applied sciences

MDPI

Article

Building Survivable Software Systems by Automatically Adapting to Sensor Changes

Yuan Shi [1,2], Ang Li [1,*], T. K. Satish Kumar [1] and Craig A. Knoblock [1]

1 Information Sciences Institute, University of Southern California, Los Angeles, CA 90292, USA; yuanshi@usc.edu (Y.S.); tkskwork@gmail.com (T.K.S.K.); knoblock@isi.edu (C.A.K.)
2 Turing Video, San Mateo, CA 94402, USA
* Correspondence: ali355@usc.edu

Abstract: Many software systems run on long-lifespan platforms that operate in diverse and dynamic environments. If these software systems could automatically adapt to hardware changes, it would significantly reduce the maintenance cost and enable rapid upgrade. In this paper, we study the problem of how to automatically adapt to sensor changes, as an important step towards building such long-lived, survivable software systems. We address challenges in sensor adaptation when a set of sensors are replaced by new sensors. Our approach reconstructs sensor values of replaced sensors by preserving distributions of sensor values before and after the sensor change, thereby not warranting a change in higher-layer software. Compared to existing work, our approach has the following advantages: (a) ability to exploit new sensors without requiring an overlapping period of time between the new sensors and the old ones; (b) ability to provide an estimation of adaptation quality; and (c) ability to scale to a large number of sensors. Experiments on weather data and Unmanned Undersea Vehicle (UUV) data demonstrate that our approach can automatically adapt to sensor changes with 5.7% higher accuracy compared to baseline methods.

Keywords: sensor adaptation; sensor change; survivable software; machine learning

check for updates

Citation: Shi, Y.; Li, A.; Kumar, T.K.S.; Knoblock, C.A. Building Survivable Software Systems by Automatically Adapting to Sensor Changes. *Appl. Sci.* **2021**, *11*, 4808. https://doi.org/10.3390/app11114808

Academic Editors: Alberto Doria, Silvio Cocuzza and Benedetto Allotta

Received: 27 February 2021
Accepted: 13 May 2021
Published: 24 May 2021

1. Introduction

An increasing number of applications require long-term autonomy of software systems and their capability to operate in dynamic environments. Maintaining the quality, durability, and performance of these software systems is very challenging and labor-intensive. Failure to effectively and timely adapt to hardware and resource changes can result in technically inferior and potentially vulnerable systems [1]. For example, software systems based on sensor data can suffer from sensor failures or changes caused by environmental conditions and technical errors [2]. Occasionally, such failures can cause severe safety issues, e.g., faulty sensor data caused the crash of a Lion Air Flight 610, killing all 189 people on board (https://www.cnn.com/2018/11/28/asia/lion-air-preliminary-report-intl/index.html, accessed on 17 May 2021). If software systems could automatically detect sensor failures, these types of catastrophes could be avoided. In addition, if software systems could adapt to sensor failures and changes, we could significantly reduce the time and effort required for software maintenance and promote the long-term use of quality software on platforms that continually change.

As an important step towards building such long-lived, survivable software systems, we study the problem of how to automatically adapt to changes in sensors. Sensor change happens when a sensor gets replaced by new sensor(s). Our goal is to build machine-learning-based adapters that can largely reduce the effect of sensor changes on higher-layer software. The solutions to this problem can have broad impact since there is an increasing volume of sensors that are deployed in real-world systems [3,4]. Without proper adaptation to sensor changes, the higher-layer software may function poorly.

Sensor changes often occur in real-world systems due to replacement of failed sensors, sensor upgrade, energy optimization, etc. [1,5,6]. We use the UUV domain as a real-world scenario: A surge sensor on a UUV may stop working and may be eventually replaced by a new surge sensor. Although the new surge sensor measures the same type of signal, it may introduce a bias different from that of the old one. Throughout this paper, we refer to sensors that are replaced by new sensors as replaced sensors. When sensor change occurs, sensor values from new sensors may not match those from replaced sensors. For example, mis-calibration could exist between replaced sensors and new sensors even when they measure the same types of signals. Furthermore, new sensors may measure additional types of signals that do not exist before the sensor change. Existing literature on sensor changes mainly focuses on change detection but rarely addresses how to adapt to these changes [7–10]. Typically, human experts are required to examine and respond to detected changes. In order to automatically adapt to sensor changes, we would like to reconstruct sensor values of replaced sensors using the remaining sensors and the new sensors. The underlying assumption is that sensor values from a subset of sensors are correlated, which is often the case in real-world systems [4,11]. For example, temperature, humidity and dew point measured by weather sensors are correlated [12], and any two of them can be used to accurately predict the third one. In general, if our adaptation algorithm can accurately reconstruct the original sensor values, then the reconstructed values can be directly input to higher-layer software without any changes to the software itself. Figure 1 shows an example of weather sensors, continually generating timestamp, latitude, longitude, pressure and temperature values. In the second and third rows, the temperature sensor is replaced by two new sensors. To recover the original sensor values of the temperature sensor, we use a reconstruction function f by leveraging sensor values from the remaining sensors and the new sensors.

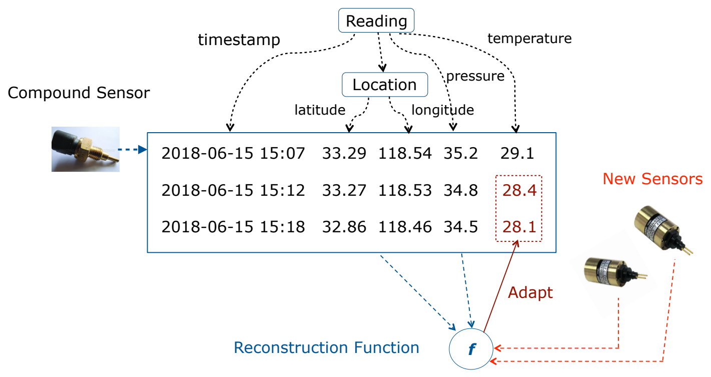


Figure 1. Example of a compound weather sensor that consists of three individual sensors. The reconstruction function f reconstructs failed temperature values from the two remaining working sensors (blue arrows) and the two new sensors (red arrows).

One approach to learning such a reconstruction function is to simply ignore the new sensors and reconstruct the replaced sensors using the remaining ones. Assuming that we have access to sufficient historical sensor values of these sensors, such a reconstruction function can be learned straightforwardly via classical regression methods [13]. In such methods, the sensor values of the remaining sensors are treated as the input variables, and the sensor values of the replaced sensors are treated as the output variables. However, the new sensors may contain complementary information over the remaining sensors, which may help us better reconstruct the replaced sensors. As an extreme example, if the new sensors are exactly the same as the replaced sensors, using their sensor values definitely aids reconstruction.

Learning a reconstruction function that exploits new sensors poses unique challenges since there is no overlapping period of time between the replaced and new sensors. If we intend to apply classical regression methods to learn the reconstruction function, we need training samples that contain the sensor values of both the replaced and the new sensors at the same timestamps. However, since there is no such overlapping period of time, the required training samples are not available and classical regression methods are therefore inapplicable. To address this challenge, we propose an approach called ASC (Adaptation to Sensor Changes) that learns a reconstruction function to preserve sensor value distributions before and after the sensor change. We further improve ASC in two aspects motivated by real-world applications. First, we propose a method to dynamically estimate the adaptation quality, which enables higher-layer software components to determine whether or not to accept an adaptation. Second, we develop a procedure to deal with a large number of sensors by selecting a subset of important sensors. This procedure can significantly reduce the overfitting to noisy values as well as the overall computational cost. It enables our approach to continually exploit new sensors in an open environment. For empirical studies, we evaluate ASC on sensor data from the weather and UUV domains. In most of the evaluation cases, ASC outperforms other baseline approaches, achieving an average improvement of 5.7%.

Our work described in this paper appears in the first author's Ph.D. dissertation [14].

2. Settings and Notations

2.1. Settings

We study the general setting of sensor changes in the context of a compound sensor, i.e., a sensor consisting of multiple individual or component sensors. For example, a compound sensor can be a weather station containing several weather sensors measuring temperature, dew point, wind speed, etc. An instant of time at which some sensor(s) are replaced by new sensors is called a change point (We only address a single change point. Repeated invocation of our methods naturally handles multiple change points as well).

In this paper, we consider two scenarios:

- Individual Sensor Change: some but not all of the individual sensors in a compound sensor are replaced by another set of individual sensors. This corresponds to the cases where new individual sensors are plugged in manually or automatically when sensor failures or sensor upgrades occur.
- Compound Sensor Change: the entire compound sensor is replaced by a new compound sensor. This happens in practice for the reason that replacing the compound sensor is technically easier than replacing individual sensors in certain systems. This scenario is more challenging than individual sensor change, since no individual sensor from the compound sensor can be used to calibrate the new sensors.

Automatic adaptation to sensor changes is challenging since there is no overlapping period between the new sensors and the replaced sensors. In individual sensor change, the remaining sensors are the key to link the information between the new sensors and the replaced sensors. We call the remaining sensors as reference sensors. Intuitively, if the reference sensors are correlated with both the new sensors and the replaced sensors, they can be helpful for reconstructing the replaced sensor values from the new sensor values. For compound sensor change, however, there are no reference sensors from the compound sensor because all sensors are replaced. In this scenario, adaptation to new sensors is very challenging or even impossible. To enable reasonable adaptation, therefore, we assume that we have access to some reference sensors outside the compound sensor. For example, in the context of weather stations, we can use sensors in other stations as reference sensors.

Using the notion of reference sensors, the two scenarios can be viewed in a unified way:

- Reference sensors always work properly.
- Replaced sensors are replaced by new sensors at the change point.

Figure 2 visualizes this unified view and the corresponding notations (explained below).

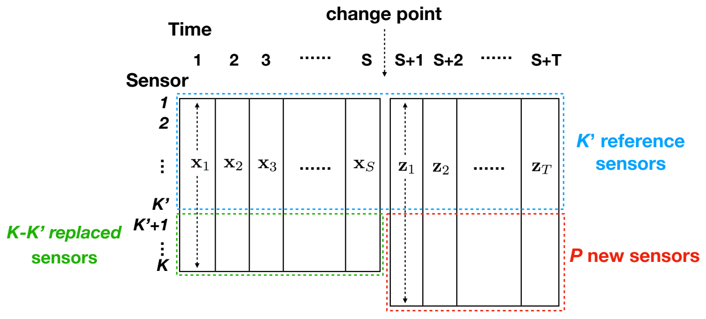

change point
Time
1 2 3 ...... S S+1 S+2 ...... S+T
Sensor
1
2
⋮
K'
K'+1
⋮
K
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ ...... $\mathbf{x}_S$ $\mathbf{z}_1$ $\mathbf{z}_2$ ...... $\mathbf{z}_T$
K' reference sensors
K-K' replaced sensors
P new sensors

Figure 2. Settings and notations for sensor failures and changes.

2.2. Notations

Suppose we are given K individual sensors, among which K' sensors are reference sensors. We assume that

- All sensors generate sensor values at fixed time intervals, and sensor values are temporally aligned.
- Sensor values start at time 1. At time $S+1$, $K-K'$ sensors are replaced by $P \geq 0$ new sensors ($P=0$ corresponds to sensor failure). We have sensor values until time $S+T$.
- There is only a single change point, i.e., time $S+1$, and it is already given.

Let $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_S$ be sensor values before the change point, where $\mathbf{x}_s \in \mathcal{R}^K$ represents sensor values at time $s \in \{1, 2, \cdots, S\}$, and $\mathbf{x}_{s,k}$ represents the corresponding sensor value from sensor $k \in \{1, 2, \cdots, K\}$. Additionally, let the replaced sensors be sensors $K'+1, K'+2, \cdots, K$. Let $\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_T$ denote sensor values after the change point, where $\mathbf{z}_t \in \mathcal{R}^{K'+P}$ represents sensor values at time $S+t$, for $t \in \{1, 2, \cdots, T\}$. Note that we use s to index $\mathbf{x}$ and t to index $\mathbf{z}$. Based on this setting, $\{\mathbf{x}_{s,k}\}$ and $\{\mathbf{z}_{t,k}\}$ for $k \in \{1, 2, \cdots, K'\}$ represent sensor values of reference sensors, and $\{\mathbf{z}_{t,k}\}$, for $k \in \{K'+1, K'+2, \cdots, K'+P\}$ represent sensor values of the P new sensors. Figure 2 illustrates the above notations.

In the following, we often refer to sensor values before the change point as the source domain, and sensor values after the change point as the target domain. Similar notions are used in the domain adaptation and transfer learning communities [15].

3. Approach

One baseline approach is to simply ignore information generated by the new sensors and reconstruct the replaced sensors using only the reference sensors. However, ignoring information generated by the new sensors is suboptimal since it could often complement information generated by the reference sensors. Therefore, we propose an approach that makes use of the new sensors as well as the reference sensors for better reconstruction.

3.1. Assumptions and Intuition

Our approach reconstructs sensor values of the replaced sensors from time $S+1$ to $S+T$ based on the reference sensors and the new sensors. The underlying assumptions are:

- Sensor values from the reference sensors are correlated with those from the replaced sensors.
- Sensor values from the reference sensors are correlated with those from the new sensors.

Such assumptions typically hold in real-world systems because sensor values of different sensors are often correlated [16].

Our approach is based on the following intuition: New sensors may contain complementary information over reference sensors, useful for reconstructing replaced sensors. Figure 3 illustrates this intuition, where the reference sensor, replaced sensor, and the new sensor are temperature, humidity, and dew point, respectively. The left plot shows two selected samples from historical data. We can see that for the same temperature value, humidity can take different values. The middle plot shows that if we attempt to reconstruct humidity from temperature alone, via the g function, then the reconstructed humidity values become exactly the same, since the temperature information alone is insufficient for the reconstruction. The right plot shows that by incorporating dew point as a new signal, the reconstructed humidity values are distributed similarly to those in the left plot. This is expected because dew point contains complementary information over temperature for reconstructing humidity. The above intuition leads to the key idea of our approach: to learn a reconstruction function that preserves the sensor value distributions before and after the sensor change.

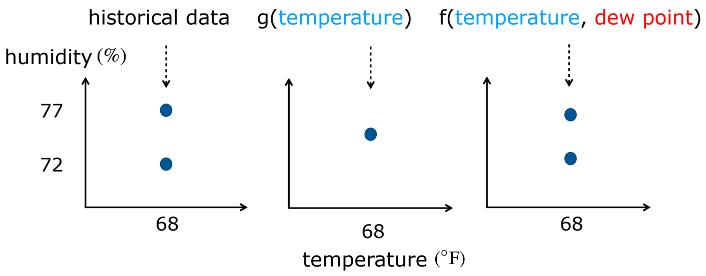


Figure 3. Illustration of the intuition behind ASC.

3.2. Formulation

We follow the notations in Section 2. We refer to sensor values before the sensor change as the source domain and to sensor values after the sensor change as the target domain. Specifically, we aim to learn a reconstruction function $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ that maps sensor values after the sensor change to values before the sensor change, where $\mathbf{\Theta}$ denotes the parameters of the function. Note that the output of $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ is a matrix when there are more than one replaced sensor. In our implementation, we use the form

$$\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}) = \mathbf{\Theta}^{\mathrm{T}}\mathbf{h}(\mathbf{z}) \tag{1}$$

where $\mathbf{h}()$ is a nonlinear feature mapping, e.g., a quadratic function.

We are interested in $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ such that distributions of sensor values are similar across domains after the reconstruction. This motivates us to seek $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ such that the two sets of samples $\{\mathbf{x}_s\}$ and $\{[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)]\}$ (i.e., reconstructed samples in the target domain) are "mixed" as much as possible. We use the notation $1 : K'$ to denote a set of indices from 1 to K'. When this happens, each source-domain sample $\mathbf{x}_s$ becomes close to its k-nearest neighbors in the target domain, and vice versa. Therefore, we propose the following objective function to minimize the cross-domain k-nearest neighbor distances

$$\begin{aligned} \min_{\mathbf{\Theta}} & \sum_{s=1}^{S} \sum_{t \in \mathcal{N}_{\mathcal{T}}^{k}(s)} \mathcal{D}(\mathbf{x}_s, [\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)]) \\ & + \sum_{t=1}^{T} \sum_{s \in \mathcal{N}_{\mathcal{S}}^{k}(t)} \mathcal{D}([\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)], \mathbf{x}_s) + \lambda \|\mathbf{\Theta}\|_2^2 \end{aligned} \tag{2}$$

where $\mathcal{D}(\cdot,\cdot)$ is the distance function defined in the space $\mathbf{x} \in \mathcal{R}^K$. $\mathcal{N}_{\mathcal{T}}^{k}(s)$ denotes the set of indices corresponding to $\mathbf{x}_s$'s k-nearest neighbors in the target domain, and $\mathcal{N}_{\mathcal{S}}^{k}(t)$ denotes the set of indices corresponding to $[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)]$'s k-nearest neighbors in the source domain. Here, nearest neighbors are determined based on the distance function $\mathcal{D}$. $\|\mathbf{\Theta}\|_2^2$ is the regularization term on $\mathbf{\Theta}$ with $\lambda \geq 0$ as the regularization parameter.

For simplicity, we set $\mathcal{D}$ to be the squared Euclidean distance. In our implementation, each dimension is normalized into the same scale. We assume that sensors are generally informative and that there aren't many noisy sensors.

$$\begin{aligned}\mathcal{D}(\mathbf{x}_s,[\mathbf{z}_{t,1:K'};\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]) = \|\mathbf{x}_{s,1:K'}-\mathbf{z}_{t,1:K'}\|_2^2 \\ +\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\end{aligned} \tag{3}$$

Letting $v_{st}^2 = \|\mathbf{x}_{s,1:K'}-\mathbf{z}_{t,1:K'}\|_2^2$, we can write Equation (2) as

$$\begin{aligned}\min_{\boldsymbol{\Theta}} \sum_{s=1}^{S} \sum_{t\in\mathcal{N}_{\mathcal{T}}^k(s)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right) \\ +\sum_{t=1}^{T}\sum_{s\in\mathcal{N}_{\mathcal{S}}^k(t)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right)+\lambda\|\boldsymbol{\Theta}\|_2^2\end{aligned} \tag{4}$$

In Equation (4), $\mathcal{N}_{\mathcal{T}}^k(s)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$ are dependent on $\boldsymbol{\Theta}$, making Equation (4) non-smooth and non-convex in $\boldsymbol{\Theta}$.

3.3. Optimization

For the ease of optimization, we introduce a set of auxiliary variables to decouple the dependency of $\mathcal{N}_{\mathcal{T}}^k(s)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$ on $\boldsymbol{\Theta}$. Let $\mathcal{V}_{\mathcal{T}}^k(s)$ index any of $\mathbf{x}_s$'s (not necessarily the nearest) k neighbors in the target domain, and $\mathcal{V}_{\mathcal{S}}^k(t)$ index any of $[\mathbf{z}_{t,1:K'};\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]$'s k neighbors in the source domain. It is easy to see that

$$\sum_{t\in\mathcal{N}_{\mathcal{T}}^k(s)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right) \tag{5}$$

$$=\min_{\mathcal{V}_{\mathcal{T}}^k(s)} \sum_{t\in\mathcal{V}_{\mathcal{T}}^k(s)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right) \tag{6}$$

and that the same relationship holds for $\mathcal{V}_{\mathcal{S}}^k(t)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$. Thus Equation (4) is equivalent to

$$\begin{aligned}\min_{\boldsymbol{\Theta},\{\mathcal{V}_{\mathcal{T}}^k(s)\},\{\mathcal{V}_{\mathcal{S}}^k(t)\}} \sum_{s=1}^{S} \sum_{t\in\mathcal{V}_{\mathcal{T}}^k(s)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right) \\ +\sum_{t=1}^{T}\sum_{s\in\mathcal{V}_{\mathcal{S}}^k(t)} \left(v_{st}^2+\|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2\right)+\lambda\|\boldsymbol{\Theta}\|_2^2.\end{aligned} \tag{7}$$

Equation (7) can be efficiently optimized via a procedure with two alternating steps. When $\boldsymbol{\Theta}$ is fixed, we update $\{\mathcal{V}_{\mathcal{T}}^k(s)\}$ and $\{\mathcal{V}_{\mathcal{S}}^k(t)\}$ based on nearest neighbor search. When $\{\mathcal{V}_{\mathcal{T}}^k(s)\}$ and $\{\mathcal{V}_{\mathcal{S}}^k(t)\}$ are fixed, we optimize $\boldsymbol{\Theta}$ by solving

$$\min_{\boldsymbol{\Theta}} \sum_{s=1}^{S} \sum_{t\in\mathcal{V}_{\mathcal{T}}^k(s)} \|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \tag{8}$$

$$+\sum_{t=1}^{T}\sum_{s\in\mathcal{V}_{\mathcal{S}}^k(t)} \|\mathbf{x}_{s,K'+1:K}-\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2+\lambda\|\boldsymbol{\Theta}\|_2^2 \tag{9}$$

which can be easier than solving Equation (4) when $\mathbf{f}_{\boldsymbol{\Theta}}$ is smooth in $\boldsymbol{\Theta}$. When $\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)$ is linear in $\boldsymbol{\Theta}$, the optimal $\boldsymbol{\Theta}$ can be computed analytically. In our implementation, we use the linear form $\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}) = \boldsymbol{\Theta}^{\mathrm{T}}\mathbf{h}(\mathbf{z})$, where the nonlinear feature mapping $\mathbf{h}$ maps $\mathbf{z}$ to a nonlinear form with linear ($\{z_j\}$), quadratic ($\{z_j z_k\}$), and exponential ($\{e^{z_j}\}$) terms.

The above procedure decreases the value of the objective function in Equation (7) in each alternating step, and converges to a local minimum of Equation (4). Empirically, the procedure converges quickly (usually within 50 iterations).

3.4. Initialization

The quality of the solution depends on how we initialize $\boldsymbol{\Theta}$. Suppose we have a way to accurately predict the values of the new sensors using $\mathbf{x}_{s,1:K'}$. Let $\mathbf{u}_s$ denote the predicted values of the new sensors. We can initialize $\boldsymbol{\Theta}$ by solving

$$\min_{\boldsymbol{\Theta}} \sum_{s} \|\mathbf{x}_{s,1:K} - \mathbf{f}_{\boldsymbol{\Theta}}([\mathbf{x}_{s,1:K'}; \mathbf{u}_s])\|_2^2. \tag{10}$$

Although estimating $\mathbf{u}_s$ can be very challenging when the correlations between the replaced sensors and the new sensors are weak, we can still estimate a candidate set for $\mathbf{u}_s$ based on target-domain data as follows: For each $\mathbf{x}_{s,1:K'}$, we find a set of its nearest neighbors in $\{\mathbf{z}_{t,1:K'}\}$ and use the corresponding $\mathbf{z}_{t,K'+1:K'+P}$ to form a candidate set $\mathcal{U}_s$. We then minimize the model error by optimizing both $\boldsymbol{\Theta}$ and $\{\hat{\mathbf{u}}_s\}$:

$$\min_{\boldsymbol{\Theta},\{\hat{\mathbf{u}}_s\}} \sum_{s} \min_{\hat{\mathbf{u}}_s \in \mathcal{U}_s} \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}([\mathbf{x}_{s,1:K'}, \hat{\mathbf{u}}_s])\|_2^2 \tag{11}$$

where $\hat{\mathbf{u}}_s$ is allowed to be any element of $\mathcal{U}_s$. Equation (11) essentially relaxes the dependency between the replaced sensors and the new sensors, and uses the optimal $\boldsymbol{\Theta}$ for the relaxed setting as an initialization. By setting $\mathcal{U}_s$ to different sizes, we can get different initial solutions for $\boldsymbol{\Theta}$.

3.5. Parameter Tuning

For tuning the regularization parameter λ, we use a special leave-one-out cross-validation strategy. We synthesize a set of sensor change scenarios by treating each sensor in the source domain as the replaced sensor and using a biased version of that sensor as the new sensor. The biased version is created by offsetting each sensor value by the same bias. We then select the optimal λ such that the average reconstruction error on these synthesized scenarios is minimized.

4. Empirical Study

We evaluated ASC on sensor data from the weather and UUV domains.

- Weather Data: the dataset consisted of weather sensor data collected from Weather Underground (https://www.wunderground.com/, accessed on 17 May 2021). The dataset involved a number of personal weather stations; and each station (compound sensor) contains a set of individual sensors including temperature, dew point, humidity, wind speed, wind gust, pressure, etc. These weather stations were selected from a set of clusters/regions (e.g., Los Angeles, San Francisco, Austin, Chicago, etc.), each with three stations. The clusters were determined based on the cities in which the stations are located. Stations within a cluster tended to produce more similar sensor values than those across clusters. Sensor values were sampled every 5 or 10 min. We temporally aligned sensor values as a preprocessing step. The data used in our empirical study are available for download (https://github.com/usc-isi-i2/sensor-adaptation/tree/master/datasets/weather/dataset, accessed on 17 May 2021). The data were organized into different geographical clusters where each cluster contains a few weather stations. Sensor values were represented in JSON format where each attribute corresponded to a different sensor type.
- UUV Data: the dataset was collected by letting a UUV travel from a starting point to an end point in a simulated environment. The UUV contained a propeller RPM sensor, a waterspeed sensor, and a compound sensor called Doppler Velocity Log

(DVL) sensor. The DVL sensor consisted of seven individual sensors including surge, heave, sway, pitch, roll, depth, and heading. Figure 4 shows the locations of these sensors on a UUV. Each sensor produced a sensor value every second. We simulated 20 trips and collected sensor values at each second. The trajectory of the UUV varies in each trip due to different starting/end points and water currents. The total number of samples in each trip varied between 500 and 2000.

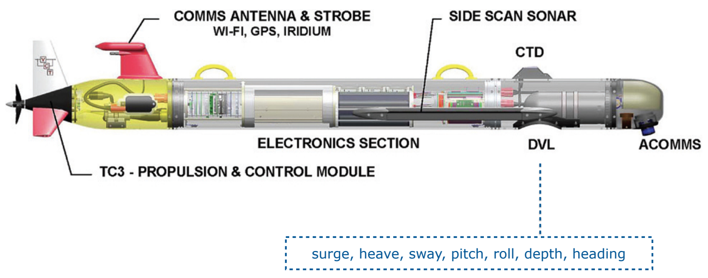


Figure 4. UUV Sensors (RPM, Waterspeed, DVL).

We compared ASC to three baseline methods:

- Replace: non-adaptation method that substitutes each replaced sensor with a new sensor that has the closest mean and variance in sensor values.
- Refer: adaptation method that reconstructs sensor values of replaced sensors using reference sensors, without exploiting any new sensor.
- ReferZ: adaptation method that works in the following three steps:
    1. Learn a regression model on the target domain to reconstruct the new sensors from reference sensors.
    2. Use the learned regression model to reconstruct the new sensors on the source domain.
    3. Learn a reconstruction function on the source domain to reconstruct replaced sensors from reference sensors and reconstructed new sensors.

This method could work well if the new sensors and reference sensors were strongly correlated, which may not hold in real-world applications.

The reconstruction error of each method was measured by RMSE (Root Mean Square Error) between the reconstructed sensor values and the ground truth. To show the robustness of the reconstruction errors, we also report the corresponding standard errors.

4.1. Results on Weather Data

We used 30 weather stations from 10 geographical clusters. We generated random triplets across clusters. We generated each triplet in the following way: (1) randomly select two clusters; (2) randomly select two stations from the first cluster (denoted as the stations A1 and A2), and one station from the second cluster (denoted as the station B). We assumed that sensors in B were correlated with those in A1 and A2. We used sensors in A1 as the compound sensor, sensors in A2 as the new sensors, and sensors in B as the reference sensors. We generated 100 random triplets, and we report results averaged on them.

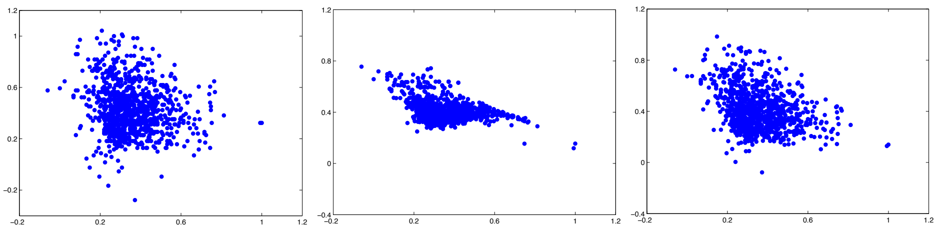
Each station consisted of six sensors including temperature (°F), humidity (%), dew point (°F), wind speed (mph), wind gust (mph), and pressure (Pa). Sensor values were collected every 5 min and are temporally aligned. Since A1 and A2 were from the same cluster, sensor values from A1 and A2 were typically more correlated than those from A1 (A2) and B. We used 2 years of data, with data in 2016 as the source domain and data in 2017 as the target domain.

Individual Sensor Changes: we treated each sensor in A1 as the replaced sensor, the remaining sensors in A1 plus all sensors in B as the reference sensors, and all sensors in A2 as the new sensors. Table 1 reports average reconstruction errors and the corresponding standard errors, with Imp. showing the average improvement (in %) of ASC over the best baseline method. We can see that ASC achieved an average improvement of 6.4% over baselines. This showed the high robustness of ASC. In general, ASC showed more statistically significant improvement on sensors whose values exhibited large variances (e.g., wind gust and pressure). Replace always underperformed compared to Refer, revealing that directly using new sensors could cause significant differences in sensor values. ReferZ performed better than Refer by leveraging new sensors. ASC further improved over ReferZ because it better exploited information from new sensors.

Table 1. Reconstruction errors (RMSE) on weather data for individual sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of ASC over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	Replace	Refer	ReferZ	ASC	Imp.
temperature	3.94 ± 0.024	0.61 ± 0.011	0.59 ± 0.009	**0.57 ± 0.010**	4.1
humidity	5.73 ± 0.023	**0.72 ± 0.016**	**0.71 ± 0.015**	**0.72 ± 0.015**	−1.7
dew point	3.89 ± 0.027	0.70 ± 0.010	**0.68 ± 0.009**	**0.67 ± 0.010**	2.8
wind speed	8.24 ± 0.084	**5.20 ± 0.063**	**5.21 ± 0.064**	**5.11 ± 0.060**	1.7
wind gust	10.81 ± 0.073	6.65 ± 0.052	6.65 ± 0.048	**6.31 ± 0.046**	5.0
pressure	7.82 ± 0.16	3.42 ± 0.19	2.48 ± 0.17	**1.83 ± 0.17**	26.2

Figure 5 visualizes the joint distributions over wind speed and reconstructed pressure on a station in San Francisco (x-axis: wind speed, y-axis: reconstructed pressure). Figure 5a is the ground-truth distribution that we would like to approximate after adaptation. As we can observe, Refer generated a significantly different joint distribution compared to the ground truth, while ASC produced a much closer distribution by leveraging new sensors.



(**a**) ground truth (**b**) Refer (**c**) ASC

Figure 5. Visualization of wind speed and reconstructed pressure on a weather station in San Francisco (x-axis: wind speed, y-axis: reconstructed pressure produced by different approaches). Ground truth corresponds to the true pressure values. Values are in normalized scales.

Compound Sensor Changes: we treated all sensors in A1 as the replaced sensors, all sensors in B as the reference sensors, and all sensors in A2 as the new sensors. Table 2 reports reconstruction errors on each replaced sensor separately. ASC statistically outperformed baselines in three cases, achieving an average improvement of 5.7%. Compared to Table 1, ASC produced larger reconstruction errors mainly because the reference sensors had lower correlations with the replaced sensors in this case.

Table 2. Reconstruction errors (RMSE) on weather data for compound sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of ASC over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	Imp.
temperature	3.94 ± 0.024	0.73 ± 0.018	0.71 ± 0.013	$\mathbf{0.68 \pm 0.014}$	4.2
humidity	5.73 ± 0.023	$\mathbf{0.87 \pm 0.021}$	$\mathbf{0.88 \pm 0.020}$	$\mathbf{0.87 \pm 0.022}$	0
dew point	3.89 ± 0.027	0.75 ± 0.018	$\mathbf{0.74 \pm 0.012}$	$\mathbf{0.72 \pm 0.011}$	2.6
wind speed	8.24 ± 0.084	$\mathbf{6.07 \pm 0.080}$	$\mathbf{6.11 \pm 0.074}$	$\mathbf{6.13 \pm 0.082}$	-1.8
wind gust	10.81 ± 0.073	7.24 ± 0.069	7.08 ± 0.072	$\mathbf{6.83 \pm 0.070}$	3.5
pressure	7.82 ± 0.16	3.82 ± 0.21	2.83 ± 0.20	$\mathbf{2.26 \pm 0.18}$	20.1

4.2. Results on UUV Data

We used the concatenated sensor values in 10 trips as the source domain, and the remaining as the target domain. We examined reconstruction errors on the surge (m/s), heave (m/s) and sway (m/s) sensors, whose sensor values are crucial for higher-layer software. To simulate new sensors, we used a biased version for each of the surge, heave and sway sensors. The biased version offset the original sensor values by a sensor-specific bias. We set the bias to 3σ, where σ is the standard deviation of the original sensor values.

Individual Sensor Changes: we treated each of the surge, heave and sway sensors as the replaced sensor, and the remaining sensors as the reference sensors. Table 3 compares reconstruction errors of different methods, where ASC improved over the best baseline by an average of 8.8%. The improvement on surge was the most statistically significant. Refer and ReferZ always outperformed Replace, consistent with our observations on weather data.

Table 3. Reconstruction errors (RMSE) on UUV data for individual sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of ASC over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	**Imp.**
surge	2.47 ± 0.14	0.66 ± 0.071	0.58 ± 0.048	$\mathbf{0.47 \pm 0.051}$	18.9
heave	0.13 ± 0.0068	$\mathbf{0.020 \pm 0.0062}$	$\mathbf{0.020 \pm 0.0046}$	$\mathbf{0.019 \pm 0.0049}$	6.5
sway	2.31 ± 0.13	$\mathbf{0.74 \pm 0.065}$	$\mathbf{0.72 \pm 0.059}$	$\mathbf{0.71 \pm 0.063}$	1.1

Compound Sensor Changes: we treated all sensors in the DVL compound sensor as the replaced sensors, and the propeller RPM and waterspeed sensors as the reference sensors. Table 4 reports the results. ASC improved over the best baseline by an average of 3.0%. Compared to Table 3, the improvement decreased for each sensor because fewer reference sensors were available.

Table 4. Reconstruction errors (RMSE) on UUV data for compound sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of ASC over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	**Imp.**
surge	2.47 ± 0.14	0.71 ± 0.084	0.67 ± 0.078	$\mathbf{0.62 \pm 0.081}$	6.0
heave	0.094 ± 0.0063	$\mathbf{0.026 \pm 0.0073}$	$\mathbf{0.026 \pm 0.0070}$	$\mathbf{0.024 \pm 0.0073}$	3.4
sway	2.31 ± 0.13	$\mathbf{0.78 \pm 0.079}$	$\mathbf{0.75 \pm 0.080}$	$\mathbf{0.75 \pm 0.076}$	-0.5

4.3. Evaluation in BRASS Project

In the evaluation of the BRASS Project [1] Phase 1, we conducted extensive experiments on Weather Underground Data. We organized data into 13 clusters, covering 13 regions in Los Angeles, San Francisco, Austin and Chicago. In each cluster, there were three weather stations, each containing 2 years of weather data. Five individual sensors (temperature, humidity, dew point, wind speed and wind gust) were used in all stations.

We evaluated our adaptation algorithms over randomly chosen clusters, stations, sensors, and time periods. Once a random cluster was chosen, we randomly picked two stations (A1 and A2). Since the two stations were from the same cluster, their sensor values were relatively similar. We further randomly picked an individual sensor from station A1, and replaced it with the same individual sensor from station A2. To enable adaptation, we used training data from a 2-month time period (without sensor change). We then used 1-month data for the adaptation period (sensor change happened in the beginning), and 1-month data for the evaluation period. The 4-month data were consecutive, as shown in Figure 6. The goal was to learn an adaptation function based on the data in the training and adaptation periods, and then evaluate adaptation performance in the evaluation period.

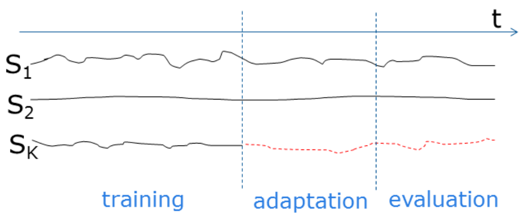


Figure 6. Illustration of training, adaptation and evaluation periods in the BRASS project evaluation. We use data from K sensors $(S_1, S_2, \cdots, S_K)$. The training period generates data without sensor change for 2 months. The adaptation period generates data for 1 month, in which S_K is replaced by a different sensor at the beginning. The evaluation period uses data from the next 1 month.

To determine whether an adaptation succeeds or not, we introduced a benchmark called the reference error. It defines a domain-specific baseline error bound that our system could tolerate. If adaptation error was less than the reference error, we considered the adaptation to be successful. In our implementation, we estimated the reference error by averaging the errors between every pair of weather stations in a cluster over the evaluation period.

Table 5 summarizes the adaptation performance on random tests described above. Our evaluation was performed on cases where the error of no adaptation (i.e., direct use of the new sensor) exceeded the reference error. ASC achieved high success rate on temperature, humidity, dew point and wind speed. On wind gust, the success rate was relatively low due to large variance in the sensor values. Despite performance drops on wind speed and wind gust, ASC showed positive improvement over reference error on all individual sensors.

Table 5. Adaptation performance on random tests in the BRASS project evaluation. Imp. shows the average improvement (in %) over the given reference error.

Sensor	Success Rate (%)	Imp.
temperature	95.4	61.6
humidity	96	65.8
dew point	100	71.1
wind speed	84.6	28.7
wind gust	66.7	24.0

Figure 7 shows the reconstructed wind gust in one random test. The blue curve represents the wind gust from a target station and a nearby station. The red curve represents the wind gust after adaptation, which was much more similar to the original signal in the training period.

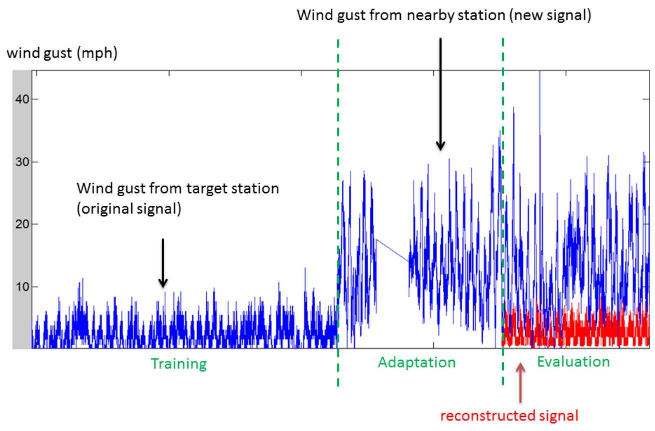


Figure 7. Visualization of the reconstructed wind gust by ASC. The blue curve represents observed sensor values and the red curve represents reconstructed sensor values.

5. Extensions

5.1. Estimating Adaptation Quality

To build survivable software, estimating the quality of adaptation is also important since it enables higher-layer software components to determine whether or not to accept a proposed adaptation. Towards this end, we developed a method to estimate an error interval for the gap between the reconstructed sensor value and the ground truth.

We would like to obtain such an error interval for each reconstructed sensor value and for each sample in the target domain. Given a reconstructed sample in the target domain $[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)]$ and a specific reconstructed sensor value, we estimated its error interval from similar samples in the source domain:

1. Find its κ nearest neighbors in the source domain according to distances defined in Equation (3).
2. Compute the standard deviation σ on the given reconstructed sensor value among the κ neighbors found in Step 1.
3. Set the estimated error interval to be $[-\alpha\sigma, \alpha\sigma]$, where $\alpha > 0$ is a scaling factor. An ideal α makes the error interval as tight as possible. α can be tuned on source-domain samples by optimizing the "excess error" notion defined below.

Excess Error of the Error Interval: To quantify the tightness of the estimated error interval, we used the notion of excess error. It is defined as the gap between the ground-truth value and the closest endpoint of the error interval, when the interval contains the ground-truth value. Figure 8 illustrates this notion. If the interval did not contain the ground-truth value, we considered the interval invalid. In practice, we could tolerate a small failure rate of the estimated error interval by setting a recall parameter (e.g., 90%). We could then find the smallest α to achieve the given recall and compute the corresponding excess error. Clearly, we favored a smaller excess error as it resulted in a tighter error interval. We present the results on excess errors in the next section.

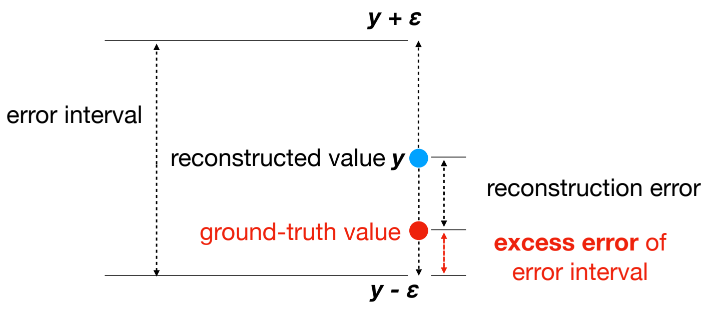


Figure 8. Notion of excess error of the error interval.

5.2. Ability to Exploit Many Sensors

As an increasing number of sensors are deployed in real-world systems, it is crucial for ASC to be able to exploit many sensors. This also enables our approach to be deployed in an open environment where new sensors continually emerge. Dealing with a large number of sensors is challenging in two aspects:

- Noisy sensors are likely to be involved and can degrade adaptation performance. For example, if some reference sensors produce highly noisy values, the nearest neighbor distances can suffer from the noise. Additionally, noisy values in reference or new sensors can cause the optimization algorithm to get stuck in poor local minima.
- A large number of sensors leads to a large parameter space of $\mathbf{\Theta}$, which significantly increases the computational cost of the adaptation algorithm.

In addressing these issues, we developed a two-step procedure to select a subset of useful sensors:

1. Selecting a subset of reference sensors: For each reference sensor, compute the average correlation between its sensor values and those from each replaced sensor, and then select N_{ref} reference sensors with the largest average correlation scores.
2. Selecting a subset of new sensors: For each new sensor, compute the average correlation between its sensor values and those from each replaced sensor as well as each selected reference sensor in Step 1, and then select N_{new} new sensors with the largest average correlation scores.

Here, N_{ref} and N_{new} were set by the user in specific applications. We denote this improved approach as ASC^{SEL}.

5.3. Empirical Study

We used the same triplets (A1, A2, B) as in Section 4.1. For each triplet, we used station A1 as the compound sensor, and simulated reference sensors and new sensors from the remaining 29 stations. Specifically, sensors from 15 randomly selected stations were used as reference sensors, and sensors from the other 14 stations were used as new sensors. This made the total number of sensors exceed 200. Some stations had additional types of sensors, e.g., precipitation. Table 6 reports the results for individual sensor changes, where ASC^{SEL} uses $N_{\text{ref}} = N_{\text{new}} = 10$. In terms of reconstruction errors, ASC^{SEL} achieved statistically significant improvement over ASC in all cases. Note that ASC^{SEL} outperformed ASC from Table 1, which revealed that a large pool of reference and new sensors actually helped. In contrast, ASC from Table 6 performed worse than itself from Table 1 due to overfitting. This demonstrated the efficacy of our sensor selection procedure when the number of sensors was large. In terms of excess errors, ASC^{SEL} achieved smaller values than ASC, consistent with the fact that ASC^{SEL} learned better reconstruction functions. The excess errors on wind speed and wind gust were relatively large because these sensor values exhibited large variances and were difficult to reconstruct. We observed similar trends in the scenario of compound sensor changes.

Table 6. Individual sensor changes on weather data with many sensors. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Replaced Sensor	Reconstruction Error		Excess Error	
	ASC	ASC^{SEL}	ASC	ASC^{SEL}
temperature (°F)	0.47 ± 0.012	**0.38 ± 0.009**	0.34 ± 0.010	**0.22 ± 0.009**
humidity (%)	0.53 ± 0.016	**0.47 ± 0.014**	0.42 ± 0.014	**0.31 ± 0.011**
dew point (°F)	0.47 ± 0.012	**0.44 ± 0.009**	0.37 ± 0.010	**0.25 ± 0.009**
wind speed (mph)	5.04 ± 0.061	**4.83 ± 0.059**	4.36 ± 0.052	**3.71 ± 0.055**
wind gust (mph)	6.28 ± 0.052	**5.61 ± 0.045**	4.75 ± 0.041	**3.96 ± 0.042**
pressure (Pa)	3.17 ± 0.19	**1.68 ± 0.18**	2.68 ± 0.19	**1.04 ± 0.18**

6. Related Work

Sensor failures and changes can be detected by identifying abrupt changes in time series of sensor readings. This problem is often called change point detection, and it has attracted researchers in statistics and data mining communities for decades [7–10,17]. Change point detection has broad applications in fraud detection, network intrusion detection, motion detection in vision, fault detection in controlled systems, etc. Change point detection methods can mainly be categorized into two types: supervised and unsupervised. Supervised methods treat change point detection as a classification problem and classify sensor readings into different states learned from training data [18–21]. Unsupervised methods, on the other hand, are capable of handling a variety of different states without prior training for each state. Examples of unsupervised methods include distribution-based [22], reconstruction-based [23,24], probabilistic [2,3,11,25], and distance-based [26,27] methods.

Existing work examining sensor failures and changes mainly focuses on detecting change points but rarely addresses the issue of adaptation to sensor failures or changes. Existing approaches typically rely on human experts to examine these change points and make subsequent decisions. Our work, on the other hand, is motivated by the notion of survivable software and aims at automatic adaptation to changes [28]. Such adaptation allows our approach to exploit the new sensors which may contain valuable information [29,30]. Although some of the existing detection methods [2,3,11] can be used to reconstruct sensor readings because they infer the actual readings through their models, they are not able to leverage any new sensor(s). The studies of [31,32] also address adaptation to sensor change; however, they require an overlapping period of time between the new sensors and the replaced sensors.

Our approach can be viewed as a special case of heterogeneous domain adaptation [15] if we treat sensor values of the replaced sensors as labels and sensor values of the reference/new sensors as features [33]. However, existing heterogeneous domain adaptation approaches are not capable of solving our problem, where the target domain has new features that are unseen in the source domain.

The notion of survivable software is similar to self-aware software [34], which requires a system to be aware of itself [35]. Self-awareness requires a system to experiment, model, hypothesize, and adapt its configuration and behavior. The goal is to improve the reliability and correctness of a software system in environments with high complexity. Recent work has studied self-aware software in Internet of Things [36,37], robotic and space applications [38,39], surveillance and security [40,41], and cloud and data centers [42,43]. Our work can be viewed as a building block for self-aware systems as these systems are often built upon sensor data.

Our work is also related to autonomic computing, in which a system is automatically configured in response to external changes, thereby removing the complexity of explicit user management of system resources [44–46]. However, our work focuses on interpreting configuration changes—particularly, sensor changes—in a modular fashion so that higher-layer software modules can function smoothly.

7. Conclusions

In this paper, we were the first to study a critical problem in building survivable software, i.e., how to automatically adapt to sensor changes in which a set of sensors are replaced by new sensors. To address this problem, we proposed a machine learning approach, called ASC, that is capable of exploiting new sensors, scaling to a large number of sensors, and estimating adaptation quality. ASC learns a reconstruction function to preserve sensor value distributions before and after the sensor change(s). It dynamically estimates the adaptation quality, thereby enabling higher-layer software components to determine whether or not to accept an adaptation. It also scales to a large number of sensors by intelligently selecting an important subset of relevant sensors using machine learning methods. This procedure significantly reduces overfitting to noisy values as well as the overall computational cost. Furthermore, ASC continually exploits new sensors in an open life-long environment. For empirical studies, we evaluated ASC on sensor data from the weather and UUV domains. In most of the evaluation cases, ASC outperformed other baseline methods. Our work is of highest relevance to researchers and practitioners working in the areas of Software Systems, Internet of Things, and Machine Learning.

Discussion: The underlying assumption of our approach is that sensor values from a subset of sensors are well correlated. Although this assumption often holds in real-world systems, it may not always be the case. This can be viewed as a limitation of our approach when the correlations among sensors are weak. However, such a limitation can often be overcome in practice if we are allowed to access or install more reference sensors that are better correlated with existing sensors.

Future Work: We would like to explore two directions in our future work. The first is to apply our ideas to new domains with larger volumes of sensor data. For example, we plan to examine the aviation domain where sensor values are sampled in milliseconds. The second is to integrate our methods into survivable software systems that operate in real-world scenarios.

Author Contributions: Conceptualization, Y.S., T.K.S.K. and C.A.K.; methodology, Y.S., T.K.S.K. and C.A.K.; software, Y.S.; Validation, Y.S. and A.L.; formal analysis, Y.S., A.L., T.K.S.K. and C.A.K.; investigation, Y.S., A.L., T.K.S.K. and C.A.K.; resources, Y.S., A.L., T.K.S.K. and C.A.K.; data curation, Y.S., A.L., T.K.S.K. and C.A.K.; writing—original draft preparation, Y.S.; writing—review and editing, Y.S., A.L., T.K.S.K. and C.A.K.; visualization, Y.S. and A.L.; supervision, T.K.S.K. and C.A.K.; project administration, T.K.S.K. and C.A.K.; funding acquisition, C.A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-16-C-0045.

Data Availability Statement: The weather data are available at https://github.com/usc-isi-i2/sensor-adaptation/tree/master/datasets/weather/dataset, accessed on 17 May 2021. The UUV data are not publicly available due to possible DARPA restrictions.

Acknowledgments: This material is based upon work supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-16-C-0045. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or DARPA.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hughes, J.; Sparks, C.; Stoughton, A.; Parikh, R.; Reuther, A.; Jagannathan, S. Building Resource Adaptive Software Systems (BRASS): Objectives and System Evaluation. *ACM SIGSOFT Softw. Eng. Notes* **2016**, *41*, 1–2. [CrossRef]
2. Dereszynski, E.W.; Dietterich, T.G. Probabilistic Models for Anomaly Detection in Remote Sensor Data Streams. *arXiv* **2012**, arXiv:1206.5250.
3. Dereszynski, E.W.; Dietterich, T.G. Spatiotemporal Models for Data-Anomaly Detection in Dynamic Environmental Monitoring Campaigns. *ACM Trans. Sens. Netw. TOSN* **2011**, *8*, 3. [CrossRef]

4. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
5. Tong, B.; Wang, G.; Zhang, W.; Wang, C. Node Reclamation and Replacement for Long-Lived Sensor Networks. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1550–1563. [CrossRef]
6. Lai, T.T.T.; Chen, W.J.; Li, K.H.; Huang, P.; Chu, H.H. Triopusnet: Automating Wireless Sensor Network Deployment and Replacement in Pipeline Monitoring. In Proceedings of the 11th International Conference on Information Processing in Sensor Networks, Beijing, China, 16–19 April 2012; pp. 61–72.
7. Basseville, M.; Nikiforov, I.V. *Detection of Abrupt Changes: Theory and Application*; Prentice Hall Englewood Cliffs: Hoboken, NJ, USA, 1993; Volume 104.
8. Gustafsson, F.; Gustafsson, F. *Adaptive Filtering and Change Detection*; Wiley: New York, NY, USA, 2000; Volume 1.
9. Brodsky, E.; Darkhovsky, B.S. *Nonparametric Methods in Change Point Problems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013; Volume 243.
10. Aminikhanghahi, S.; Cook, D.J. A Survey of Methods for Time Series Change Point Detection. *Knowl. Inf. Syst.* **2016**, *51*, 1–29. [CrossRef]
11. Dietterich, T.G.; Dereszynski, E.W.; Hutchinson, R.A.; Sheldon, D.R. Machine Learning for Computational Sustainability. In Proceedings of the International Green Computing Conference, San Jose, CA, USA, 4–8 June 2012.
12. Lawrence, M.G. The Relationship Between Relative Humidity and the Dewpoint Temperature in Moist Air: A Simple Conversion and Applications. *Bull. Am. Meteorol. Soc.* **2005**, *86*, 225–233. [CrossRef]
13. Friedman, J.; Hastie, T.; Tibshirani, R. *The Elements of Statistical Learning*; Springer Series in Statistics; Springer: Berlin/Heidelberg, Germany, 2001; Volume 1.
14. Shi, Y. Learning to Adapt to Sensor Changes and Failures. Ph.D. Thesis, Computer Science Department, University of Southern California, Los Angeles, CA, USA, 2019.
15. Pan, S.J.; Yang, Q. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1345–1359. [CrossRef]
16. Elnahrawy, E.; Nath, B. Context-Aware Sensors. In *European Workshop on Wireless Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 77–93.
17. Pimentel, M.A.; Clifton, D.A.; Clifton, L.; Tarassenko, L. A Review of Novelty Detection. *Signal Process.* **2014**, *99*, 215–249. [CrossRef]
18. Reddy, S.; Mun, M.; Burke, J.; Estrin, D.; Hansen, M.; Srivastava, M. Using Mobile Phones to Determine Transportation Modes. *ACM Trans. Sens. Netw. TOSN* **2010**, *6*, 13. [CrossRef]
19. Zheng, Y.; Liu, L.; Wang, L.; Xie, X. Learning Transportation Mode from Raw GPS Data for Geographic Applications on the Web. In Proceedings of the 17th International Conference on World Wide Web, Beijing, China, 21–25 April 2008; pp. 247–256.
20. Cleland, I.; Han, M.; Nugent, C.; Lee, H.; McClean, S.; Zhang, S.; Lee, S. Evaluation of Prompted Annotation of Activity Data Recorded from a Smart Phone. *Sensors* **2014**, *14*, 15861–15879. [CrossRef]
21. Han, M.; Vinh, L.T.; Lee, Y.K.; Lee, S. Comprehensive Context Recognizer Based on Multimodal Sensors in a Smartphone. *Sensors* **2012**, *12*, 12588–12605. [CrossRef]
22. Harchaoui, Z.; Moulines, E.; Bach, F.R. Kernel Change-Point Analysis. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*; Vancouver, BC, Canada, 8–11 December 2008; pp. 609–616.
23. Idé, T.; Tsuda, K. Change-Point Detection Using Krylov Subspace Learning. In Proceedings of the 2007 SIAM International Conference on Data Mining, Minneapolis, MN, USA, 26–28 April 2007; pp. 515–520.
24. Moskvina, V.; Zhigljavsky, A. An Algorithm Based on Singular Spectrum Analysis for Change-Point Detection. *Commun. Stat. Simul. Comput.* **2003**, *32*, 319–352. [CrossRef]
25. Saatçi, Y.; Turner, R.D.; Rasmussen, C.E. Gaussian Process Change Point Models. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 927–934.
26. Angiulli, F.; Pizzuti, C. Fast Outlier Detection in High Dimensional Spaces. In Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery, Helsinki, Finland, 19–23 August 2002; pp. 15–27.
27. Chen, H.; Zhang, N. Graph-Based Change-Point Detection. *Ann. Stat.* **2015**, *43*, 139–176. [CrossRef]
28. Neema, S.; Parikh, R.; Jagannathan, S. Building Resource Adaptive Software Systems. *IEEE Softw.* **2019**, *36*, 103–109. [CrossRef]
29. Grosse-Puppendahl, T.; Berlin, E.; Borazio, M. Enhancing Accelerometer-Based Activity Recognition with Capacitive Proximity Sensing. In *International Joint Conference on Ambient Intelligence*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 17–32.
30. Chen, C.; Jafari, R.; Kehtarnavaz, N. Improving Human Action Recognition Using Fusion of Depth Camera and Inertial Sensors. *IEEE Trans. Hum. Mach. Syst.* **2014**, *45*, 51–61. [CrossRef]
31. Hu, C.; Chen, Y.; Peng, X.; Yu, H.; Gao, C.; Hu, L. A Novel Feature Incremental Learning Method for Sensor-Based Activity Recognition. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 1038–1050. [CrossRef]
32. Hou, B.J.; Zhang, L.; Zhou, Z.H. Learning with Feature Evolvable Streams. *IEEE Trans. Knowl. Data Eng.* **2019**. [CrossRef]
33. Shi, Y.; Knoblock, C. Learning with Previously Unseen Features. In Proceedings of the International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017.
34. Kounev, S.; Lewis, P.; Bellman, K.L.; Bencomo, N.; Camara, J.; Diaconescu, A.; Esterle, L.; Geihs, K.; Giese, H.; Götz, S.; Inverardi, P.; et al. The Notion of Self-Aware Computing. In *Self-Aware Computing Systems*; Springer: Cham, Switzerland, 2017; pp. 3–16.

35. Esterle, L.; Rinner, B. An Architecture for Self-Aware IoT Applications. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 6588–6592.
36. Möstl, M.; Schlatow, J.; Ernst, R.; Hoffmann, H.; Merchant, A.; Shraer, A. Self-Aware Systems for the Internet-of-Things. In Proceedings of the 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), Pittsburgh, PA, USA, 2–7 October 2016; pp. 1–9.
37. Chen, B.W.; Imran, M.; Nasser, N.; Shoaib, M. Self-Aware Autonomous City: From Sensing to Planning. *IEEE Commun. Mag.* **2019**, *57*, 33–39. [CrossRef]
38. Mundhenk, T.N.; Everist, J.; Landauer, C.; Itti, L.; Bellman, K. Distributed Biologically-Based Real-Time Tracking in the Absence of Prior Target Information. In Proceedings of the Intelligent Robots and Computer Vision XXIII: Algorithms, Techniques, and Active Vision, Boston, MA, USA, 23–26 October 2005; Volume 6006.
39. Landauer, C.; Bellman, K.L. An Architecture for Self-Awareness Experiments. In Proceedings of the 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, USA, 17–21 July 2017; pp. 255–262.
40. Guettatfi, Z.; Hübner, P.; Platzner, M.; Rinner, B. Computational Self-Awareness as Design Approach for Visual Sensor Nodes. In Proceedings of the 2017 12th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), Madrid, Spain, 12–14 July 2017; pp. 1–8.
41. Esterle, L.; Simonjan, J.; Nebehay, G.; Pflugfelder, R.; Domínguez, G.F.; Rinner, B. Self-Aware Object Tracking in Multi-Camera Networks. In *Self-Aware Computing Systems*; Springer: Cham, Switzerland, 2016; pp. 261–277.
42. Salama, M.; Shawish, A.; Bahsoon, R. Dynamic Modelling of Tactics Impact on the Stability of Self-Aware Cloud Architectures. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 27 June–2 July 2016; pp. 871–875.
43. Iosup, A.; Zhu, X.; Merchant, A.; Kalyvianaki, E.; Maggio, M.; Spinner, S.; Abdelzaher, T.; Mengshoel, O.; Bouchenak, S. Self-Awareness of Cloud Applications. In *Self-Aware Computing Systems*; Springer: Cham, Switzerland, 2017; pp. 575–610.
44. Parashar, M.; Hariri, S. *Autonomic Computing: Concepts, Infrastructure, and Applications*; CRC Press: Boca Raton, FL, USA, 2018.
45. Lalanda, P.; McCann, J.A.; Diaconescu, A. *Autonomic Computing: Principles, Design and Implementation*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013.
46. Abeywickrama, D.B.; Ovaska, E. A Survey of Autonomic Computing Methods in Digital Service Ecosystems. *Serv. Oriented Comput. Appl.* **2017**, *11*, 1–31. [CrossRef]