Mobile Browser Fingerprinting

Viyat Bhalodia, Vibha Iyengar, Shardul Mahadik

September 23, 2017

Abstract

Browser fingerprinting has been primarily performed by fingerprinting the software stack such as the browser info, plug-ins, fonts and, to some extent, using the hardware specification information provided by the browser. However, studies have shown that their reliability is fairly low when fingerprinting mobile devices. Recently, techniques have emerged which fingerprint the mobile device hardware such as accelerometer, gyroscope, microphone and magnetometer. We present two hardware fingerprinting techniques and one software fingerprinting technique which can be used to augment the existing fingerprinting libraries to improve reliability of mobile fingerprinting and even differentiate between devices with the same software stack.

1 Introduction

Browser fingerprinting is a method of tracking web browsers by the configuration and settings information they make visible to websites, rather than traditional tracking methods such as IP addresses and unique cookies. Existing fingerprinting techniques can anonymously identify a web browser with accuracy of up to 94%[1]- this number is not so reliable on mobile browsers because of sand-boxed browsers and less measurable parameters. There are not enough variables to reliably fingerprint a mobile device using just the software stack but we can make use of the variability of hardware due to a variety of sensors and also the manufacturing errors that come with them coupled with the JavaScript runtime to make a reliable estimate of the hardware stack.

We present three techniques to increase the reliability of fingerprinting on mobile browsers. We investigated the use of cipher suites supported by the client browser to improve the reliability of other fingerprinting techniques and mitigate some of the anti-fingerprinting techniques like spoofing of the User-Agent header, display resolution, etc. The other technique that we investigated was touch event support. Every mobile device supports different touch based events like gestures and can have a variable number of maximum touch points. This data can be used to further increase the entropy of the fingerprint hash.

Hardware sensors are exposed through HTML5 APIs to JavaScript which make it possible to reliably collect sampled sensor data. Many of these APIs do not event required user granted permissions to be able to access sensor data. Even between he same make and model of a sensor, the sensor characteristics are not perfectly equal owning to the imperfections introduced during the manufacturing process of these sensors. These imperfections can be used to uniquely identify mobile devices apart. A preliminary investigation with the data reveals that the data stays almost constant for a specific device if collected in lab setting.[2] In this project, we specifically aim to investigate the gyroscope sensor which allows us to detect the orientation of the mobile device as a means of fingerprinting. With a limited amount of user interaction, we can detect the errors present in the sensor along its various axes. Multiple runs of the fingerprinting algorithm on the same device should yield similar error values.

In section II we go over some recent work in the field of hardware sensor fingerprinting. In section III we describe our approach to the three different fingerprinting techniques we investigated. Section IV provides a brief description of the results of our experiments with the above mentioned techniques. Section V discusses some of the limitations of our techniques and the major roadblocks that we faced along the way. Finally, we conclude our finding in section VI. We also discuss some potential solutions to the limitations and methods to improve accuracy of our techniques in VIII.

2 Related Work

There are a few recent works which propose methods to fingerprint accelerometers, gyroscopes and other sensors. Bojinov et al. in their paper [2], presented a new approach to mobile device identification which allows for devices to be recognized without relying on soft identifiers (which may be lost after a device reset). The discussed fingerprinting method exploits sensor calibration variations in the speaker-microphone system and in the accelerometer. This method of identification is interesting because it can be performed by untrusted web code running within a mobile browser. The authors used available JavaScript APIs for sensors to get sampled data from the best devices.

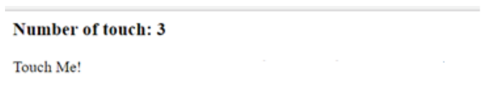
Anupam datta et al., in their study, fingerprint smart devices through on-board acoustic components like microphones and speakers[3]. They collected fingerprints from 52 different smartphones covering a total of five different brands of smartphones. The authors tested the devices in both controlled environments and environments with ambient noise. Their results indicate that the devices can be successfully fingerprinted using acoustic sound waves. In [4], the authors demonstrate techniques to fingerprint the accelerometer through a stimulus generated by a vibration motor. They fingerprint 80 different accelerometer chips and 25 Android phones to find a high reliability in the lab setting only to notice a significant drop in a public setting.

3 Methodology

3.1 Touch events

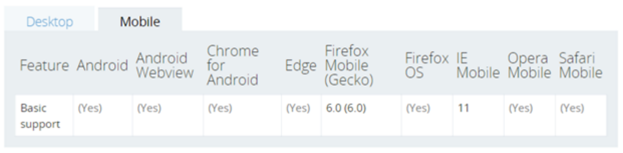
An evident feature of a mobile device is its touch screen. In our study, we exploited this feature, to examine if it can be used as a parameter for device fingerprinting. We used JavaScript Web API to access the event values from the touch sensors. The API used helped in detection of simultaneous touch events, indicating if the touch is enabled or not, and if it is enabled, how many simultaneous touch events is supported by the device.

As part of the experiment, we created a HTML page with a defined section for `touch`, and an embedded JavaScript. This script contained an event listener to capture any touch event within the defined section of the page, and trigger a counter. The counter value displayed onto the HTML page indicated the simultaneous touch events triggered at that moment. A major advantage of using `TouchEvent` class from the API is that, as shown in Figure 1, it is compatible with all the mobile browsers. [5]



Number of touch: 3

Touch Me!


Desktop
Mobile

Feature	Android	Android Webview	Chrome for Android	Edge	Firefox Mobile (Gecko)	Firefox OS	IE Mobile	Opera Mobile	Safari Mobile
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	6.0 (6.0)	(Yes)	11	(Yes)	(Yes)

Figure 1: Touch Event Output and Browser Compatibility Information

3.2 Cipher suite support

A cipher suite, in SSL, is a collection of cryptographic techniques that defines a secure communication channel. There are hundreds of cipher suites, and they are all built out of a dozen or so basic building blocks: key exchange, encryption and integrity validation algorithms. Different software programs often support a variety of cipher suites so that they can be compatible with many different services that they consume over the network. By observing the list of supported cipher suites one can guess the make of the SSL client on the other side.

It is possible to the list of support TLS ciphers by looking at the `ClientHello` messages in the TLS/SSL data stream. The `ClientHello` message is defined in the RFC[6] as,

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
    Extension extensions;
} ClientHello;
```

The field `CipherSuite` includes the the list of supported SSL ciphers with the server. The server will select the best cipher it supports from that list. Browsers generally support a wide variety of SSL/TLS ciphers to ensure maximum compatibility with all servers on the Internet. We can use the number and types of cipher suites supported to increase the reliability of fingerprinting even in case of spoofed browser version or user agent.

This concept can be applied to both desktop and mobile browsers. One disadvantage of this fingerprinting technique is that it cannot be implemented completely on client side i.e. in JavaScript. For implementation, we used pyTLS[7] which is a TLS implementation in Python. An example output of the querying the cipher suites supported by the browser is shown below:

```
Connection from ('127.0.0.1', 60708)
Record Version: TLS1_0
Handshake Version: TLS1_2
Session ID Length: 0
Cipher Suites Length (bytes): 26
Cipher Suites:
0xc02b TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
0xc02f TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xcca9 TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
0xc02c TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
0xc030 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xc00a TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
0xc009 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
0xc013 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
0xc014 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
0x0033 TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0x0039 TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0x002f TLS_RSA_WITH_AES_128_CBC_SHA
0x0035 TLS_RSA_WITH_AES_256_CBC_SHA
```

Although this is a versatile tool, it needs to be refined in terms of usability, particularly because it does not return a response until it has iterated all possible cipher lists. This makes the client browser "hang" while the server tests for all possible cipher suites by initiating handshakes using every cipher in its list. Another disadvantage of this tool (and possibly this method), is that repeat connections from the same browser will often show different sets of cipher suites, as the browser may randomly throw in different GREASE ciphers at various locations in the requested cipher list.[8] A complete list of TLS/SSL cipher suites available and their RFCs are provided.[9]

3.3 Gyroscope Fingerprinting

3.3.1 Overview

A gyroscope is a sensor which provides the orientation of a device at any given moment of time. The orientation is described by a combination of three values viz. α, β and γ. α is the rotation of the devices along the z-axis. Similarly, β and γ describe rotation along the x-axis and y-axis respectively. We capture and observe these readings over a period of time to fingerprint the mobile device. However, unlike the accelerometer which has a fixed baseline reading of g along the z-axis when at rest on a flat surface, the gyroscope has no such baseline. Thus, we cannot measure the errors in the sensor using just one reading. Hence, in our approach we rely on two readings in two different orientations. Based on the readings, we calculate two type of errors:

1. Sensitivity: The drift in the reading along an axis after every rotation

2. Offset: The absolute drift in the reading along an axis

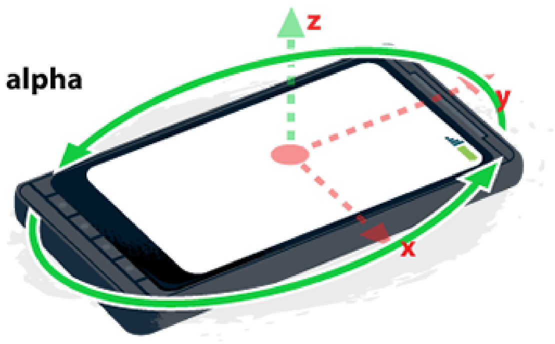


Figure 2: α-axis in a mobile device gyroscope

The first reading is regarded as the baseline reading and using the second reading we calculated the deviation from the expected readings. To be able to deterministically calculate deviation, we use the following two orientations.

1. The device lying flat on a surface with its side being aligned to an edge

2. The device now rotated 180° being aligned to the same edge

This allows us to calculate the sensitivity of the gyroscope sensor along the α-axis (S_α) and the offsets of the sensor along the β (O_β) and γ (O_γ) axes. If the readings in the first orientation are α_0, β_0 and γ_0, then readings in the second orientation should be as follows:

- α_{180} should be 180° away from α_0
- β_{180} should be equal to $-\beta_0$
- γ_{180} should be equal to $-\gamma_0$

Using these, we can calculate the sensitivity and offsets as follows:

$$S_\alpha = \frac{\alpha_0 - \alpha_{180}}{180} \tag{1}$$

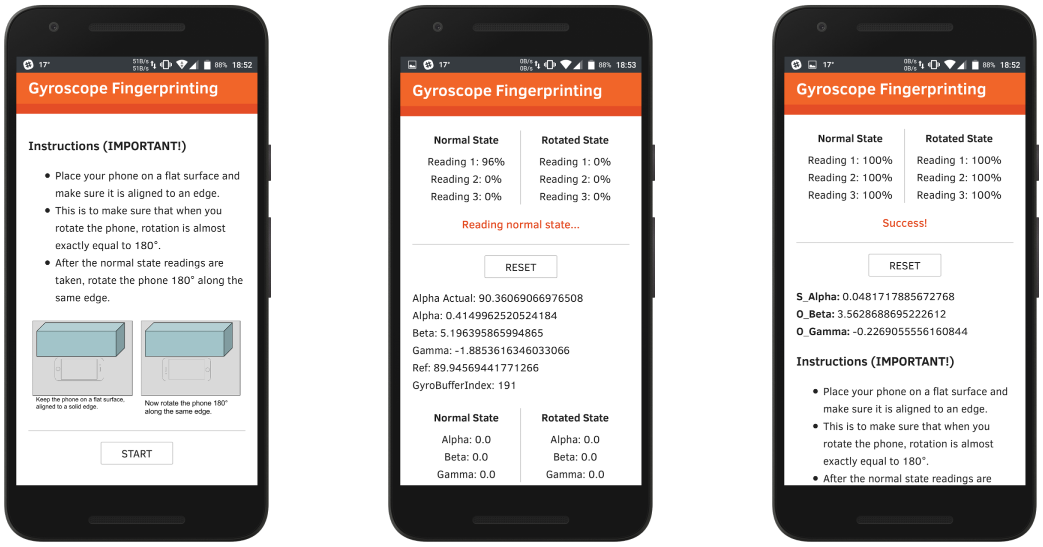
$$O_\beta = \frac{\beta_0 + \beta_{180}}{2} \tag{2}$$

$$O_\gamma = \frac{\gamma_0 + \gamma_{180}}{2} \tag{3}$$

Since these errors are a virtue of the device hardware, the errors will remain within the margin of error for all readings on the same device.

3.3.2 Implementation

We wrote a server in Node.js for data collection. The server used MySQL as a back-end database for storing the data. In each orientation, we take 3 readings. Each reading consists of an average of values from 200 gyroscope events. The gyroscope events are captured using the `deviceorientation` event in Javascript. These readings are then used to calculate S_α, O_β and O_γ. Once we obtain these values, we log them in the database along with a unique identifier per device-browser pair via the means of a browser cookie. This cookie helps us identify readings taken from the same device and verify if they are similar. This also helps us verify that there does not exist a high correlation between readings taken from different devices. We also store the User-Agent of the browser in order to use user-agent strings complementary to our gyroscope fingerprint readings.


Gyroscope Fingerprinting
Keep the phone on a flat surface, aligned to a solid edge.
Now rotate the phone 180° along the same edge.
START
Gyroscope Fingerprinting
RESET
Gyroscope Fingerprinting
RESET

Instructions (IMPORTANT!)

- Place your phone on a flat surface and make sure it is aligned to an edge.
- This is to make sure that when you rotate the phone, rotation is almost exactly equal to 180°.
- After the normal state readings are taken, rotate the phone 180° along the same edge.

Normal State	Rotated State
Reading 1: 96%	Reading 1: 0%
Reading 2: 0%	Reading 2: 0%
Reading 3: 0%	Reading 3: 0%

Reading normal state...

Alpha Actual: 90.36069066976508
Alpha: 0.4149962520524184
Beta: 5.196395865994865
Gamma: -1.8853616346033066
Ref: 89.94569441771266
GyroBufferIndex: 191

Normal State	Rotated State
Alpha: 0.0	Alpha: 0.0
Beta: 0.0	Beta: 0.0
Gamma: 0.0	Gamma: 0.0

Normal State	Rotated State
Reading 1: 100%	Reading 1: 100%
Reading 2: 100%	Reading 2: 100%
Reading 3: 100%	Reading 3: 100%

Success!

S_Alpha: 0.0481717885672768
O_Beta: 3.5628688695222612
O_Gamma: -0.2269055556160844

Instructions (IMPORTANT!)

- Place your phone on a flat surface and make sure it is aligned to an edge.
- This is to make sure that when you rotate the phone, rotation is almost exactly equal to 180°.
- After the normal state readings are

Figure 3: Data Collection Website

4 Results

4.1 Touch events

Majority of the devices tested were limited to 10 simultaneous touch points. Also, the exact value collected from different devices is significantly dependent on the user behavior, with respect to how a user interacts with the touch screen (Single touch/Multi touch). Hence, including this parameter in the fingerprinting would mean it is no longer a stand alone device fingerprint, but has components derived from the user interactions linked to it.

4.2 Cipher suite support

This method has been particularly difficult to test extensively because it requires an extended connection with the client browser. The fingerprinting method still needs to be refined to be usable for general purposes. One of the Qualys SSLLabs projects,[10] implemented an Apache module to collect SSL client fingerprints. An example of some of the data collected is given below,

```
Mozilla/5.0 (iPhone; U; CPU iPhone OS 2_2 like Mac OS X; en-us)
   AppleWebKit/525.18.1 (KHTML, like Gecko) Mobile/5G77
    Handshake: h3
    Protocol: 03.01

    TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)
    TLS_RSA_WITH_RC4_128_SHA (0x05)
    TLS_RSA_WITH_RC4_128_MD5 (0x04)
    TLS_RSA_WITH_AES_256_CBC_SHA (0x35)
    TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x0a)
    TLS_RSA_WITH_DES_CBC_SHA (0x09)
    TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x03)
    TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x08)
    TLS_RSA_WITH_NULL_MD5 (0x01)
```

```
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
    Handshake: h2
    Protocol: 03.01

    TLS_RSA_WITH_RC4_128_MD5 (0x04)
    TLS_RC4_128_WITH_MD5 (0x010080)
    TLS_RSA_WITH_RC4_128_SHA (0x05)
    TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x0a)
```

More examples have been included with the project files.

4.3 Gyroscope Fingerprinting

Through the website, we were able to gather data from a combination of 22 device-browser pairs. The data was gathered from primarily own our devices and also from devices of friends and fellow students from the Browser Security course. We gathered a total of 74 readings from all the devices combined. Since we gathered three dimensions of data from the readings, we plotted two dimensions at a time on the scatter plots.

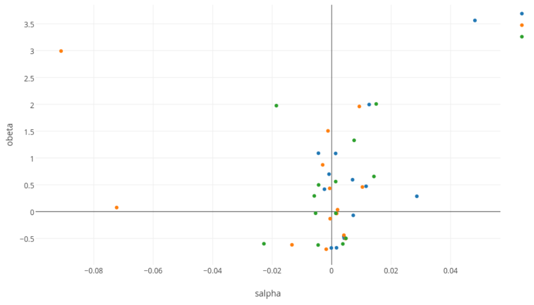


Figure 4: S_α v/s O_β

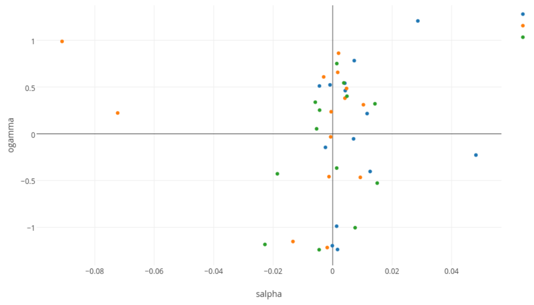


Figure 5: S_α v/s O_γ

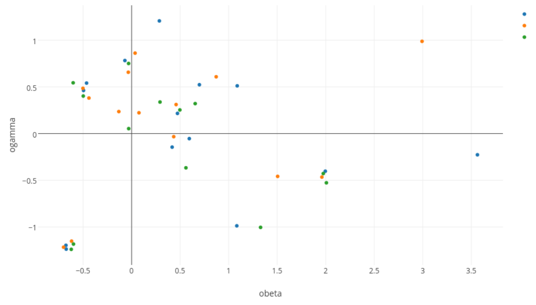


Figure 6: O_β v/s O_γ

We observed that the graphs for S_α v/s O_β and S_α v/s O_γ do not have a distinct clustering of the datapoints like the O_β v/s O_γ. This implies that the S_α dimension may need some amount of scaling to be able to create clusters in the scatter plot. Further, to include all the three dimensions in our calculations, we perform Principal Component Analysis on our data to obtain a scatter plot of the two most dominant eigenvectors. In the figure below, we have plotted the eigenvector for 9 devices with one or multiple browsers for which we had at least three readings. As we can see, the readings from the same device are clustered closely together even if the browsers are different. Also, we can see one devices as an outlier with the plots spread across the whole plot. We suspect this device had a broken gyroscope sensor or the readings were not taken accurately. We were unable to calculate the entropy that the gyroscope fingerprinting added since we did not have sufficient readings.

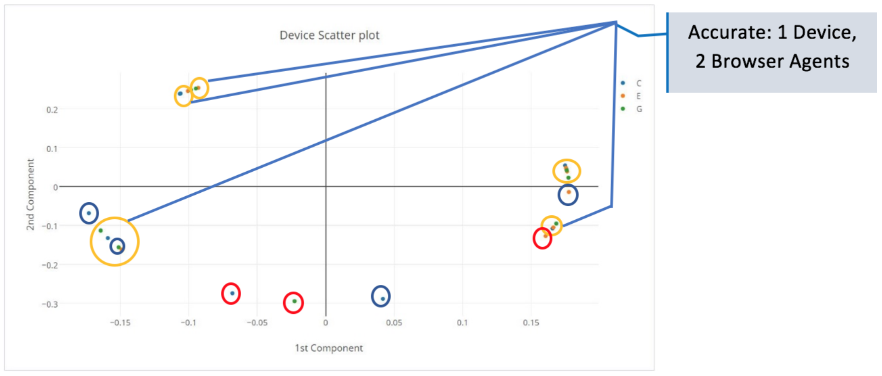


Figure 7: Plot of the dominant eigenvectors after PCA

5 Limitations

A large section of our results was based on the fingerprint data collection from multiple devices and browsers, and hence a major stumbling block was to get participants who were willing to let us collect their mobile device fingerprints, which is considered as a sensitive data. 1/3rd participants we approached were reluctant to take part in this study, and 2/3rd of the participants who did agree, were hesitant to install Firefox or other study-compatible browsers in their device. As a result, our study consisted of less devices, and thus less datasets.

Since the data collection for our study was done by the participants without inspection, we predict that the less accurate results found, might be due to wrong procedure followed by the participants. Another facet of difficulty we faced was collecting fingerprint data from the Webkit based browsers like Google Chrome and Opera, which are widely used in mobile devices. The problem with these browsers is that, it does not trigger events pertaining to gyroscope sensor when still, which tampered our iterative data collection process. To measure the mobile device fingerprinting using browsers, and device capabilities based on touch features, cipher suites or gyroscope sensors, it is imperative and unavoidable to involve user intervention, and this is a challenge that needs to be addressed.

6 Conclusion

Since device fingerprint authentication is a paradigm that future might experience, our study would be a good contribution towards understanding how the touch sensors, gyroscope sensors, and the cipher suite support can be relevant parameters for fingerprinting. We were successful in constructing a study methodology to evaluate the gyroscope sensors in multiple devices. We also think, that it could act as a pilot study for any future work that would be done involving gyroscope sensors. The challenges we faced have been noted to help future researchers with the list of factors that might get into their way.

7 Acknowledgement

We thank Dr. Limin Jia for introducing us to Browser Security, and providing us the opportunity to work on the project based on the Mobile Browser Fingerprinting. Her mentoring played a crucial role in our study methods. We would also like to thank the teaching assistant, Alex James for his help in the completion of this study. The contributions from the open source projects from Github helped us narrow down our study focus, and prepare a module to test specifics with respect to gyroscope sensors, cipher suites supported and touch features. Last, but not the least, we thank all our classmates for their continual support and guidance throughout the study.

8 Future Work

8.1 Improving the accuracy of the readings

Currently we rely on only two orientations of the device to calculate the values of S_α, O_β and O_γ, one placed on a flat surface and another 180° rotated along the z-axis on the same flat surface. However, we can increase its accuracy by recording readings in four orientations instead of two. We rely on the rectangular shape of mobile devices to be able to obtain these orientations easily without the need of a template. These orientations are described as follows:

1. The device lying flat on a surface with its side being aligned to an edge
2. The device now rotated 90° being aligned to the same edge
3. The device now rotated 180° being aligned to the same edge
4. The device now rotated 270° being aligned to the same edge

Now, we can derive the following conclusions:

- α_{90} should be 90° away from α_0. Similarly, α_{180} and α_{270} should be 180° and 270° away from α_0 respectively
- β_{180} should be equal to $-\beta_0$ and β_{270} should be equal to $-\beta_{90}$
- γ_{180} should be equal to $-\gamma_0$ and γ_{270} should be equal to $-\gamma_{90}$

Using these, the new calculations for sensitivity and offsets are:

$$S_\alpha = \frac{\frac{1}{90}(\alpha_0 - \alpha_{90}) + \frac{1}{180}(\alpha_0 - \alpha_{180}) + \frac{1}{270}(\alpha_0 - \alpha_{270})}{3} \tag{4}$$

$$O_\beta = \frac{(\beta_0 + \beta_{180}) + (\beta_{90} + \beta_{270})}{4} \tag{5}$$

$$O_\gamma = \frac{(\gamma_0 + \gamma_{180}) + (\gamma_{90} + \gamma_{270})}{4} \tag{6}$$

8.2 Workaround for Webkit-based browsers

As mentioned above, Webkit-based browsers do not trigger gyroscope events when the device is still. This negatively affects our data collection method since it relies on the device being stationery to calculate the small errors. Hence, we require a way to get gyroscope sensor data even when the device is still. Recently we stumbled upon the WebVR spec[11] being developed by W3C in order to support Virtual Reality devices via the browser. Google has developed a WebVR polyfill[12] to be able to support the WebVR spec without the need of a special browser build. The polyfill uses sensor fusion developed by Boris Smus[13] to be able to fuse the data gathered by the `devicemotion` event which is fired on regular intervals on all browsers with the `deviceorientation` event used to get the gyroscope readings. The `devicemotion` event exposes instantaneous rotation rate of the device and then can be paired with the latest readings obtained from the `deviceorientation` event to approximate the instantaneous gyroscope readings. The polyfill also allows us to specify a sampling frequency rather than depending upon the variety in the sampling frequencies provided by the browsers. This is acheived by predicting future readings when the sampling frequencies are too low.

Device + Browser/API	Sampling Frequency
One Plus One + Android API	200Hz
Samsung Galaxy S3 + Android API	100Hz[14]
Samsung Galaxy S2 + Android API	50Hz
One Plus One + Firefox	200Hz
One Plus One + Chrome	60Hz[15]
iOS + Chrome	60Hz
iOS + Safari	20Hz[16]

Table 1: Sampling Frequencies for various devices and APIs

The sensor fusion API exposes a set of quaternions which are three dimensional vector than provide acceleration and rotational information. The sensor fusion technique also aims increasing the accuracy of the gyroscope by fusing it with accelerometer readings which provide absolute values with respect to the world. The `Complementary Filter` uses the noise-free readings from the gyroscope and combines them with the absolute readings from the accelerometer. We could use the complementary filter in our calculations, but we fear that it might smooth out the gyroscrope readings and adversely affect our potential to calculate errors.

References

[1] Peter Eckersley. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2010.

[2] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416*, 2014.

[3] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. *NDSS âĂŹ16*, February 2016.

[4] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. Accelprint: Imperfections of accelerometers make smartphones trackable. 2014.

[5] Mozilla Developer Network. Pointerevent.pointertype documentation. *http://developer.mozilla.org/en-US/docs/Web/API/PointerEvent/pointerType*.

[6] IETF. The transport layer security (tls) protocol version 1.2. *http://tools.ietf.org/html/rfc5246#page-41*.

[7] Westpoint Ltd. A python tls library for penetration testers. *https://github.com/WestpointLtd/pytls*.

[8] Is there a list of which browser supports which tls cipher suite? - stack overflow. *http://stackoverflow.com/questions/31785430/is-there-a-list-of-which-browser-supports-which-tls-cipher-suite*.

[9] IANA. Transport layer security (tls) parameters. *https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4*.

[10] SSL Labs. Http client fingerprinting using ssl handshake analysis. *https://www.ssllabs.com/projects/client-fingerprinting/*.

[11] W3C. Webvr spec version list. *https://w3c.github.io/webvr/*.

[12] Google. Google webvr polyfill. *https://github.com/googlevr/webvr-polyfill*.

[13] Boris Smus. Sensor fusion and motion prediction. *http://smus.com/sensor-fusion-prediction-webvr*.

[14] Ozlem Durmaz Incel. Analysis of movement, orientation and rotation-based sensing for phone placement recognition. *Sensors*, 15(10):25474–25506, 2015.

[15] Google. Chromium project. *https://chromium.googlesource.com/chromium/src.git/+/150650db7028bb7d738436f73071bc7127918390*.

[16] WebKit Bugzilla. Bug 145814. *https://bugs.webkit.org/show_ bug.cgi?id=145814*.