A Telemonitoring Solution to Long-Distance Running Coaching

Eugene Song
Carlos Asuncion
Uma Balakrishnan
Hannah Sarver
Lucas Serven
Daniel Aranki, Ed.
Ruzena Bajcsy, Ed.
Ali Javey, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-99
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-99.html

May 13, 2016

Acknowledgement

Daniel Aranki, University of California, Berkeley
Professor Ruzena Bajcsy, University of California, Berkeley
Professor Ali Javey, University of California, Berkeley
Dr David Liebovitz, MD, University of Chicago Medicine

BERKELEY TELE-MONITORING

A Telemonitoring Solution to Long-Distance Running Coaching

Master of Engineering Capstone Report 2016

Eugene Song

with Carlos Asuncion, Uma Balakrishnan, Hannah Sarver, and Lucas Serven

Prepared for Professors Ruzena Bajcsy and Ali Javey

Contents

List of Figures

Introduction

My team's capstone project involves expanding an Android-based health telemonitoring framework [1][2] to incorporate new sensing, estimation, and data analysis tools that would be useful for tracking and advising on the training of marathon runners, and building a long distance coaching application using the aforementioned tools, under the guidance of PhD candidate Daniel Aranki and our advisor Professor Ruzena Bajcsy.

1 Overview of the Telemonitoring Framework

Shown in Figure 1, telemonitoring refers to a technology which can take readings about a subject or environment remotely, while indicating interventions in the form of modifying the environment or providing suggestions. This project is concerned with a particular implementation of telemonitoring as a robust, easy-to-use framework that can be used to build a variety of applications by non-experts in software engineering. For example, a doctor may desire an application for telemonitoring patients, only being interested in a particular set of devices, data, and analyses. In the current environment, if no such product exists, then the doctor must used existing products that are not ideal, contract out the development of the ideal application at a cost, or simply proceed without any application. With access to the framework, however, the doctor can quickly build an application that fulfills the requirements.

The framework primarily consists of two nodes: the client and the server. The client is the device and associated software that interacts directly with the environment being monitored,

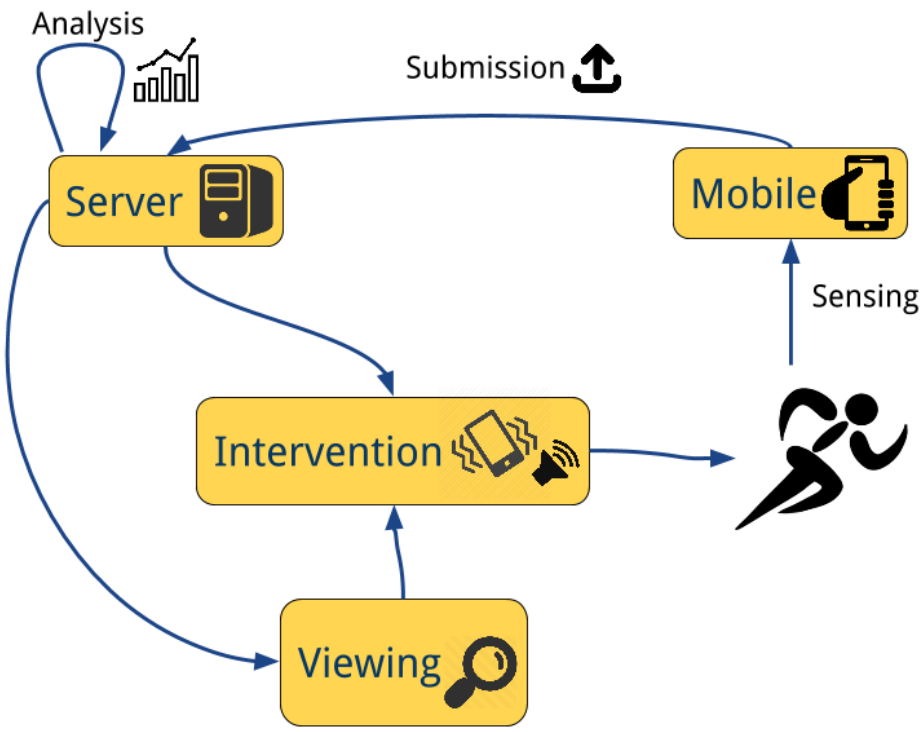


Figure 1: A telemontoring loop.

such as a smartphone. It collects data using objects called extractors (which read data from sensors) and estimators (which estimate secondary measurements from sensor readings), and issues interventions to the environment. The server receives data from the client, analyzes it, then automatically submits a relevant intervention back to the client, which indicates the intervention to its environment or subject. A human viewer may also interact with the server, monitoring the incoming data and manually submitting his or her own interventions.

2 Long-Distance Running Coaching

Up to 79% of runners are expected to sustain a running-related injury in the lower extremities [3], so they may benefit from applications developed on the framework. However, while the framework currently provides some health-related monitoring, it lacks the tools necessary to deliver assistance to long-distance running. We are interested in expanding the framework with additional features that can help improve performance and reduce injury of long-distance runners. We have found that the cadence of a runner (measured in steps

per minute) is an important performance metric. Proper cadence has been connected with reducing impact forces on joints [4], reducing muscle soreness and fatigue [5], and increasing efficiency of oxygen use [6]. The multitude of advantages to cadence control is apparent. Consequently, we are primarily interested in using a cell phone application to measure the cadences of runners and developing interventional models to adjust their cadences toward safer and more efficient values. Additionally, speed, geolocation, and heart-beat rate are of interest, as they may be useful in constructing more sophisticated interventional models in the future.

3 Description of Framework Expansion and Work Breakdown

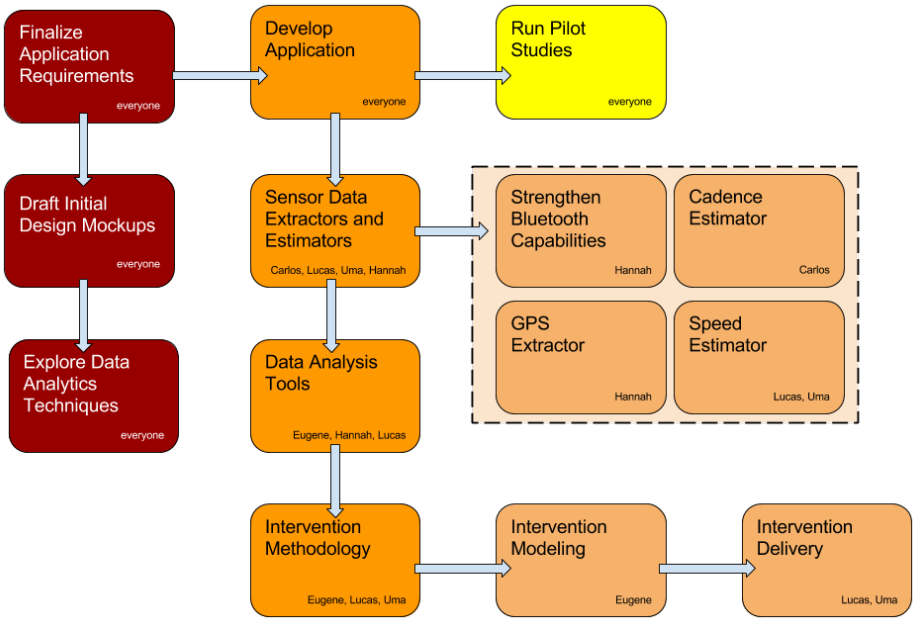


Figure 2: Breakdown of tasks between team members.

The work breakdown by team member is shown in Figure 2. The client-side development involved development of new extractors and estimators. Carlos Asuncion was responsible for developing the cadence estimator, while Lucas Servén and Uma Balakrishnan developed the speed estimator. Hannah Sarver implemented a new GPS extractor, and worked on enhancing the framework's Bluetooth stack. The server-side development consisted of implementing

new data analysis tools. Models for delivering personalized coaching were explored as well. I was involved in creating and implementing a model for a training regimen, while Lucas and Uma developed a method of delivering intervention. We collectively contributed to development of the final Android application. My contributions regarding the design and implementation of data analysis tools and the model of performance improvement will be discussed in this paper. Additionally, I will discuss the development of application features related to continuous subject monitoring.

Chapter 1

Individual Technical Contributions

1.1 Data Analysis Tools

1.1.1 Rationale for Analysis Tools

A major component of the work on the telemonitoring framework is to add the capability to advise on training trajectories. Statistical and machine learning tools for data analysis can be used in conjunction with training models to provide personalized training suggestions for each individual. To this end, a reusable toolkit for analysis needs to be implemented in the server library. This toolkit needs to be designed such that it can be reused for different training models and even in entirely different contexts. This would allow future developers to avoid having to re-implement the same software tools every time the same machine learning model needs to be applied to a different purpose.

1.1.2 Design of Data Analysis Tools

The tools for analysis needed to be designed in accordance with the high level design goals of maintaining modularity and re-usability in the context of the framework. This meant the following high level design goals:

1. Users of the analysis tools should not be concerned with the interfaces and data structures of different underlying models.

2. An analysis tool interface is generic and independent of the implementation of the underlying model.

3. Analysis tools are compatible with the data structures used by the framework.

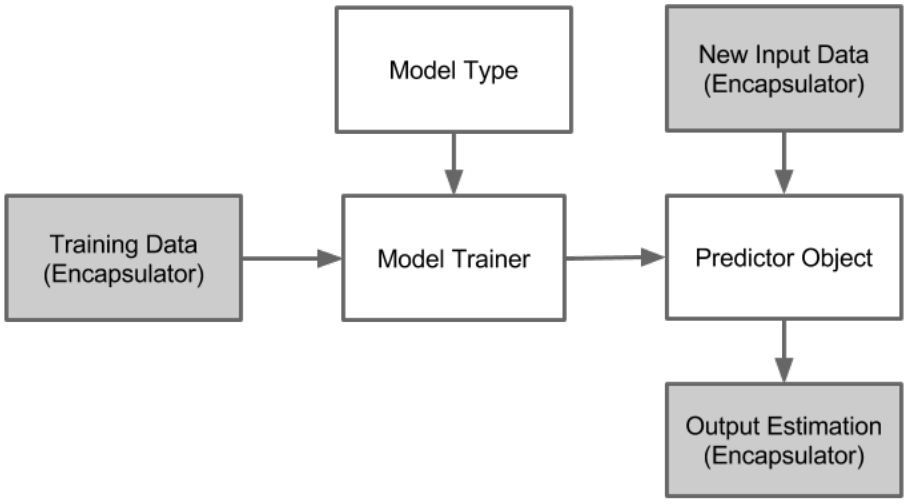


Figure 1.1: Usage flow of analysis tools. White blocks are the core elements of the analysis tools. Shaded blocks are input and output data.

The fulfillment of these design goals is shown in Figure 1.1. The core elements of the analysis tools shown are described as follows:

1. Model Type: A representation of the type of statistical or machine learning model which is being used.

2. Model Trainer: A service that trains the model and produces a reusable Predictor.

3. Predictor Object: An object that makes predictions based on given inputs, and can be reused as long as the model does not need to be retrained.

The shown data encapsulators, which are the basic data structures used by the framework, are used as follows:

1. Training Data: An encapsulator containing a set of inputs and known corresponding outputs used to train the model.

2. New Input Data: An encapsulator containing inputs for which corresponding outputs are unknown but desired.

3. Output Estimation: An encapsulator containing the predicted outputs corresponding to the new input data.

In this case, the data never interacts directly with the model type. This is the primary advantage of this modular design. Through a common interface, trainers and predictors can be reused without concern about the implementation details of the underlying model type.

1.1.3 Example Implementation: Least Squares Linear Regression

Library and Model Selection

Due to its simplicity and ubiquity, linear regression served as a useful model for testing the analysis tools design. Because linear regression is so commonplace, numerous implementations already exist in various Java libraries. To avoid wasted effort in reimplementing something so common, no custom linear regression implementation was made within the framework. Instead, one of the existing Java classes from these libraries was chosen to be adapted. When choosing the library, major considerations were the scope of the library components and the freedom offered by its license. For these reasons, the Apache Commons mathematics library [7] was the overwhelmingly favorable choice. It contained a comprehensive library of commonly used mathematical, statistical, and machine learning functions implemented in Java, including least squares linear regression. Additionally, as an open source project intended for expansive community reuse, it had a very permissive license.

Even within the Apache Commons library, there were multiple packages for different mathematical domains. Naturally, the Least Squares package was explored first. Because this package was designed for general least squares work and not just for linear regression, it actually contained a set of very generalizable features potentially useful to any model that might employ a least squares optimization. Unfortunately, complexity followed its generalizability, as several different objects had to be used to construct even a simple linear

regression model. Additionally, due to poor and seemingly inconsistent documentation, it was difficult to understand how to correctly use these particular components of the library. This package was eventually abandoned because of these reasons.

Fortunately, an alternative was found in the Statistics package. A single Java class, called `OLSMultipleLinearRegression`, could take training data and generate a set of linear regression model parameters. In contrast with the previously discussed option, `OLSMultipleLinearRegression` was simple to use, but at the cost of generalizability, as it functioned solely to perform ordinary least squares linear regression. However, because of proper alignment to design goals, the effect of this cost was mitigated in the context of the telemonitoring framework. The software design and this mitigating effect will be discussed in the sections below.

Incorporation into the Framework

For training, an `OLSMultipleLinearRegression` object needs Java arrays of known inputs and outputs. Once trained, it outputs estimated regression parameters as a Java array as well. This is already an issue as the framework does not handle data directly as Java arrays, but as encapsulators instead. Every explicit use of `OLSMultipleLinearRegression` would require the user to convert the encapsulators to arrays, which violates the design goals.

Instead, `OLSMultipleLinearRegression` is abstracted as a model type that is given to a model trainer. The functional details of `OLSMultipleLinearRegression` are therefore hidden, and the user of the analysis tools only needs to be concerned about training the model through the generic interface offered by the model trainer. The resulting predictor serves a similar purpose. The user no longer has to know about the array of regression parameters produced by `OLSMultipleLinearRegression`, instead simply requesting the predicted output with an encapsulator of inputs.

An additional consideration was how the regression data could be represented in the framework. A sufficiently generic interface to the analysis tools also required a sufficiently generic and reusable data container to store data points. While encapsulators are the frame-

work's common data structure, they are capable of holding various types of data containers. If different types of data containers were to be used for different types of models, then generic interface design would have become impossible. With this in mind, a simple data container was designed specifically for use with the analysis tools. This container could hold an arbitrary number of input values, and a single corresponding output. These data can be read from and written to using corresponding getter and setter methods. In this way, any model that uses one or more independent variables with a single dependent variable will be able to use this data container.

Through this example, the benefit of adherence to the design goals becomes clear. The user of the framework is able to enjoy a generalized set of interfaces for modeling, without having to be concerned with the underlying implementation. In the eventual library of data analysis tools, other model types may deal with different data types and require different function calls to work. However, protected from these details, the user is free to swap between different model types without significant code changes.

1.2 Performance Improvement Model

1.2.1 Rationale for the Improvement Model

As one of the goals of this project was to provide automated coaching for marathon runners, a model for training and improvement needed to be developed and implemented for the runners to follow. This model would indicate performance targets with respect to time in the training regimen. Additionally, this model should be adjustable to suit each individual runner, per the goals of the marathon coaching application. Exploration of different models will eventually be conducted, but one such model was initially formulated and implemented in the application. This particular model is discussed below.

1.2.2 Formulating the Improvement Model

Under particular aerobic training regimens, a performance improvement trajectory has been observed that is characterized by a tapering of improvement rate [8]. We hypothesized that a similar progression would also be applicable to cadence and sought to develop an improvement trajectory that had the characteristic of a high initial rate of improvement that transitions into a lower rate of improvement.

Generally, a negative exponential model possesses this desired characteristic. Our particular negative exponential model would also need to output the runner's starting performance as an initial value and the performance goal after a desired period of time.

For a given performance metric, this negative exponential model for performance can be represented in the general form:

$$f(C, G, R, \alpha, t) = A(C, G, R, \alpha) - B(C, G, R, \alpha)e^{-\alpha t}$$

where

- C = current performance
- G = performance goal
- R = remaining time to reach performance goal
- α = time constant to modify the model curvature

To fulfill the initial and final output requirements, A and B can be solved such that the curve starts at C at $t = 0$, and goes to G at $t = R$:

$$f\left(C, G, R, \alpha, t\right) = \frac{Ge^{\alpha R} - C}{e^{\alpha R} - 1} - \frac{C - G}{e^{-\alpha R} - 1}e^{-\alpha t}$$

This can be rearranged to:

$$f\left(C, G, R, \alpha, t\right) = \frac{Ce^{-\alpha R} - G + \left(G - C\right)e^{-\alpha t}}{e^{-\alpha R} - 1}$$

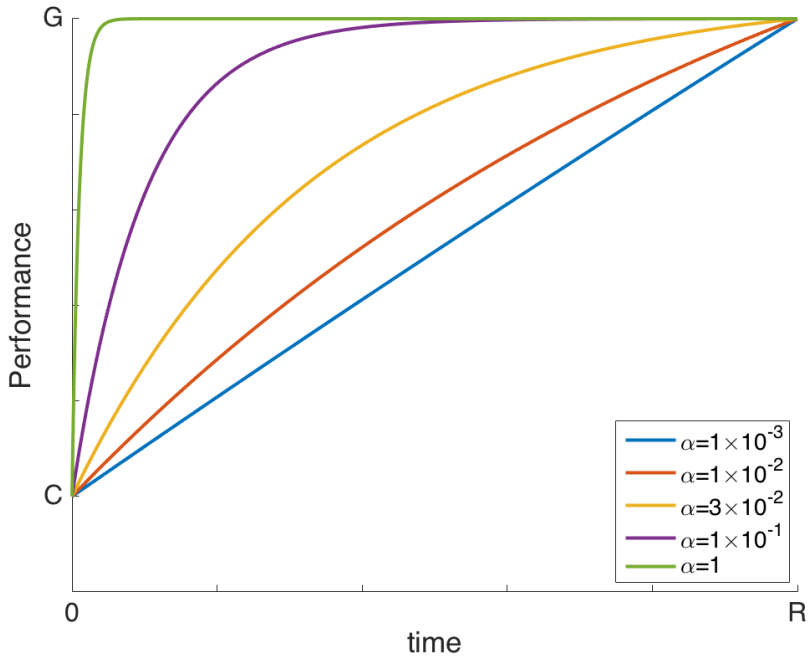


Figure 1.2: Models with various values of α. The model becomes more aggressive as α increases, while it approaches a line as α approaches 0.

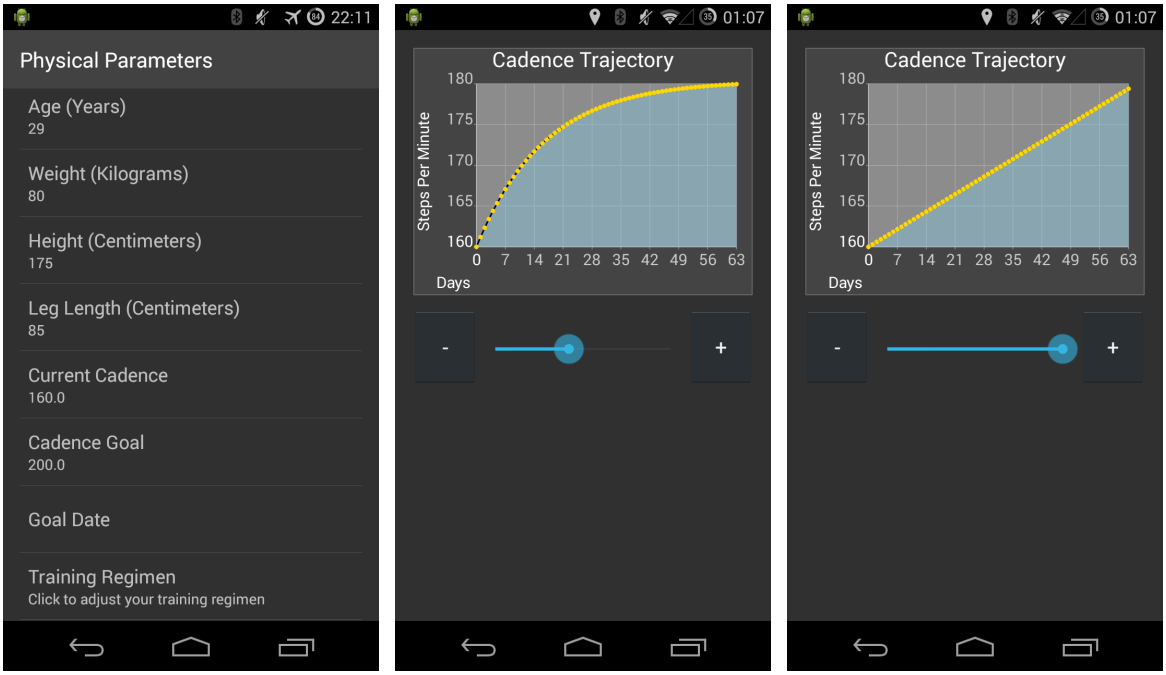
For given values of C, G, and R, the value of α can be modified to adjust the curvature of the exponential model, which has the effect of redistributing the rate of improvement. A larger value of α will result in a performance trajectory with more aggressive early improvement, and a value that approaches zero yields a more linear improvement curve. This can be seen in Figure 1.2. The proof that as $\alpha \to 0$, the curve approaches a line follows (note the use of L'Hôpital's Rule in the first step):

$$\begin{aligned}
\lim_{\alpha \to 0} \frac{Ce^{-\alpha R} - G + (G - C)\, e^{-\alpha t}}{e^{-\alpha R} - 1} &= \lim_{\alpha \to 0} \frac{-RCe^{-\alpha R} - t\,(G - C)\, e^{-\alpha t}}{-Re^{-\alpha R}} \\
&= C + \lim_{\alpha \to 0} \frac{t\,(G - C)\, e^{-\alpha t}}{Re^{-\alpha R}} \\
&= C + \frac{G - C}{R} t
\end{aligned}$$

Since C, G, and R are constants, the result is linear with respect to time t. This is significant in that it guided the implementation of the model in the application.

1.2.3 Implementing the Improvement Model in the Application

The aforementioned improvement model was implemented in the application as an interactive plot of the cadence trajectory found in the Runner Profile settings, shown in Figure 1.3(a). Since current cadence, cadence goal, and the date by which the goal is to be achieved are required for the model, these parameters must be set before the cadence improvement model can be configured. Once they are configured, however, the runner can then access the screen shown in Figure 1.3(b) and (c). By moving a slider left and right, or tapping the corresponding left and right buttons for finer incremental adjustments, the value of α changes and the plot is updated accordingly. The value of α is stored in Android's Shared Preferences, which allow for persistent storage of simpler data types. Every time the user completes a run with the application, data for that run will be uploaded from the application to the server, including the current model parameters and point along the trajectory on that particular day.



(a) (b) (c)

Figure 1.3: In-app configuration of cadence trajectory. (a) Physical parameters and performance goals input by the runner. (b) A more aggressive cadence training regimen that corresponds to a higher value of α. (c) A linear regimen issued when the slider reaches the maximum value.

1.2.4 Future Work

Model Validation

A study will be performed with long-distance runners who will use our application. Part of the study will involve collecting application accuracy and usability feedback from the subjects. Based on the feedback of these more experienced runners, the proposed model may be further refined or discarded. As previously mentioned, other models will be explored as well.

Training the Improvement Model

Ultimately, the goal will be to recommend a personalized training regimen based on the runner's physical parameters and performance goals. However, an initial set of test subjects will configure their own model in order to generate training data. This data will be gathered during a pilot study. For now, test subjects will configure α according to their preferences, and we will use the combination of their personal parameters, preferred α's, and actual cadence improvement trajectory, to train a model. Actual cadence improvement will be taken into account in the event that an individual exceeds or falls short of their configured improvement trajectory. For instance, a more optimal training regimen may lie in between the subject's actual performance improvement, and that configured in the application. In the future, this trained model can be used to generate recommendations of training trajectories on an individual basis. In other words, a latent model for α as a function of a runner's physical parameters and goals will be developed.

Using the pilot study data, we will first attempt to develop this latent model as a least-squares regression. A kernel function for α will be fitted as a function of the following:

- age
- gender
- height
- weight

- leg length
- current performance
- performance goal
- duration of training regimen

Since gender is a categorical predictor with 3 levels (male, female, other), three dummy variables will be created for that parameter corresponding to male, female, and other. A coefficient value of 1 will be assigned to the variable that is indicated by the user input, and 0 for the remaining variables.

1.3 Background Monitoring and Server Synchronization

As a requirement of the application, data monitoring and synchronization with a remote server must be enabled whenever the user has enabled it for a long-distance run. To accomplish this, and to maintain a separation of concerns between user interface and monitoring functionality, this ongoing monitoring and synchronization functionality was implemented by extending the functionality of relevant telemonitoring framework components, namely `AbstractMonitoringService` and `ServerHandler`. `AbstractMonitoringService` facilitates the implementation of ongoing background monitoring processes, and `ServerHandler` handles the process of sending data to the server in a fault-tolerant manner. The use of these components is discussed below.

1.3.1 Monitoring Service

The monitoring service provides a means for other parts of the application to start or stop data collection without concern about the initialization or state of each extractor or estimator. The telemonitoring framework contains the previously mentioned `AbstractMonitoringService` abstract class for implementing background monitoring processes. It is itself an extension of Android's built-in `Service` class, which is responsible for running

asynchronous background processes. In our application, this abstract class was extended simply as `MonitoringService`, which primarily serves the following functions:

- Initialize the encapsulators for data to be sent to the server, namely those for cadence, speed, GPS, heart rate via Bluetooth device, battery usage, screen light, and energy expenditure;
- Register the listeners to each encapsulator for submitting data to the server handler for server synchronization;
- Initialize the extractors and estimators corresponding to the aforementioned encapsulators used for ongoing background monitoring during runs;
- Provide interface methods to start and stop each extractor and estimator.

One challenge in implementing the monitoring service arose from the asynchronous nature of the monitoring service. Because services run asynchronously, other components of the application could try to start or stop data collection before initialization was completed. Making requests to the monitoring service to start non-initialized components would likely result in an application crash. To prevent this, there had to be a guarantee that the data structures were initialized before being referenced. Additionally, this guarantee would ideally be fulfilled in a manner transparent to components of the application other than the monitoring service in order to maintain a separation of concerns. Ultimately, this was accomplished through judicious use of the guarded block idiom [9]. A sketch of how this might be implemented is shown in Figure 1.4. As an example, the following sequence of events may occur:

1. `doSomethingWithDataStructures()` is called externally from a thread before those data structures have been initialized.
2. The method then calls `checkAndWaitForInitialization()`, where it acquires the `lock`.

```java
private synchronized void initializeDataStructures() {
    // Initialize data structures here.
    // ...

    // After data structures are initialized, notify to release other methods
    // that are currently waiting for initialization.
    isInitialized = true;
    synchronized (lock) {
        lock.notify();
    }
}

private synchronized void checkAndWaitForInitialization() {
    synchronized (lock) {
        while (!isInitialized) {
            // Initialization is not complete.
            try {
                // Wait for notification of lock here.
                lock.wait();
            } catch (InterruptedException e) {
                // Wait was interrupted for some reason.
            }
        }
    }
}

public synchronized void doSomethingWithDataStructures() {
    // This method might be called from an external class with no knowledge
    // of the initialization status of MonitoringService's data structures.
    // Since there is no guarantee that this is called before initialization
    // is complete, the method must wait for initialization to complete before
    // continuing.

    checkAndWaitForInitialization();

    // Now that initialization is complete, do something with data structures.
    // ...
}
```

Figure 1.4: A sketch of an implementation and subsequent use of a guarded block. The method `checkAndWaitForInitialization` serves as the guarded block which prevents any action from being taken until initialization is completed.

3. The checking method sees that the data structures are not initialized, and subsequently invokes `lock.wait()`. The thread is blocked from completing its original invocation and goes to sleep.

4. The thread now releases control and waits for notification before resuming.

5. Meanwhile, `initializeDataStructures()` is finally called internally on a different thread.

6. The initialization method completes, acquires `lock`, and invokes `lock.notify()`.

7. The original thread is now regains control and is able to complete `checkAndWaitForInitialization()`.

8. Now the original invocation of `doSomethingWithDataStructures()` can be completed.

By employing a check for initialization using a guarded block in each method that references any of the objects in `MonitoringService`, initialization is guaranteed before each respective extractor or estimator is referenced.

Overall, this modular design allows for proper separation of concerns. Issues regarding the initialization and maintenance of the elements necessary for ongoing background monitoring are abstracted away through the monitoring service. Other parts of the application simply need to make requests to start and stop their desired extractors and estimators.

Another important consideration was the fact that the Android operating system could kill the service process under circumstances of high memory or CPU usage [10]. Because the monitoring service contained the references to the data structures, extractors, and estimators, killing the service process would result in the termination of any currently running data collection. Because of this, flags indicating the state of each extractor and estimator were created that would persist between killing and restarting the service. Thus, whenever the monitoring service is started or restarted, these flags are checked to determine if any data collection should be restarted as well. Testing of this implementation and results are discussed in the following sections.

Testing of Monitoring Service: Methodology

Because of the configuration of `AbstractMonitoringService` as a foreground service, it is very unlikely to be killed normally by the Android OS [10]. Thus, to test the robustness of the monitoring service's resumptive capabilities, the behavior of the Android OS in killing and resuming processes had to be emulated or forcefully induced. While very little documentation exists regarding this practice, Android developer community consensus indicated that stopping a process via the Dalvik Debug Monitoring Server (DDMS) [11] after sending the application to the background would induce the desired behavior [12]. Using this method, the following protocol was executed on a newly installed instance of the application running on a Samsung Galaxy S7 Edge [13] while the phone was plugged into a computer with DDMS

active:

1. Enter the required runner parameters.
2. Disable collection of speed and energy expenditure. This is to provide an indication that, upon resuming, the monitoring service does not indiscriminately resume all data collection.
3. Start a run and proceed past the heart rate estimation steps.
4. Push the home key to send the application into the background.
5. With the application process highlighted in DDMS, press the "Stop Process" button in DDMS.
6. Wait for the Android OS to resume the service.
7. Once the service is resumed (as indicated by a message in DDMS, or the service icon in the Android notification drawer), open the application.
8. Verify that the same extractors and estimators are running through debugging messages and UI activity, i.e. all extractors and estimators except for speed and energy expenditure should be running.
9. Repeat steps 4-8 nine more times during the same execution of the application to ensure stability.

During this test, the native Android stopwatch application was started on a second phone simultaneously with the run. The timer in the Android run activity could be compared to this stopwatch each time the application was killed and restarted to verify that the elapsed time was correct. Because Android would take up to several minutes to restart the service, any discrepancy would be obvious.

A similar test was performed with all data collection disabled, and then with all data collection enabled, with one iteration each. This was to ensure that the monitoring service would always restart or keep disabled the appropriate extractors and estimators, and that any results from the described protocol were not purely coincidence.

Testing of Monitoring Service: Results

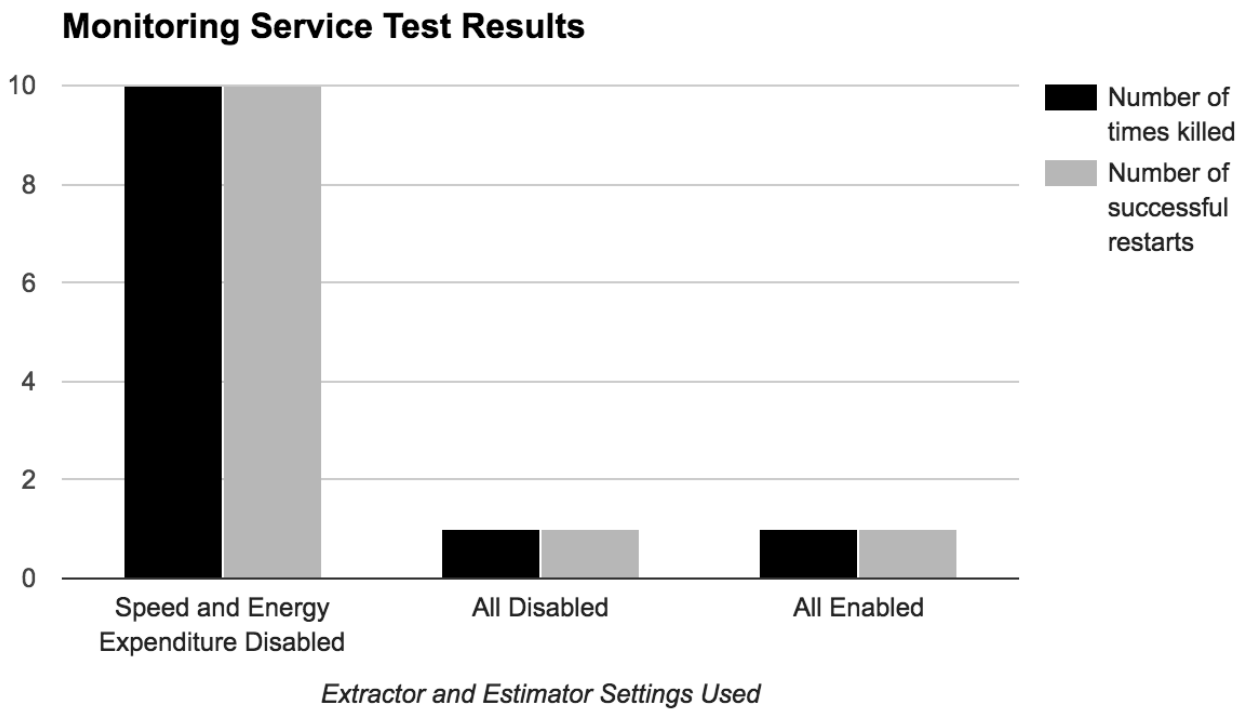


Figure 1.5: Results of testing how many times the monitoring service successfully restarted after being killed.

The results of this test are shown in Figure 1.5. In all cases, the monitoring service was restarted successfully with the correct extractors and estimators. Additionally, the elapsed time in the application was found to be accurate across all instances of being killed and restarted.

As previously mentioned, the configuration of `AbstractMonitoringService` as a foreground service makes it unlikely that it is killed normally by the Android OS. Being killed and successfully restarted ten times in succession indicates that it is likely that the monitoring service would remain stable during the course of a person's training or marathon run.

1.3.2 Server Handler

The server handler is a component of the telemonitoring framework which provides fault-tolerant handling of data being sent to a remote server. It created the connection to the

server and manages the data uploading process. By design, each client-server connection requires one, and only one, server handler. A reasonable design pattern might have involved initialization of the server handler by the monitoring service. However, due to the aforementioned proclivity of the Android OS for killing processes, it was possible that after the monitoring service was killed, upon its subsequent restart, it would initialize a new server handler before the old one was garbage collected. In this event, the new server handler would fail to connect to the server, as the old connection was still active.

To prevent this, the singleton design pattern [14] was employed for the server handler to create a `SingletonServerHandler` class. Since only one instance of a singleton could exist, this guaranteed that no more than one server handler would exist at a given moment.

1.4 Continuing Work and Conclusions

Groundwork has been laid for data analysis and training models. Careful design and implementation practices will have hopefully resulted in a robust application. The next steps will involve conducting the pilot study with long-distance runners in order to evaluate the usability of the application, the accuracy of its data collection, and the effectiveness of the training model.

In this study, long-distance runners will test our application while performing long-distance runs. They will be able to configure what data will be collected, and will offer feedback about data accuracy and application usability. Data and personal feedback collected from this study will bu used to guide improvements to estimators and interventions, as well as further development of training regimens, such as through teaching a model for automated regimen generation, enhancement of the exponential model proposed in this paper, and development of different models entirely. Finally, to carry out this study, a study protocol has been submitted to, review by, and approved by the UC Berkeley Institutional Review Board.

As explained in the introduction, the field of long-distance running, with its focus on

performance and tendency toward injury, serves as a suitable area to concentrate expansion and use the telemonitoring framework. Nevertheless, the scope of the framework extends beyond long-distance running training, and even beyond the areas of fitness and athletics. The framework has implications for health and medicine at large, as was already demonstrated in a pilot study involving telemonitoring of chronic heart failure patients [15]. A fully mature telemonitoring framework has tremendous beneficial potential to reduce health care costs, and to improve health and quality of care. The work discussed in this paper and the contributions of my team have hopefully served as a meaningful step toward achieving these goals.

Chapter 2

Engineering Leadership

2.1 Industry and Market Analysis Overview

While there exist several smartphone based platforms that can be used to create applications for various purposes, such as Canvas, Appery.io and Mobile Rodie [16] most of these offer limited functionality and access to sensors. Additionally, these products lack the ability to easily build predictive models for automated generation of personalized interventions. More generally, there are no such platforms that cater to the issue of telehealth. Our telemonitoring framework, targeted towards doctors and coaches, addresses this unmet need.

To guide the expansion of the framework, we will consider the design and implementation of new features in the context of a commercial application that would be used by marathon trainees. To this end, it is useful to perform a market and industry analysis on existing fitness tracking technology. By examining consumer behavior and industry offerings, we can better understand what functionality is missing and what features athletes desire.

2.2 Market Analysis

According to surveys [17] in 2014 there were more than 1200 marathon events held within the US, with a total of 550,637 finishers. These are both all-time high statistics, with the number of marathon finishers growing about 1.8% from 2013 to 2014. A survey of marathon

runners showed that 74% of them relied on wearable technology for training and 88% of them relied on said technology for motivation [18]. Between 2014 and 2015, the number of wearables purchased is said to have nearly tripled from 17.6 million to 51.2 million [19]. Of Internet users who exercise between the ages of 18-34, 38% of males and 21% of females use wearable fitness trackers [20]. Wearable technology has clearly entered into the mainstream, especially in the area of fitness training with fitness trackers. Marathon runners are no exception. With their ubiquity and proclivity for training technology, they represent an acceptable target market for our application.

2.3 Porter's Five Forces Analysis

We will now conduct a Porter's Five Forces analysis of our mobile application for marathon runners to contextualize it in the industry and develop a strategy for differentiating and promoting it [21].

2.3.1 Bargaining Power of Buyers

Buyers have strong bargaining power only when they are consolidated. Consumers of fitness tracking products are numerous, but diffuse in their buying patterns. Buyers are many and demand is great, weakening the bargaining power of buyers.

2.3.2 Bargaining Power of Suppliers

The power of suppliers refers to the power of businesses that provide materials to another business to demand increased prices for materials [21]. The application is developed for the Android platform, and cell phones have become a commodity, indicating a weak bargaining power.

2.3.3 Threat of New Entrants

New entrants have the potential to bring new ideas to a market. The market of activity monitors poses few barriers and connected fitness trackers are projected to grow from $2 billion to $5.4 billion from 2014 to 2019 [22]. With the burgeoning of the Internet of Things, it is expected that there will be new players in many applications of telemonitoring. Thus, the threat of new entrants is perceived to be strong.

2.3.4 Threat of Substitutes

A product from a different industry that offers the same benefits to consumers is referred to as a substitute product. Substitute products pose a threat because there is possibility of replacement for a company's product [21]. One substitute product for runners training for marathons is meeting one-on-one or in small groups with dedicated professional trainers and coaches. There is an approximately $1.5 billion industry existing in intensive personal athletic training in the United States [23]. This includes firms and independent individuals who provide services granting personalized attention to athletes training for sports seasons or upcoming events such as marathons. However, human trainers conducting in-person training generate problems not seen in the activity monitor/trainer application. For example, scheduling is a factor for this substitute, as the athlete would need to train according to the trainer's schedule and location. Having a human trainer is also significantly more expensive than using an activity monitor. The application does not come with these added cost and conditions. For these reasons we believe that the threat of substitutes is weak.

2.3.5 Rivalry Amongst Existing Competitors

Rivalry can pose a great threat when the rivals compete over price instead of features. The market for tracking and training of fitness, including endurance running, is a crowded one. In this market, our application will be competing with a variety of technologies, such as smartphone apps and specialized fitness tracking hardware. We will need to ensure that our feature offerings are differentiated in order to avoid significant threat from price-based

rivalry.

Wearable fitness tracking devices have seen widespread adoption among runners and other athletes. There are several subcategories of device functionality in this area, ranging in metrics measured, accuracy of these metrics, and price. These include step counters such as the Fitbit One or Nike+ FuelBand at the lower end of functionality and price, GPS-based speed and distance monitors like the Garmin ForeRunner 620 or TomTom Runner at the higher end, and multi-functional devices like smartwatches, such as the Apple Watch or Pebble Time that have some built-in fitness features [24].

Other competing fitness devices include specialized peripheral hardware, such as chest straps to monitor heart rate, shoe inserts to track impact and step duration, and devices that help athletes recover from training in terms of bloodflow and muscle relaxation, such as the Firefly Recovery System [25]. These products are more targeted at health monitoring and feedback for runners, which we can compete with by providing without specialized hardware outside of the mobile phone itself.

Additionally, given the demand for personal training, new products which provide personalized feedback, such as the Moov, have already begun to appear. Moov's successful crowdfunding campaign indicates a demand for fitness trackers that can provide this type of feedback [26]. Major players are pushing for greater personalization. For example, FitBit, a key player in wearable fitness tracking, acquired the startup FitStar in 2015 [27] which provides users with personalized instructional videos. Finally, our application will be competing with a host of other smartphone fitness applications. A huge market for personal fitness tracking exists in the app stores of the smartphones that so many Americans already carry with them daily. A study [28] estimated that in 2012 there were over 40 thousand health and fitness applications available for mobile phones, reaching over 500,000 active users, and that number has only increased in the past few years. A wide variety of fitness and run tracking, goal-setting and socially competitive, and motivational applications are available. Some of the most popular apps specifically targeted at runners are RunKeeper, MapMyRun, and Runtastic [24]. On the more creative side are apps like Zombies, Run

which provides audio motivation in a narrative form, taking a runner through customizable missions in a fictional environment.

Given the great number of players in this industry, the threat of existing rivals is strong. However, given the still largely unexplored area of personalized coaching within the crowded space of fitness tracking technology, we believe that this rivalry will primarily be features-based.

2.4 Technology Strategy

Considering our market research and Porter's Five Forces analysis, we have developed a strategy for our product in order to minimize the threats posed to our product. Our strategy revolves around marketing to customers based on the features offered by our product, particularly focusing on measurement and real-time feedback regarding performance metrics, such as speed and cadence. For instance, despite its importance, many fitness tracking solutions do not measure cadence. In addition, the products that do are typically not transparent about the estimation algorithms used and their accuracies. Even for those that do report accuracy, the algorithms used are still unpublished, and the accuracy of specific metrics, such as cadence, are conspicuously missing [29]. Our application uses algorithms backed by published scientific literature, and the accuracy of our implementation will be further measured and published. Furthermore, the framework on which the application is built includes a fault-tolerant client-server protocol for secure and convenient data syncing, and a wide library of well-tested data collection and analytics functionality to support our application's features and ensure they remain reliable and easy to use. Raising the standards of information transparency, estimation accuracy, and application reliability would not only allow our application to gain traction in the market if we were to actively promote it, but would also impose barriers to new entrants.

Bibliography

[1] D. Aranki, G. Kurillo, A. Mani, P. Azar, J. van Gaalen, Q. Peng, P. Nigam, M. P. Reddy, S. Sankavaram, Q. Wu, and R. Bajcsy, "A telemonitoring framework for android devices," in *Proceedings of the 1st IEEE Conference on Connected Health: Applications, Systems and Engineering Technologies*, to appear, IEEE, Jun. 2016.

[2] (2016). The berkeley telemonitoring project - privacy-aware health telemonitoring, [Online]. Available: `https://telemonitoring.berkeley.edu/`.

[3] R. N. van Gent, D. Siem, M. van Middelkoop, A. G. van Os, S. M. A. Bierma-Zeinstra, and B. W. Koes, "Incidence and determinants of lower extremity running injuries in long distance runners: A systematic review," *British Journal of Sports Medicine*, vol. 41, no. 8, pp. 469–480, 2007. DOI: `doi:10.1136/bjsm.2006.033548`. [Online]. Available: `http://dx.doi.org/10.1136/bjsm.2006.033548`.

[4] B. C. Heiderscheit, E. S. Chumanov, M. P. Michalski, C. M. Wille, and M. B. Ryan, "Effects of step rate manipulation on joint mechanics during running," *Medicine & Science in Sports & Exercise*, vol. 43, no. 2, pp. 296–302, 2011. DOI: `10.1249/MSS.0b013e3181ebedf4`. [Online]. Available: `http://dx.doi.org/10.1249/MSS.0b013e3181ebedf4`.

[5] A. V. Rowlands, R. G. Eston, and C. Tilzey, "Effect of stride length manipulation on symptoms of exercise-induced muscle damage and the repeated bout effect," *Journal of Sports Sciences*, vol. 19, no. 5, pp. 333–340, 2001. DOI: `10.1080/02640410152006108`. [Online]. Available: `http://dx.doi.org/10.1080/02640410152006108`.

[6] J. Hamill, T. Derrick, and K. Holt, "Shock attenuation and stride frequency during running," *Human Movement Science*, vol. 14, no. 1, pp. 45–60, 1995. DOI: `10.1016/0167-9457(95)00004-C`. [Online]. Available: `http://dx.doi.org/10.1016/0167-9457(95)00004-C`.

[7] The Apache Software Foundation. (2016). Commons math: The apache commons mathematics library, [Online]. Available: `https://commons.apache.org/proper/commons-math/`.

[8] T. A. Astorino, M. M. Schubert, E. Palumbo, D. Stirling, D. W. McMillan, C. Cooper, J. Godinez, D. Martinez, and R. Gallant, "Magnitude and time course of changes in maximal oxygen uptake in response to distinct regimens of chronic interval training in sedentary women," *European Journal of Applied Physiology*, vol. 113, no. 9, pp. 2361–2396, 2013. DOI: 10.1007/s00421-013-2672-1. [Online]. Available: http://dx.doi.org/10.1007/s00421-013-2672-1.

[9] Oracle. (2016). The java tutorials: Guarded blocks, [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html.

[10] Google. (2016). Android API guides: Services, [Online]. Available: http://developer.android.com/guide/components/services.html.

[11] ——, (2016). Android developers: Using ddms, [Online]. Available: http://developer.android.com/tools/debugging/ddms.html.

[12] Mark. (2013). Stack overflow: How to simulate android killing my process, [Online]. Available: http://stackoverflow.com/a/18695974/4577206.

[13] Samsung. (2016). Galaxy s7 and galaxy s7 edge, [Online]. Available: http://www.samsung.com/us/explore/galaxy-s7-features-and-specs/.

[14] P. D. Kombate. (2015). Effective ways to implement and use the singleton design pattern, [Online]. Available: https://community.oracle.com/docs/DOC-918906.

[15] D. Aranki, G. Kurillo, P. Yan, D. M. Liebovitz, and R. Bajcsy, "Continuous, real-time, tele-monitoring of patients with chronic heart-failure: Lessons learned from a pilot study," in *Proceedings of the 9th International Conference on Body Area Networks*, ser. BodyNets '14, London, United Kingdom: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 135–141, ISBN: 978-1-63190-047-1. DOI: 10.4108/icst.bodynets.2014.257036. [Online]. Available: http://dx.doi.org/10.4108/icst.bodynets.2014.257036.

[16] G. Smith. (2013). 10 excellent platforms for building mobile apps, [Online]. Available: http://mashable.com/2013/12/03/build-mobile-apps/#SXF0URANsqqg.

[17] Running USA. (2015). 2014 running usa annual marathon report, [Online]. Available: http://www.runningusa.org/marathon-report-2015.

[18] Freescale. (2014). The next evolution in running, [Online]. Available: https://web.archive.org/web/20141223192158/http://blogs.freescale.com/iot/2014/12/wearables-next-evolution-in-running-marathon/.

[19] Gfk. (2015). Gfk forecasts 51 million wearables will be bought globally in 2015, [Online]. Available: gfk.com/news-and-events/press-room/press-releases/pages/gfk-forecasts-51-million-wearables-sold-globally-2015.aspx.

[20] Mintel, "Exercise trends - us - october 2014," Mintel, Market report, 2014.

[21] M. E. Porter, "The five competitive forces that shape strategy," *Harvard Business Review*, vol. 86, no. 1, pp. 78–93, 2008.

[22] Parks Associates. (2015). Global revenues from connected fitness trackers to exceed $5 billion by 2019, [Online]. Available: http://www.parksassociates.com/blog/article/pr-march2015-whcc.

[23] D. Witter, "Ibisworld industry report 61162: Sports coaching in the us," IBISworld, Industry report, 2015.

[24] J. Carter. (2013). 10 best running gadgets: The top tech for training, [Online]. Available: http://www.techradar.com/us/news/world-of-tech/roundup/10-best-running-gadgets-the-top-tech-for-training-1157180/1.

[25] K. Alger. (2014). Training for a marathon: Tech for half marathon, marathons and beyond, [Online]. Available: http://www.t3.com/features/training-for-a-marathon-half-marathon-ultra-and-beyond.

[26] J. Colao. (2014). Who needs kickstarter? exercise sensor moov raises $1 million in 15 days, [Online]. Available: http://www.forbes.com/sites/jjcolao/2014/03/14/exercise-sensor-moov-raises-1-million-in-15-days-without-kickstarter/#704c414614e3.

[27] R. Lawler. (2015). Fitbit confirms fitstar acquisition to bring training to its fitness portfolio, [Online]. Available: http://techcrunch.com/2015/03/05/fitbit-confirms-fitstar-acquisition-to-bring-training-to-its-fitness-portfolio/.

[28] Stanfy. (2013). Fitness in mobile: Case-study, [Online]. Available: http://www.slideshare.net/stanfymobile/fitness-cs-22962137.

[29] Garmin. (2016). How accurate are the running dynamics from the hrm-run and how was the accuracy tested? [Online]. Available: https://support.garmin.com/support/searchSupport/case.faces?caseId=%7B435cc8c0-e2e1-11e3-47b1-000000000000%7D.