voltage truth and estimate plot using Kalman filter in minutes

Figure 24: Voltage truth and estimate plot using Kalman filter in hours

In summary, Kalman filter performs better with time quantized in minutes instead of hours. This can be attributed to the fact that in the former case the Kalman filter is trained with more epochs, and hence is able to construct a more accurate model. This is especially true when the variability of the original feature (e.g., light) is high. In our experiments, we found that the difference in average residual when modeling light using minute and hour quantization was very large (Table 1).

In the case of regression, multi-quadric kernel was found to perform better than linear and quadratic kernels for all features. The better performance was most likely due to the higher complexity of the multi-quadric function which allows for greater modeling capacity. Higher complexity, however, can increase the chances of the model to overfit. The quadratic kernel performed better than the linear kernel for temperature and humidity, while the inverse was true for light and voltage.

Although regression with multi-quadric kernel was found to perform very well, even better than Kalman filter in all cases except light, we suspect that it may not generalize well to other datasets due to excessive modeling. It would be interesting to investigate the tradeoff between generalization and over-fitting for this kernel in the future. Kalman filter, on the other hand, uses fewer variables for modeling, and may generalize better to other datasets. When the variability in the data is large as in the case with light, the Kalman filter is still the best data cleaning method.

6. Related Work

Traditional data cleaning systems tend to address problems such as schema transformations, string analysis, and duplicate elimination. Schema transformations usually involve writing ad hoc and laborious scripts for preparing the data format for various analysis and visualization tools. Potter's wheel [14] is an interactive tool for defining schema operations for data cleaning and removing content and structure discrepancies in data. AJAX [6] provides an extensible, declarative language similar to SQL for specifying data cleaning operations in data warehouses. The data quality problems addressed in AJAX are the absence of universal keys across different databases, the existence of keyboard errors in the data, and the presence of inconsistencies in data coming from multiple sources. Both Potter's wheel and AJAX address issues that are intrinsically different from the data quality problems in sensor data. Some commercial data cleaning tools [1, 9] are also available, but cannot be applied to sensor data, as they also do not take into account the spatio-temporal aspects of the data.

Elnahrawy et al. [5] presents a framework for cleaning and querying noisy sensor data. They used a Bayesian approach for reducing uncertainty associated with the data in an online fashion. Their approach assumes an independent Gaussian distribution for each sensor at every time step. Parameters for the distributions are estimated using only data of a particular sensor at the current time step. This assumption leads to a loss of correlation not only along time but also across sensors. Additionally, they assume that all the tuples exist and all the attributes are complete. Hence, their method is unable to handle missing and incomplete sensor data and can perform interpolation only within the time window on which the model was trained. Nevertheless, their model is simple and easily deployable because of the naïve Bayesian assumptions used in their approach.

A recent work that attempts to address cleaning and error correction for sensor data is the ESP (Extensible receptor Stream Processing) platform, which is part of the HiFi project. ESP is a pipelined data processing framework for online cleaning of sensor data streams. It is developed with the intention to clean receptor data streams at the edge of the HiFi network. The ESP platform consists of a pipeline of processing stages designed to operate on data as it is streamed through the system. The stages segment the data cleaning process into separate tasks, each responsible for a different logical aspect of the data. The stages include tuple-level corrections, correction for missed readings, removal of outliers, removal of duplicate readings, and utilization of readings from across different types of receptors to increase the confidence in the data. The authors mentioned that the architecture proposed for ESP provides a platform for utilizing statistical models to assist in data cleaning. Nevertheless, the stages in ESP are currently programmed independently as declarative queries, without employing any such models.

7. Conclusions

In this paper, we discussed the high-level design of a toolkit for cleaning noisy and incomplete sensor data. The toolkit supports data cleaning, interpolation, and extrapolation functionalities. In addition, it provides data analysis tools like visualization and a step utility to examine the actual training process. We present an initial prototype of the toolkit that supports two modeling techniques: Kalman filter and regression. Using this prototype, an experimental study was conducted to explore the effectiveness of the two techniques for temperature, humidity, light, and voltage data. Our experiments show promising results for the Kalman filter for all the physical attributes modeled. The regression model using a high-ordered polynomial also modeled most of the attributes well, but performed poorly relative to Kalman filters when there is large variability in the data, as in the case of light.

References

1. Ascential. http://www.ascential.com/
2. Breiman, L., Friedman, J., Olshen, R., and Stone, C., *Classification and Regression Trees*. Wadsworth International Group, 1984.
3. Dennis, G., Drej - a Java regression library. http://www.gregdennis.com/drej.html
4. Duda, R., Hart, P., Stock, D., *Pattern Classification*. 2nd Ed. John Wiley and Sons, Inc. 2001.
5. Elnahwary, E., Nath, B., Cleaning and Querying Noisy Sensors. In *WSNA'03: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*. 2003.
6. Galhardas, H., et al., Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pp. 371-380, 2001.
7. Golub, G., Van Loan, C., *Matrix Computations.* Johns Hopkins, 1989
8. Guestrin, C., et al., Distributed Regression: An efficient framework for modeling sensor network data. In *ISPN'04*, April 26-27, 2004.
9. Informatica. http://www.informatica.com/
10. Intel Lab Data. http://berkeley.intel-research.net/labdata/
11. Jefferey, S., Alanso, G., Franklin, M., Hong, W., Widom, J., A Pipeline Framework for Online Cleaning of Sensor Data Streams. Technical Report No. UCB/CSD-5-1413, Computer Science Division, University of California at Berkeley, Sept. 2005.
12. Kalman filter. Answers.com. http://www.answers.com/topic/kalman-filter. Wikipedia, 2005.
13. Mitchell, T., *Machine Learning*. McGraw Hill, 1997.
14. Raman, V. and Hellerstein, J.M. Potter's Wheel: An Interactive Data Cleaning System. In *The VLDB Journal*, pp. 381-390, 2001.
15. Russell, S., Norvig, P., *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.
16. Welch, G., Bishop, G., *An Introduction to the Kalman Filter*. Technical Report No. TR 95-041, Department of Computer Science, University of North Carolina, March 2003.
17. Welch, G., Riley, C., Bodenheimer, T., Parker, E., Carpenter, J., Kalman Filter Learning Tool. http://www.cs.unc.edu/~welch/kalman/kftool/index.html