Fall Detection using SmartWatch Sensor Data with Accessor Architecture

Anne Ngu, Yeahuay Wu, Andrew Polican, Brock Yarbrough, Habil Zare, and Lina Yao
Texas State University, San Marcos

Some historical perspectives

- The launch of the Web has changed how people interact, how businesses are run and how information is exchanged.
- The launch of “app” framework such as Android and iOS brought another new wave of disruptive applications such as AirBnB, Uber, Waze, Instagram.
- We believe the launch of an equivalent of an “app” or framework (accessor host) for IoT will open up unanticipated range of applications that have impacts beyond our imagination.

Motivation

- Falls in the elderly cause ~26,000 deaths and $34 billion in medical costs annually.
- Existing fall detection systems use expensive custom trunk-mounted sensors.
- Can we use light-weight accessor programming model for fall detection so that it can run robustly on inexpensive IoT devices?

Related Works

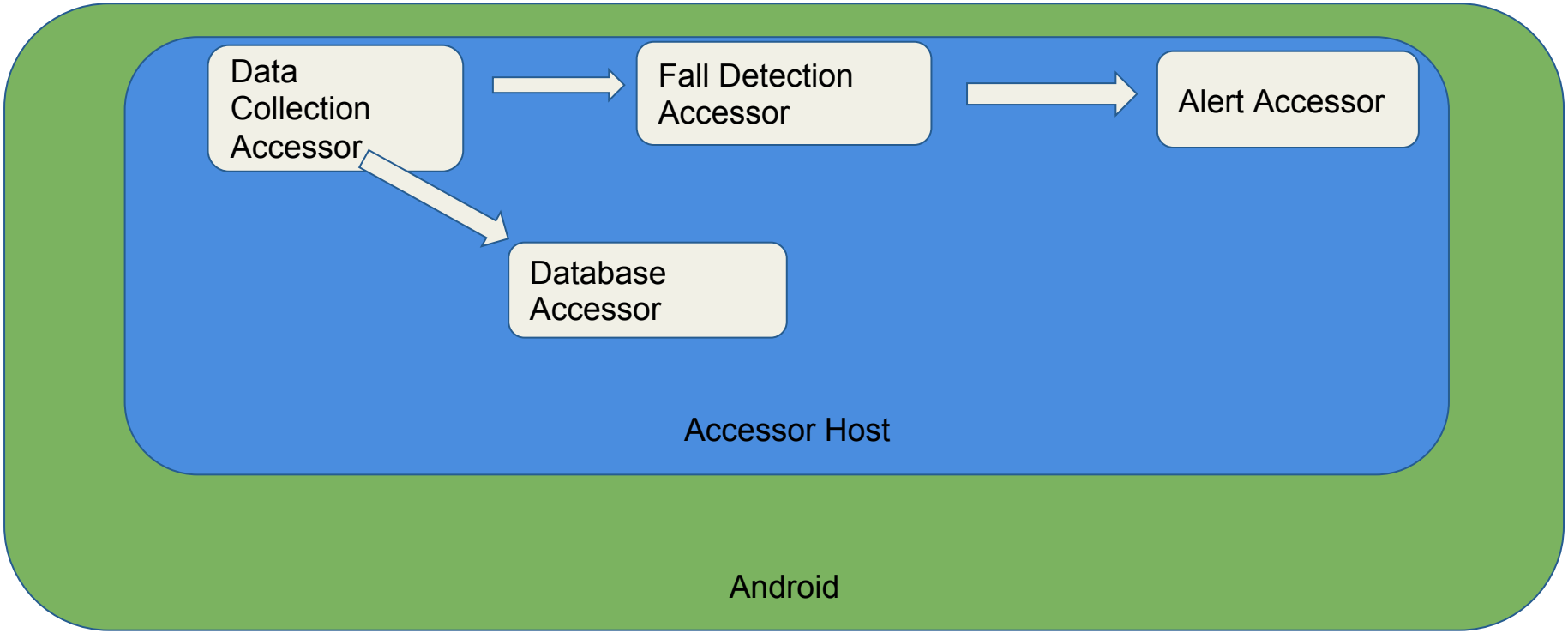
- Fall detection using specially built hardware by Liu and Cheng in 2012
- Fall detection using Support Vector Machine by Jantaraprim et. al. in 2012
- No prior work addressed fall detection using a smartwatch that can be worn as a piece of jewelry.
- No prior work addressed using a light programming modelfor fall detection application in small devices.

The challenges

- Fall detection needs to happen asap.
- Running fall detection software on the cloud incurs latency and make the application impractical to use
- Prediction application needs to run in heterogeneous devices (Fitbit, different brand of smartwatches)

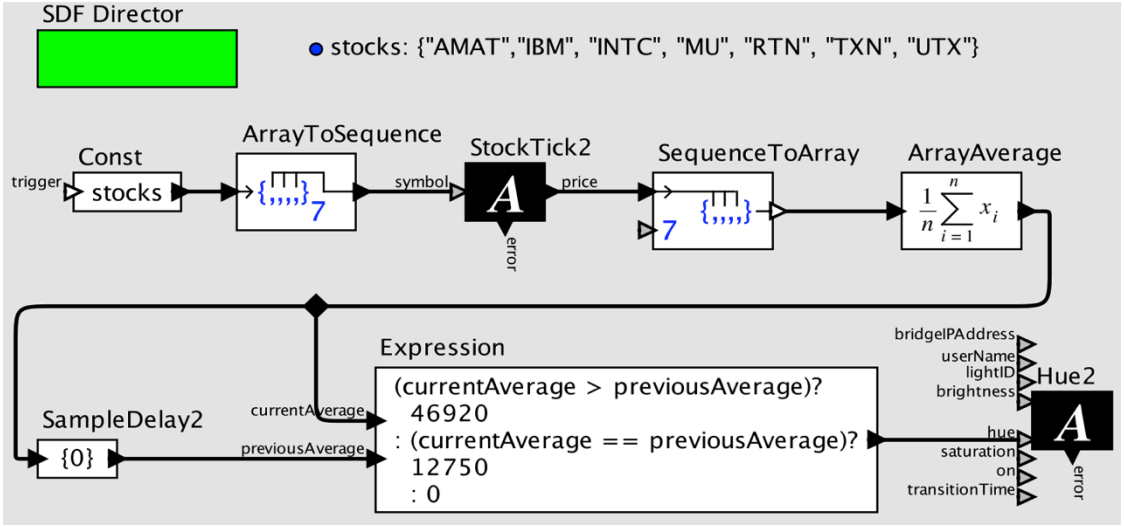
The Setup

Our Approach



Why Accessor host?

Platform independence, much like a web browser that can operates in different operating systems



What is an accessor?

- Accessor is a concurrent computing design pattern inspired by the Actor Oriented programming developed by the Industrial Cyber-Physical Systems Center (formerly known as Terraswam) at University of California, Berkeley.
- Accessors are implemented as Javascript objects that provide interfaces to hardware and software services.
- Accessors can be chained together into IoT applications or “Swarmlets”
- In summary, accessor embraces concurrency, atomicity and asynchrony (i.e. embed AAC in a streaming actor)

Accessors

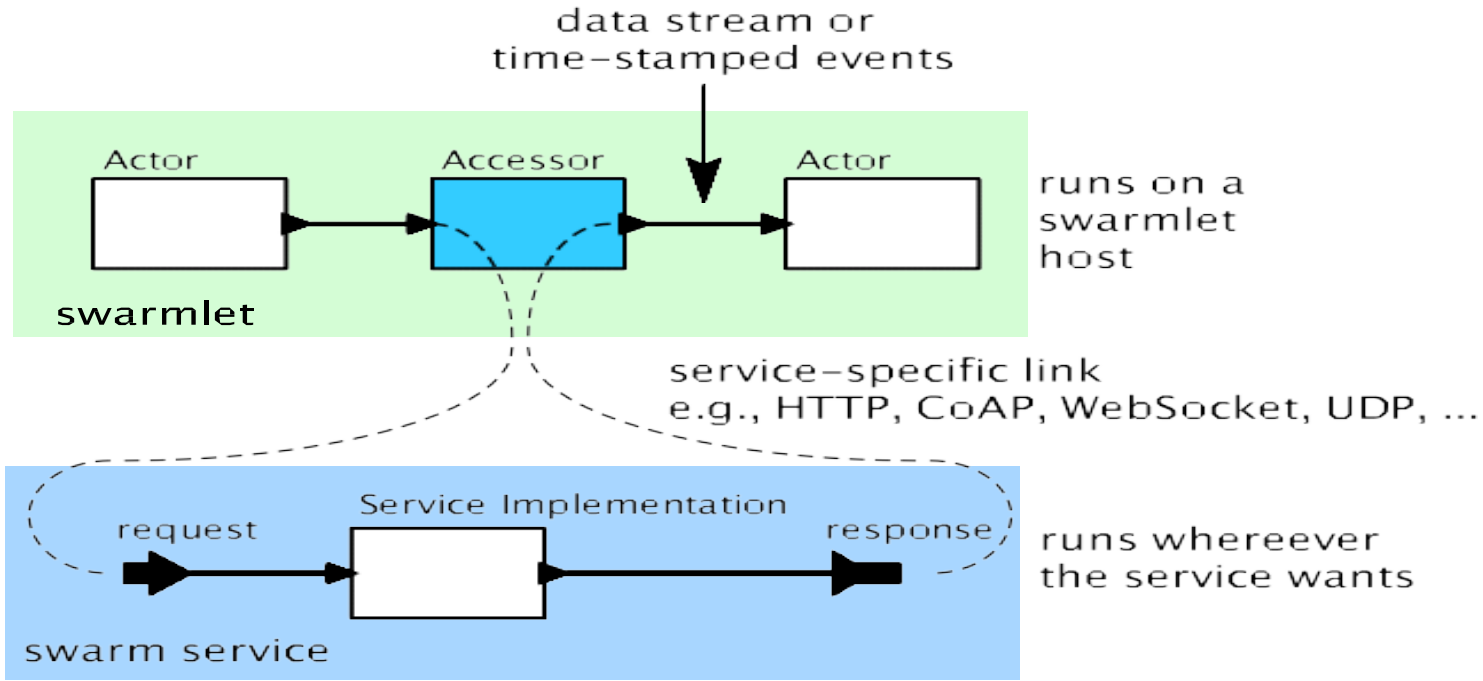


Fig. 1. Design pattern of accessors.

TEXAS STATE UNIVERSITY
SAN MARCOS
The rising STAR of Texas

Android J2V8 host

J2V8 is Google's V8 Javascript engine that provides interface to Java

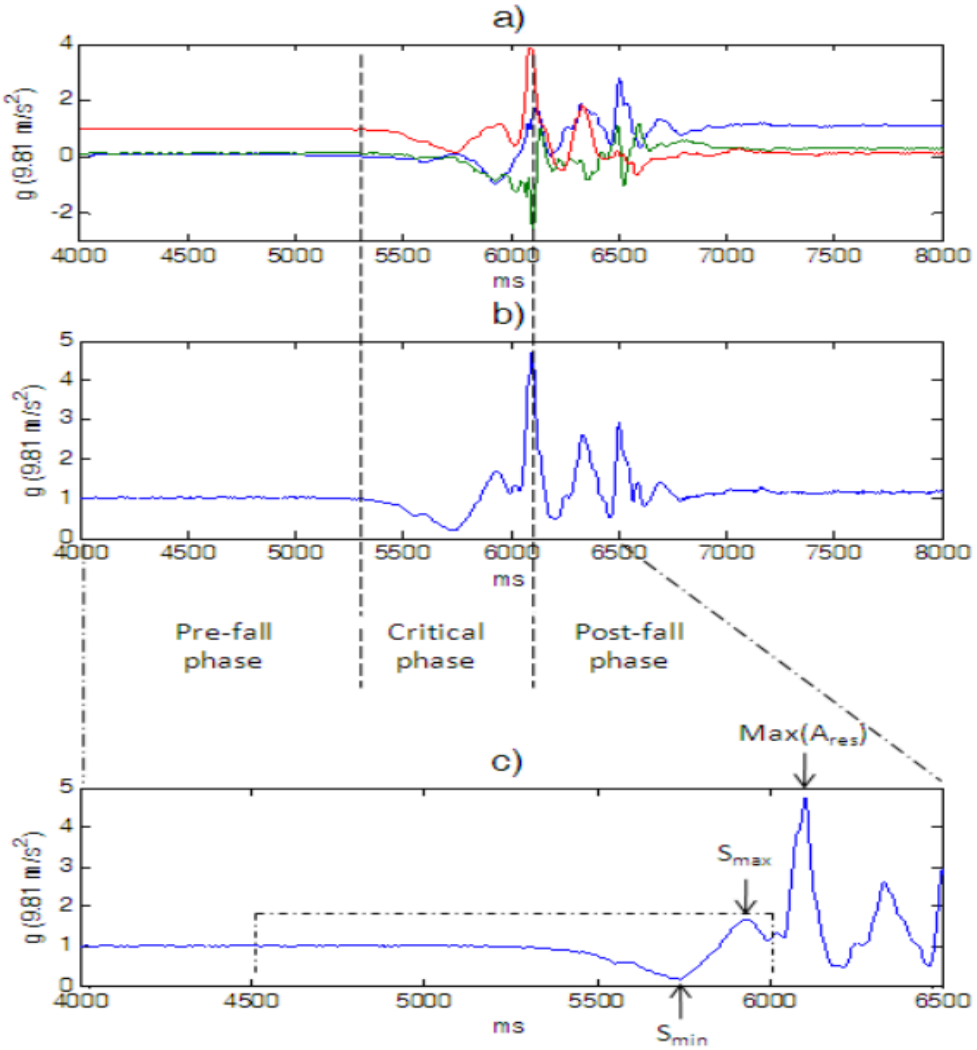
J2V8 is a high performance Javascript engine with small foot print.

Allows hardware interface through Java callbacks

Falls data collection

- Subjects were ages ranging from 19-29, all in good physical condition.
- Training set consisted of four types of falls: front, back, right, and left falls. For each type of fall there was one fast fall and one slow fall.
- Each Subject was told to wear the smartwatch on the dominant hand wrist and perform at least eight falls.
- Accelerometer data is collected every 250 ms.

Critical Phase of a fall



Features used

Resultant Acceleration: The norm of the acceleration vector

$$\|\boldsymbol{r}\|_2 = \sqrt{r_x^2 + r_y^2 + r_z^2}. \qquad A_{\text{res}} = \|A\|_2.$$

S-max and S-min: The maximum and minimum resultant acceleration in a 750 millisecond sliding window.

Change is S-max and S-min: $\Delta S = \|S_{\max} - S_{\min}\|_2.$

Fall Model Creation

We first trained our model for fall detection using Support Vector Machine (SVM) with an RBF Kernel.

We then trained a KNN with rectangular kernel with the same data in order to get a better idea of how an eager learner (SVM) would compare to a lazy learner (KNN).

Results

Method	SVM		KNN		
Prediction	Not fall	Fall	Not fall	Fall	Total
Actual not fall	511 (TP)	3 (FN)	509 (TP)	5 (FN)	514
Actual fall	11 (FP)	44 (TN)	17 (FP)	38 (TN)	55
Total	522	47	526	43	559

Algorithm	Accuracy	Recall	Precision	Specificity
SVM	98%	80%	93.6%	99%
KNN	96.13%	69%	88%	99.2%

Fall prediction

- We implemented two versions of our fall detection application using the trained model
- The first one is a native Android application in Java.
- The second is an accessor compliance application running in Android J2V8 Accessor Host.

Test Results

Accuracy = 93.8489% (534/569) w/ Testing File

Trained model using a sample that consisted of approx. 8% falls

Tested various configurations for testing files

- w/ greater percentage of falls causes accuracy to drop
- Smaller percentage causes accuracy to rise

How data is sampled

5 Participants go through a range of different types of falls

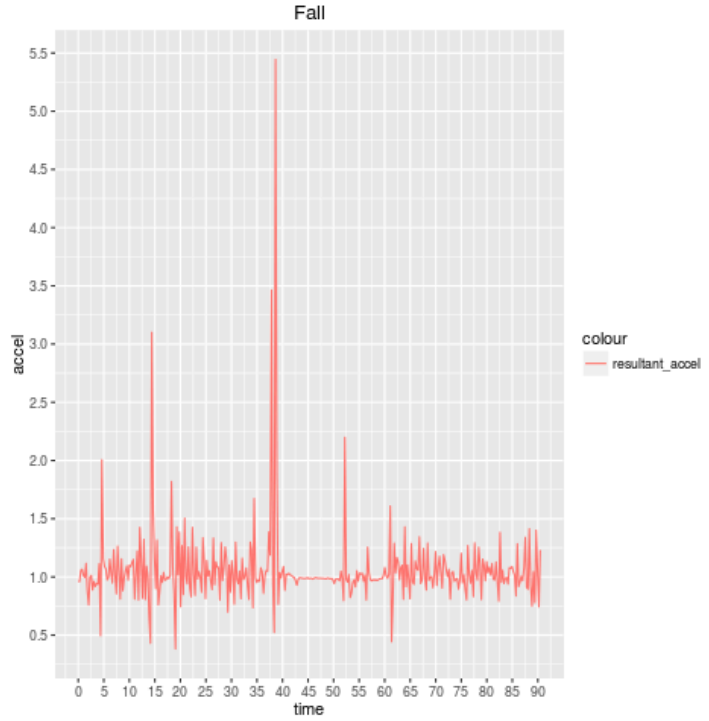
Backfall, Frontfall, L. Side fall, R Side Fall

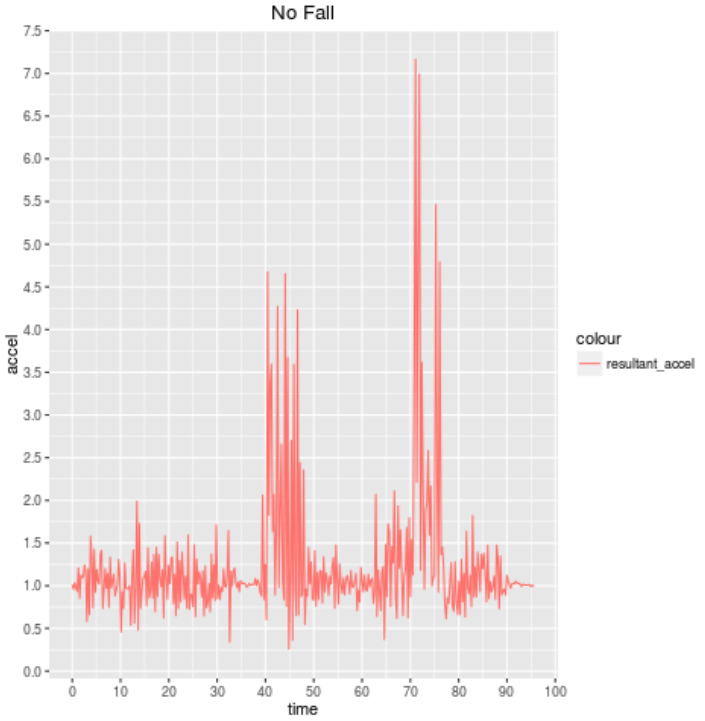
One slow fall and one fast fall

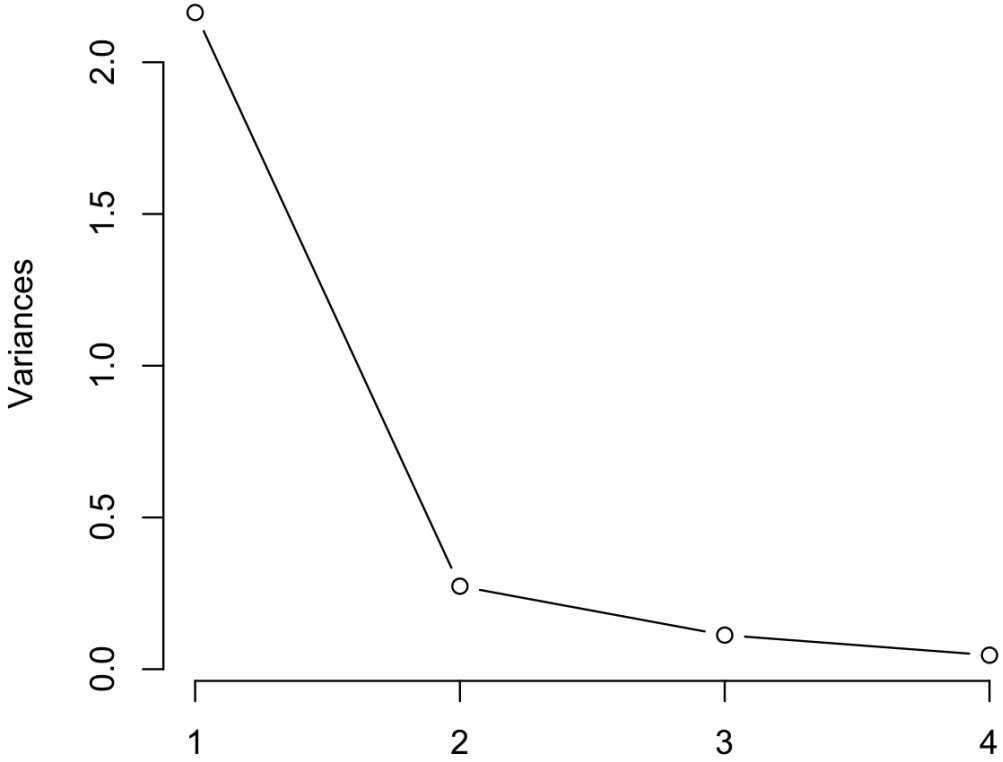
- Afterwards, each 250 ms sample is labeled as either falling or not falling.
- After a couple samples after participant hits ground, activity is counted as as not falling

TEXAS STATE UNIVERSITY
SAN MARCOS
The rising STAR of Texas

Sample Sequence






pc_train
Variances
0.0
0.5
1.0
1.5
2.0
1
2
3
4

Examining the results

~93% is great right?

Let's look at what was predicted wrongly in the test file

Puzzling Undetected Fall

```
1 1:0.783554 2:1.019721 3:0.944764 4:0.783554
1 1:0.944764 2:4.03615 3:4.09498 4:0.884236
1 1:0.884236 2:4.302216 3:4.09498 4:0.884236
1 1:4.09498 2:4.177086 3:4.09498 4:1.257684
1 1:1.341961 2:1.017734 3:1.341961 4:1.116681
1 1:1.257684 2:0.422339 3:1.257684 4:1.056105
```

Conclusions

SVM with our current features is a suitable algorithm for fall detection

Using commercial smartwatch sensors can yield fall detection results that are competitive with those of custom sensors

The accessor-based fall detection rivals native Android apps in both functionality and development time and it is more portable and 20% more energy efficient.

Questions and Comments?