SKRIPSI

PENDETEKSI GERAKAN KEPALA DENGAN GOOGLE CARDBOARD

EGA PRIANTO

NPM: 2013730047

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2016

UNDERGRADUATE THESIS

HEAD MOTION DETECTOR USING GOOGLE CARDBOARD

EGA PRIANTO

NPM: 2013730047

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2016

LEMBAR PENGESAHAN

PENDETEKSI GERAKAN KEPALA DENGAN GOOGLE CARDBOARD

EGA PRIANTO

NPM: 2013730047

Bandung, 20 «bulan» 2016

Menyetujui,

Pembimbing Utama	Pembimbing Pendamping
Pascal Alfadian, M.Comp.	**«pembimbing pendamping/2»**
Ketua Tim Penguji	**Anggota Tim Penguji**
«penguji 1»	**«penguji 2»**

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENDETEKSI GERAKAN KEPALA DENGAN GOOGLE CARDBOARD

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 20 «bulan» 2016

Meterai

Ega Prianto
NPM: 2013730047

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini...?»

KATA PENGANTAR

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Bandung, «bulan» 2016

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Virtual Reality adalah teknologi yang mampu membuat penggunanya dapat berinteraksi dengan lingkungan buatan oleh komputer, suatu lingkungan yang sebenarnya ditiru atau hanya ada di dalam imajinasi.[1] *Virtual Reality* membuat pengalaman sensorik, di antaranya penglihatan, pendengaran, perabaan, dan penciuman secara buatan.[2] Gawai *Virtual Reality* terbaru sekarang yaitu dengan menggunakan *head-mounted display*, Google Cardboard salah satunya. *Head-mounted display* adalah menempatkan layar di kepala, sehingga pengguna hanya dapat melihat tampilan yang ditampilkan oleh layar.[3]

Google Cardboard[4] adalah gawai murah yang terbuat dari kardus untuk dapat merasakan pengalaman *virtual reality* dengan *smartphone* Android atau iOS. Kita dapat membuat Google Cardboard kita sendiri secara gratis dengan mengunduh templatenya di situs web Google Cardboard. [4]Template tersebut membantu dalam merakit kardus dengan dibentuk, dilipat dan digunting sedemikian rupa sehingga berbentuk kacamata. Bahan-bahan untuk merakit Google Cardboard hanyalah kardus, lem, dan lensa dengan spesifikasi tertentu.

Pada Gawai Google Cardboard cara pengguna memberikan *input* kepada program sangatlah terbatas. Cara tersebut hanyalah dengan gerakan kepala dan tombol magnet. Tombol magnet ini pun terkadang tidak berfungsi dengan baik, karena bergantung pada medan magnet yang di deteksi oleh *smartphone* yang digunakan. Cara lainnya agar dapat memberikan *input* kepada program adalah dengan menghubungkan *smartphone* yang digunakan dengan *bluetooth controller*.

Skripsi ini akan membuat aplikasi untuk menambahkan cara baru memberikan *input* pada Google Cardboard. Pada skripsi ini, akan dibuat dua buah perangkat lunak. Perangkat lunak pertama akan digunakan untuk menganalisis data yang didapat dari sensor-sensor pada Android. Perangkat lunak kedua akan dapat mendeteksi gerakan kepala penggunanya ketika sedang menggeleng atau mengangguk. Pada perangkat lunak kedua ini akan memberikan *input* baru kepada program virtual reality. Jenis *input* yang diberikan kepada komputer hanya ya(mengangguk) atau tidak(menggeleng).

Agar *Virtual Reality* menggunakan Google Cardboard dapat berjalan dengan sempurna, dibutuhkan *smartphone* yang memiliki 3 jenis sensor. Ketiga sensor itu adalah *Magnetometer*, *Accelerometer*, dan *Gyroscope*.[5] Jika salah satu sensor itu tidak ada, tampilan gambar pada *Virtual Reality* akan tidak akurat atau lambat. *Magnetometer* digunakan untuk mengetahui arah pandang pengguna. *Accelerometer* digunakan untuk mengetahui arah gaya gravitasi.[6] *Gyroscope* digunak-

an untuk mengetahui percepatan perputaran sudut kepala pengguna. Ketiga sensor ini juga harus menggunakan sensor 3 sumbu. Ketiga sensor tersebut tidak hanya berfungsi agar dapat menjalankan *Virtual Reality* dengan Google Cardboard dan *smartphone*, tetapi juga dapat berfungsi sebagai pendeteksi gerakan kepala.

1.2 Rumusan Masalah

- Bagaimana cara menampilkan grafik data yang diambil dari sensor-sensor pada *smartphone*?
- Bagaimana cara mendeteksi gerakan kepala dari data yang didapat dari sensor-sensor pada *smartphone*?

1.3 Tujuan

- Mengetahui cara untuk menampilkan grafik data dari sensor-sensor pada *smartphone*.
- Mengetahui cara mendeteksi gerakan kepala dari data yang didapat dari sensor-sensor pada *smartphone*.

1.4 Batasan Masalah

Penelitian ini dibuat berdasarkan batasan-batasan sebagai berikut:

1. Program pertama yang akan dibuat dalam skripsi ini hanya akan digunakan untuk membantu dalam menganalisis sensor.
2. Program kedua yang akan dibuat hanya dapat melakukan pendeteksi gerakan kepala khusus untuk mengangguk dan menggeleng saja.

1.5 Metode Penelitian

Berikut adalah metode penelitian yang digunakan dalam penelitian ini:

1. Melakukan studi literatur tentang Android SDK, Google VR SDK, Quaternion, *Sensor Fusion*, dan algoritma *Head Motion Detection*.
2. Merancang dan membuat aplikasi untuk menampilkan grafik sensor-sensor pada *smartphone* Android.
3. Menganalisis aplikasi-aplikasi sejenis.
4. Merekam dan menganalisis grafik dari sensor-sensor pada *smartphone* ketika mengangguk dan menggeleng.
5. Menganalisis metode pendeteksi gerakan kepala.
6. Merancang aplikasi untuk mendeteksi gerakan kepala
7. Mengimplementasi algoritma pendeteksi gerakan kepala ke aplikasi *virtual reality*.

1.6 Sistematika Penulisan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan, yaitu membahas mengenai gambaran umum penelitian ini. Berisi tenang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

2. Bab 2: Dasar Teori, yaitu membahas teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang Android SDK, Google VR SDK, Quaternion, dan algoritma *Head Motion Detection*.

3. Bab 3: Analisis, yaitu membahas mengenai analisa masalah. Berisi tentang analisis aplikasi-aplikasi sejenis, analisis grafik dari sensor-sensor pada *smartphone* ketika mengangguk dan menggeleng, analisis metode pendeteksi gerakan kepala.

4. Bab 4: Perancangan yaitu membahas mengenai perancangan

BAB 2

DASAR TEORI

Pada bab ini akan dijelaskan dasar-dasar teori mengenai Android SDK, Google VR SDK,Kuaternion, *Sensor Fusion*, dan algoritma *Head Motion Detection*.

2.1 Android SDK

Android SDK(*software development kit*) adalah kumpulan *source code, development tools, emulator*,[7] dan semua *libraries* untuk membuat suatu aplikasi untuk *platform* Android. IDE (*integrated development environment* yang resmi untuk Android SDK adalah Android Studio. Android Studio dapat di download di halaman situs web Google Developer[7], sekaligus dengan Android SDKnya.

2.1.1 Struktur File Android Studio Project

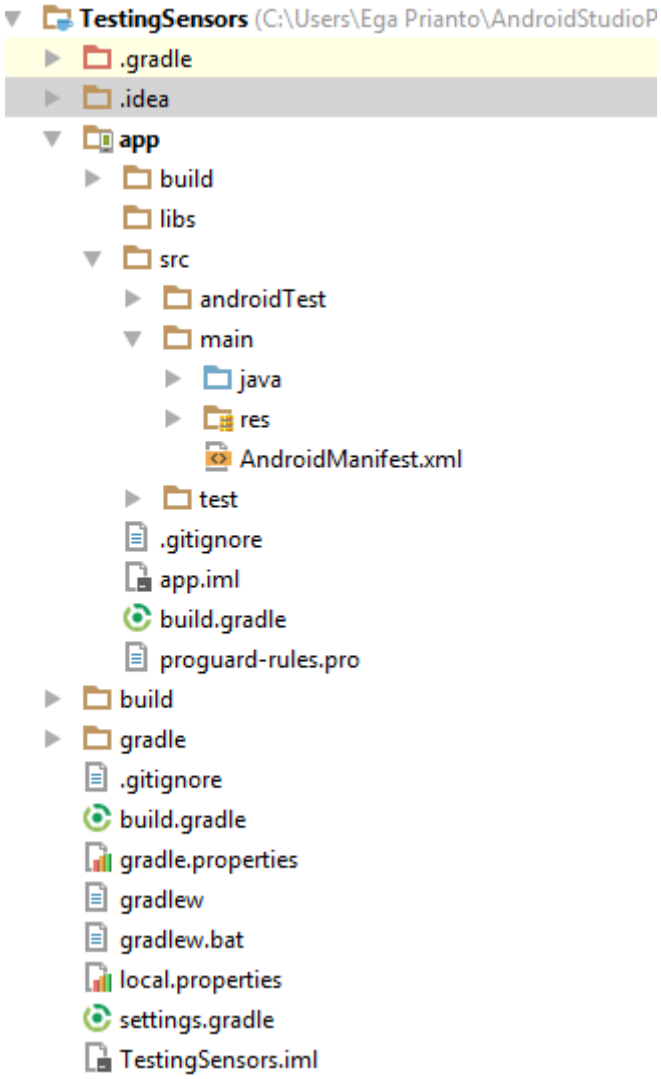
[8] Pada saat **project** baru telah dibuat, Android Studio akan membuatkan folder-folder standar (Gambar 2.1). Berikut adalah sebagian penjelasan dari hal yang perlu di perhatikan pada struktur tersebut:

- Folder ***module*/build** mengandung file-file dari hasil pembangunan *project*.
- Folder ***module*/libs** mengandung *libraries* privat.
- Folder ***module*/src** mengandung semua kode dan file sumber untuk suatu modul yang terbagi menjadi *subdirectories* berikut:
  - Folder **androidTest**/ mengandung kode untuk mengetes yang berjalan di perangkat Android.
  - Folder **main**/ mengandung file-file kode dan file sumber inti dari suatu module yang terbagi menjadi *subdirectories* berikut:
    - File **AndroidManifest.xml** mendeskripsikan sifat dari aplikasi dan setiap komponennya.
    - Folder **java**/ mengandung kode java.
    - Folder **jni**/ mengandung kode yang menggunakan Java Native Interface (JNI).
    - Folder **gen**/ mengandung file java yang dihasilkan oleh Android Studio.
    - Folder **res**/ mengandung file-file sumber untuk aplikasi, seperti file drawable, layout, dan UI String.

    * Folder **assets**/ mengandung file yang akan di kompile menjadi file **.apk**. File-file yang biasa disimpan di folder ini adalah file audio, video, file html, gambar dan file bantu lainnya.
  - File **build.gradle**(module) mendefinisikan konfigurasi modul untuk proses *build*.

- File **build.gradle**(project) mendefinisikan konfigurasi proses **build** untuk *project* yang berlaku untuk semua modul.

Folder **App** pada Gambar 2.1 merupakan folder *module*.



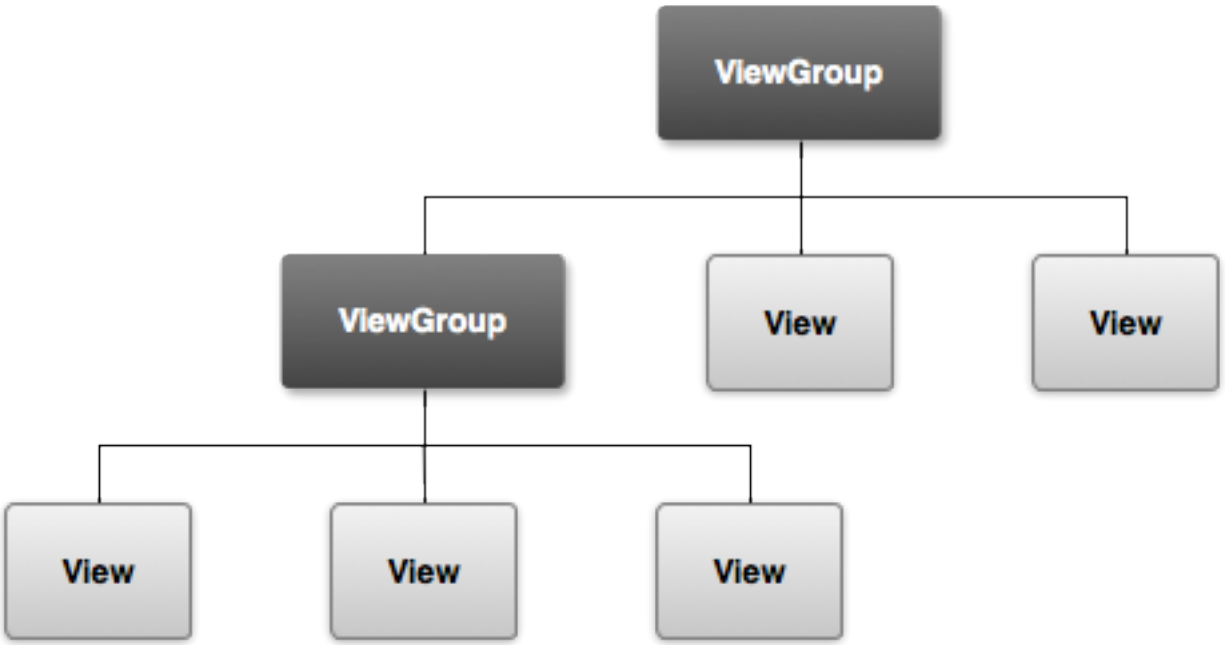
Gambar 2.1: Tampilan struktur folder pada *project* Android Studio

2.1.2 Membuat User Interface

[8] Pada subbab ini akan dijelaskan bagaimana membuat layout di XML termasuk *text field* dan *button*

Hierarki *Graphical User Interface* (GUI) untuk Aplikasi Android

GUI untuk aplikasi Android dibuat dengan hierarki dari objek View dan ViewGroup (Gambar 2.2). Objek-objek dari View biasanya adalah *UI(User Interface) Widgets* seperti *button* atau *text field*. Objek-objek dari ViewGroup tidak terlihat oleh *view containers* yang mendefinisikan bagaimana *child views* ditata seperti *grid* atau *vertical list*.



Gambar 2.2: Illustrasi bagaimana percabangan objek ViewGroup pada *layout* dan mengandung objek View lainnya.

Android menggunakan file XML yang berkorespondesi kepada *subclasses* dari View dan ViewGroup, sehingga UI dapat didefinisikan dalam XML menggunakan hierarki dari elemen UI.

Attribut-attribut Objek View

Pada subbab ini akan dijelaskan attribut-attribut object View yang digunakan dalam membuat GUI pada file activity_main.xml

- **android:id** Attribut ini merupakan pengidentifikasi dari suatu view. Attribut ini dapat digunakan untuk menjadi referrensi object dari kode aplikasi seperti membaca dan memanipulasi objek tersebut (Akan dijelaskan lebih lanjut pada subbab 2.1.3). Tanda '@' dibutuhkan ketika mereferensi object dari suatu XML. Tanda '@' tersebut diikuti dengan tipe (id pada kasus ini), *slash*, dan nama (edit_message pada Listing 2.2). Tanda tambah (+) sebelum tipe hanya dibutuhkan jika ingin mendefinisikan *resource ID* untuk pertama kalinya.

- **android:layout_width** dan **android:layout_height** Attribut ini digunakan untuk mendefinisikan panjang dan lebar dari suatu objek View. Daripada menggunakan besar spesifik untuk panjang dan lebarnya, lebih baik menggunakan "wrap_content" yang menspesifikasi viewnya hanya akan sebesar yang dibutuhkan untuk memuat konten-konten dari View. Ji-

ka menggunakan "match_parent" pada kasus Listing 2.2 View akan memenuhi layar, karena besarnya akan mengikuti besar dari paretnya LinearLayout.

- **android:hint** Attribut ini merupakan *default string* untuk di tampilkan ketika objek View kosong. Daripada menggunakan *hard-coded string* sebagai *nilai* untuk ditampilkan, *value* "@string/edit_message" mereferensi ke sumber string pada file yang berbeda. Karena mereferensi ke sumber konkrit, maka tidak dibutuhkan tanda tambah (+). Nilai string ini akan di simpan pada file Strings.xml yang ditunjukkan pada Listing 2.1.

Listing 2.1: Contoh kode pada string.xml

```
<resources>
<string name="app_name">MyFirstAndroidApp</string>
<string name="edit_message">Ini adalah hint</string>
<string name="button_send">Send</string>
</resources>
```

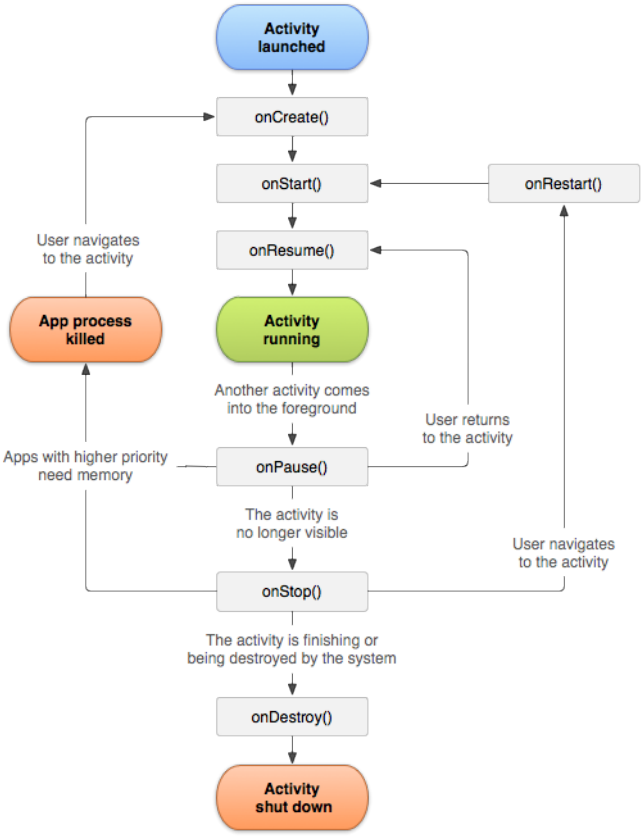
- **android:onClick** Attribut ini akan memberitahu *system* untuk memanggil method yang sesuai namanya (contoh pada Listing 2.2 adalah **sendMessage()**) di Activity ketika pengguna melakukan klik pada *button* tersebut. Agar *system* dapat memanggil method yang tepat, method tersebut harus memenuhi kriteria berikut.
  - *Access Modifier* haruslah *public*.
  - Harus *void return value*nya.
  - Mempunyai View sebagai parameter satu-satunya. View ini akan diisi dengan View yang di klik.

Listing 2.2: Contoh kode file XML pada folder layout

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="wrap_content"
              android:hint="@string/edit_message" />
    <Button
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="@string/button_send"
              android:onClick="sendMessage" />
</LinearLayout>
```

2.1.3 Activity

[8] Activity adalah suatu hal yang terfokuskan dengan apa yang bisa pengguna lakukan. Hampir semua *Activity* berinteraksi dengan pengguna, jadi kelas Activity akan membuat suatu halaman baru yang bisa ditambahkan dengan konten-konten View. Selain Activity dapat direpresentasikan kepada pengguna dengan halaman *full-screen*, Activity juga dapat direpresentasikan dengan cara lain: seperti halaman *floating* atau tertanam di Activity lain.



Gambar 2.3: *State diagram* siklus Activity

Activity Lifecycle

Aktivitas dalam sistem android di atur sebagai *activity stack*. Ketika ada Activity baru yang dimulai, Activity tersebut ditempatkan di paling atas pada *stack* dan menjadi Activity aktif. Activity sebelumnya akan tetap berada di bawah stack, dan tidak akan muncul lagi sampai Activity yang baru berakhir.

Activity didasari dari empat kondisi:

- Jika Activity berada di latar depan pada layar, Activity tersebut sedang aktif.
- Jika Activity sudah tidak terfokuskan tetapi masih dapat terlihat, Activity tersebut sedang berhenti sementara. Pada kondisi ini Activity tersebut masih berjalan, tapi bisa diberhentikan ketika system berada dalam situasi kekurangan memori.
- Jika suatu Activity benar-benar dihalangi oleh Activity lain, Activity tersebut telah berhenti. Activity tersebut akan tetap mengingat seluruh keadaan dan infomasi anggota tetapi, Activity tersebut tidak lagi terlihat oleh pengguna jadi tampilan jendelanya akan tersembunyi dan seringkali akan diberhentikan Activitynya ketika system membutuhkan memori.
- Jika suatu Activity sedang berhenti sementara atau berhenti total, sistem dapat membuang Activity dari memory dengan cara menanyakan kepada pengguna untuk memberhentikannya atau langsung diberhentikan oleh sistem. Jika Activity tersebut ditampilkan lagi kepada pengguna, Activity tersebut harus memulai dari awal dan kembali ke keadaan sebelumnya.

Gambar 2.3 menunjukkan pentingnya alur keadaan dari suatu Activity. Gambar segi empat merepresentasikan *callback methods* yang dapat diimplementasikan untuk melakukan operasi ketika Activity berubah kondisi. Oval berwarna merupakan kondisi-kondisi utama dari suatu Activity. Ada 3 *key loops* untuk memantau suatu Activity:

- *Entire lifetime* terjadi diantara pemanggilan pertama pada onCreate(Bundle) sampai ke satu pemanggilan akhir onDestroy(). Suatu Activity akan melakukan semua persiapan pada kondisi umum pada method onCreate(), dan melepaskan seluruh sisa pemrosesan pada method onDestroy().
- *Visible lifetime* terjadi antara pemanggilan onStart() sampai pemanggilan yang sesuai pada onStop(). Pada tahap ini pengguna dapat melihat Activity pada layar meskipun tidak berada pada *foreground* dan berinteraksi dengan pengguna.
- *Foreground lifetime* terjadi antara pemanggilan method onResume() sampai ke satu pemanggilan akhir onDestroy(). Pada tahap ini Activity berada di depan semua Activity lainnya dan sedang berinteraksi dengan user.

2.1.4 Android Sensor Framework

[8]Sebagian besar dari perangkat android sudah memiliki sensor yang mengukur gerakan, orientasi, dan berbagai keadaan lingkungan. Sensor-sensor ini dapat memberikan data mentah dengan tingkat akurasi yang tinggi. Sensor ini juga berguna untuk memantau pergerakan tiga dimensi atau posisi perangkat. Sensor ini juga dapat memantau perubahan keadaan lingkungan yang dekat dengan perangkat. Android mendukung tiga kategori sensor:

- **Sensor gerak** Sensor-sensor ini mengukur akselerasi dan rotasi pada tiga sumbu. Kategori sensor ini meliputi *accelerometers*, sensor gravitasi, *gyroscope*, dan *rotation vector*.
- **Sensor keadaan lingkungan** Sensor-sensor ini mengukur berbagai keadaan lingkungan seperti suhu udara, tekanan, pencahayaan, dan kelembaban. Kateori sensor ini termasuk barimeter, fotometer dan termometer.
- **Sensor posisi** Sensor-sensor ini mengukur posisi perangkat. Kategori sensor ini meliputi sensor orientasi dan magnetometer.

Android Sensor Framework membantu developers untuk mengakses berbagai jenis sensor. Beberapa sensor berbasis perangkat keras dan beberapa sensor berbasis perangkat lunak. Sensor berbasis perangkat keras mendapatkan data dengan langsung mengukur sifat lingkungan tertentu, seperti percepatan, kekuatan medan geomagnetik, atau perubahan sudut. Sensor berbasis perangkat lunak mendapatkan data dari satu atau lebih sensor berbasis perangkat keras. Sensor berbasis perangkat lunak ini juga terkadang disebut sensor virtual atau sensor sintetis. Pada tabel berikut akan dirincikan tipe-tipe setiap sensor posisi dan gerak, deskripsi, dan penggunaan umumnya.

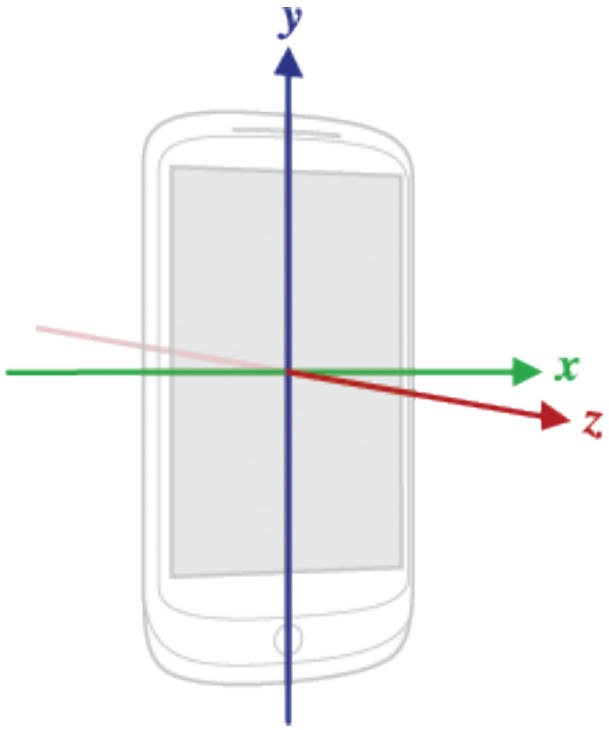
Sistem Koordinat Sensor

Pada umumnya, sensor framework menggunakan sistem tiga sumbu koordinat standar untuk mengekspresikan nilai data. Sebagian besar sensor sistem koordinat didefinisikan relatif terhadap layar

Tabel 2.1: Tipe-tipe Sensor pada Android

Sensor	Tipe	Deskripsi	Penggunaan umum
TYPE_ACCELEROMETER	Perangkat Keras	Mengukur percepatan dalam m/s^2 yang terjadi pada perangkat di semua tiga sumbu fisik (x,y,z), termasuk percepatan gravitasi.	Deteksi gerak(goncangan, keseimbangan, dan lain-lain)
TYPE_GRAVITY	Perangkat Lunak atau Perangkat Keras	Mengukur percepatan gravitasi dalam m/s^2 yang terjadi pada perangkat di tiga sumbu fisik (x,y, dan z)	Deteksi gerak(goncangan, keseimbangan, dan lain-lain)
TYPE_GYROSCOPE	Perangkat Keras	Mengukur rata-rata rotasi sudut dalam rad/s di tiga sumbu fisik (x,y, dan z).	Deteksi rotasi (putaran, belokan, dan lain-lain).
TYPE_LINEAR_ACCELERATION	Perangkat Lunak atau Perangkat Keras	Mengukur percepatan dalam m/s^2 yang terjadi pada perangkat di semua tiga sumbu fisik (x,y,z), tidak termasuk percepatan gravitasi.	Memantau percepatan pada suatu sumbu.
TYPE_MAGNETIC_FIELD	Perangkat Keras	Mengukur medan magnet sekitar untuk semua tiga sumbu fisik (x,y, dan z) di satuan μT.	Membuat Kompas.
TYPE_ORIENTATION	Perangkat Lunak	Mengukur derajat rotasi yang terjadi pada perangkat pada semua tiga sumbu fisik (x,y, dan z).	Menentukan posisi perangkat
TYPE_ROTATION_VECTOR	Perangkat Lunak dan Perangkat Keras	Mengukur orisentasi dari suatu perangkat dengan menyediakan tiga element dari vektor rotasi perangkat.	Deteksi gerak dan deteksi rotasi.

perangkat bila perangkat dibuat dalam orientasi standar (lihat Gambar 2.4) Sensor-sensor yang



Gambar 2.4: Sistem koordinat (relatif dengan perangkatnya) yang digunakan oleh Sensor API

menggunakan sistem tiga sumbu seperti Gambar 2.4 adalah sebagai berikut :

- Accelerometer
- Sensor Gravitasi
- Gyroscope
- Sensor Percepatan Linear
- Sensor Medan Geomagnetik

Koordinat sistem yang sumbunya tidak tertukar ketika orientasi perangkat berubah. Sistem koordinat sensor tidak pernah berubah seiring perangkatnya bergerak. Dalam aplikasi android tidak dapat diassumsikan bahwa standar orientasi perangkat android adalah *portrait*. Kebanyakan perangkat *Tablet* standar orientasinya adalah *landscape*. Sistem koordinat sensor selalu di dasarkan pada orientasi dasar dari suatu perangkat android.

Mengidentifikasi Sensor dan Kapabilitas Sensor

Android Sensor Framework menyediakan beberapa method yang dapat membuat developer mudah untuk menentukan sensor mana yang akan digunakan. APInya juga dapat menyediakan method yang memungkinkan penggunanya menentukan kapabilitas masing-masing sensor, seperti jangkauan maksimum, resolusi, dan kebutuhan dayanya. Untuk mengidentifikasi sensor-sensor yang ada pada perangkat, hal pertama yang perlu dilakukan adalah mendapatkan referensi sensor tersebut. Untuk

mendapatkan referensi tersebut, dapat dilakukan dengan membuat instansiasi dari kelas SensorManager dan memanggul method getSystemService() dan memasukkan isi Parameternya dengan SENSOR_SERVICE. Contohnya pada Listing 2.3.

Listing 2.3: Contoh inisialisasi kelas SensorManager

```
1 private SensorManager mSensorManager;
2 ...
3 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Kemudian untuk mendapatkan daftar dari setiap sensor pada suatu perangkat dapat dilakukan dengan cara memanggil method getSensorList() dan menggunakan konstanta TYPE_ALL pada kelas Sensor. Contohnya pada Listing 2.4

Listing 2.4: Contoh untuk mendapatkan daftar dari setiap sensor yang ada

```
1 List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

Namun jika ingin mendapatkan sensor-sensor yang sesuai dengan tipe sensor yang diberikan, dapat menggunakan TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, TYPE_GRAVITY, atau TYPE_GRAVITY.

Untuk menentukan jenis tertentu dari sensor yang ada pada perangkat dapat didapatkan dengan method getDefaultSensor() dengan dimasukkan dengan konstanta yang berada pada kelas Sensor. Jika perangkatnya memiliki lebih dari satu sensor dari tipe sensor yang diberikan, salah satu dari sensor tersebut akan dianggap sebagai sensor dasar. Jika sensor dasanya tidak ada untuk sensor tersebut, perangkat tersebut berarti tidak memiliki sensor dengan jenis yang diberikan. Listing 2.5 adalah contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan.

Listing 2.5: Contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan

```
1 private SensorManager mSensorManager;
2 ...
3 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
4 if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
5   // Sukses! Perangkat ini memiliki sensor magnetometer.
6   }
7 else {
8   // Gagal! Perangkat ini tidak memiliki sensor magnetometer.
9   }
```

Jika ingin membatasi versi atau vendor dari sensor yang akan digunakan, dapat menggunakan method getVendor() dan getVersion(). Seperti pada Listing 2.6 yang mengharuskan sensor gravitasi bervendor "Google Inc." dan memiliki versi 3. Jika sensor tersebut tidak tersedia pada perangkat, sensor accelerometerlah yang digunakan.

Listing 2.6: Contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan

```
1  private SensorManager mSensorManager;
2  private Sensor mSensor;
3
4  ...
5
6  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
7  mSensor = null;
8
9  if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
10   List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
11   for(int i=0; i<gravSensors.size(); i++) {
12     if ((gravSensors.get(i).getVendor().contains("Google Inc.")) &&
```

```
13          (gravSensors.get(i).getVersion() == 3)){
14        // menggunakan sensor gravitasi versi 3.
15        mSensor = gravSensors.get(i);
16      }
17    }
18  }
19  if (mSensor == null){
20    // Use the accelerometer.
21    if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
22      mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
23    }
24    else{
25      // Tidak ada sensor gravitasi dan sensor accelerometer!
26    }
27  }
```

Salah satu method yang sangat berguna lagi adalah, getMinDelay(). Method ini digunakan untuk mengetahui minimum interval waktu suatu sensor dapat menerima data dalam mikrodetik. Jika method getMinDelay() mengembalikan nilai nol, hal ini berarti sensor ini akan mengembalikan data setiap kali ada perubahan nilai pada sensor tersebut.

Memonitor Nilai Sensor

Untuk memonitor data mentah dari sensor, dibutuhkan untuk mengimplement dua buah method callback yang berada pada interface SensorEventListener. Kedua method tersebut adalah onAccuracyChanged(Sensor sensor, int accuracy) dan onSensorChanged(SensorEvent event). Sistem android akan memanggil kedua method ini ketika terjadi salah satu kondisi ini:

- **Perubahan akurasi sensor.**
  Dalam kasus ini sistem memanggil method onAccuracyChanged(Sensor sensor, int accuracy). Parameter sensor akan diberikan objek Sensor yang telah berubah akurasinya, dan parameter accuracy adalah nilai akurasi sensor yang baru.

- **Sensor memberitahu adanya nilai baru.**
  Dalam kasus ini sistem memanggil method onSensorChanged(SensorEvent event), dengan parameter event akan diisi dengan objek SensorEvent untuk mendapatkan nilai barunya. Objek SensorEvent Mengandung semua infromasi tentang data sensor yang baru, termasuk: akurasi dari data, sensor yang mendapatkan data, dan catatan waktu data tersebut didapatkan, dan data yang baru yang telah didapatkan.

Pada Listing 2.7 akan ditunjukkan bagaimana menggunakan method onSensorChanged(SensorEvent event) untuk memonitor data dari sensor cahaya. Pada Listing 2.7 akan menampilkan data mentah dari sensor ke TextView yang telah didefinisikan pada file main.xml sebagai sensor_data.

Listing 2.7: Contoh memonitor data mentah pada sensor cahaya

```
1  public class SensorActivity extends Activity implements SensorEventListener {
2    private SensorManager mSensorManager;
3    private Sensor mLight;
4
5    @Override
6    public final void onCreate(Bundle savedInstanceState) {
7      super.onCreate(savedInstanceState);
8      setContentView(R.layout.main);
9
10     mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
11     mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
12   }
13
14   @Override
```

```
15    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
16      // Hal yang perlu dilakukan aplikasi ketika akurasinya berubah.
17    }
18
19    @Override
20    public final void onSensorChanged(SensorEvent event) {
21      // Sensor cahaya akan mengembalikan 1 nilai saja.
22      // Banyak sensor lain yang akan mengembalikan lebih dari 1 nilai.
23      float lux = event.values[0];
24      // Hal yang perlu dilakukan ketika ada perubahan data.
25    }
26
27    @Override
28    protected void onResume() {
29      super.onResume();
30      mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
31    }
32
33    @Override
34    protected void onPause() {
35      super.onPause();
36      mSensorManager.unregisterListener(this);
37    }
38  }
```

Pada method onSensorChanged(SensorEvent event), struktur nilai-nilai yang dikembalikan akan dijelaskan pada subbab 2.1.4. Pada method onResume(), ada pemanggilan method registerListener(). Method registerListener() ini berguna untuk menspesifikasikan waktu delay pada pemanggilan onSensorChanged(). Untuk menspesifikasikan delaynya dapat menggunakan konstanta yang ada pada kelas SensorManager. Konstanta-konstanta tersebut adalah SENSOR_DELAY_NORMAL (200.000 mikrodetik),SENSOR_DELAY_GAME (20.000 mikrodetik),SENSOR_DELAY_UI (60.000 mikrodetik), atau SENSOR_DELAY_FASTEST (0 mikrodetik). Pengaturan dasar untuk waktu delay ini menggunakan konstanta SENSOR_DELAY_NORMAL. Perlu diperhatikan juga pemesanan sensor ketika activity sedang di berhentikan sementara maupun dilanjutkan kembali. Sistem tidak boleh tetap merekam sensor ketika activity tidak aktif. Hal ini diperlukan karena pada saat perangkat android menggunakan sensor, perangkat akan menggunakan tenanga yang banyak. Akan lebih baik jika penggunaan sensor diberhentikan ketika activitynya sudah tidak lagi digunakan.

Struktur Nilai yang Dikembalikan oleh Sensor

Setiap sensor android akan mengembalikan nilai-nilainya dengan struktur-struktur tertentu. Pada bab ini akan dijelaskan secara detil struktur nilai sistem android dalam mengembalikan nilai-nilai yang diperoleh dari sensor. Nilai ini akan didapatkan dengan tipe data array of float. Besar dan isi dari array tergantung pada sensor yang sedang di pantau. Berikut ini adalah struktur-struktur nilai dari setiap sensor pada sistem android :

- **TYPE_ACCELEROMETER**
  Semua nilai didefinisikan sebagai satuan m/s^2
  - values[0]: Percepatan yang terjadi pada sumbu x dikali -1
  - values[1]: Percepatan yang terjadi pada sumbu y dikali -1
  - values[2]: Percepatan yang terjadi pada sumbu z dikali -1

  Sensor ini mengukur percepatan(Ad) yang diterapkan pada perangkat. Sensor tersebut dapat mengukur percepatan dengan mengukur gaya(Fs) yang terjadi pada sensor menggunakan

relasi berikut:

$$Ad = -\Sigma Fs/mass$$

Secara khusus, gravitasi selalu mempengaruhi percepatan yang diukur :

$$Ad = -g - \Sigma F/mass$$

Karena inilah ketika perangkat android sedang diam, accelerometer membaca percepatan gravitasi sebesar $g = 9.81 m/s^2$. Demikian pula ketika perangkat android sedang dalam keadaan jatuh bebas. Perangkat akan mempercepat menuju ke tanah pada percepatan $9.81 m/s^2$, sehingga accelerometer membaca percepatan total sebesar $0 m/s^2$. Suatu saat akan di butuhkan untuk mengukur percepatan asli yang terjadi pada perangkat, sehingga kontribusi gravitasi harus di eliminasi. Hal ini bisa dilakukan dengan menerapkan *high-pass* filter. Sebaliknya, *low-pass* filter dapat digunakan untuk mendapatkan nilai gravitasi saja.

Listing 2.8: Implementasi *low-pass* filter

```
1   public void onSensorChanged(SensorEvent event)
2   {
3       // alpha dikalkulasikan sebagai t / (t + dT)
4       // dengan t adalah low-pass filter's time-constant
5       // dan dT, rata-rata event tersampaikan
6
7       final float alpha = 0.8;
8
9       gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
10      gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
11      gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];
12
13      linear_acceleration[0] = event.values[0] - gravity[0];
14      linear_acceleration[1] = event.values[1] - gravity[1];
15      linear_acceleration[2] = event.values[2] - gravity[2];
16  }
```

Low-pass filter dapat diimplementasikan pada Listing 2.8

- **TYPE_MAGNETIC_FIELD**
  Sensor ini mengukur medan magnet sekitar perangkat pada sumbu X,Y, dan Z dalam satuan micro-Tesla.

- **TYPE_GYROSCOPE**
  Sensor ini mengukur rata-rata perputaran pada perangkat yang berputar di sumbu X,Y, dan Z dalam satuan radians/second. Sistem koordinat yang digunakan sama dengan sistem koordinat pada sensor percepatan(Accelerometer). Jika perangkat berputar berlawanan arah jarum jam pada sumbu tertentu, maka rotasi yang terjadi akan bernilai positif. Perhatikan bahwa standar perputaran ini adalah definisi matematika standar pada rotasi positif.

  – values[0]: Percepatan angular pada sumbu X.
  – values[1]: Percepatan angular pada sumbu Y.
  – values[2]: Percepatan angular pada sumbu Z.

  Biasanya keluaran dari gyroscope terintegrasi dari waktu ke waktu untuk menghitung rotasi yang menggambarkan perubahan sudut atas langkah waktu, misalnya pada Listing 2.9

Listing 2.9: contoh implementasi gyroscope

```
1       private static final float NS2S = 1.0f / 1000000000.0f;
2       private final float[] deltaRotationVector = new float[4]();
3       private float timestamp;
4
5       public void onSensorChanged(SensorEvent event) {
6            // Pada tahapan ini delta rotasi akan dikalikan dengan rotasi saat ini
7            // setelah mengomputasinya dari data gyro.
8            if (timestamp != 0) {
9                 final float dT = (event.timestamp - timestamp) * NS2S;
10                // Sumbu dari rotasi, masih belum di normalisasi.
11                float axisX = event.values[0];
12                float axisY = event.values[1];
13                float axisZ = event.values[2];
14
15                // Menghitung percepatan angular
16                float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY + axisZ*axisZ);
17
18                // Normalisasi rotasi vektor jika cukup besar untuk mendapatkan sumbunya.
19                if (omegaMagnitude > EPSILON) {
20                     axisX /= omegaMagnitude;
21                     axisY /= omegaMagnitude;
22                     axisZ /= omegaMagnitude;
23                }
24
25                // Integrate around this axis with the angular speed by the time step
26                // in order to get a delta rotation from this sample over the time step
27                // We will convert this axis-angle representation of the delta rotation
28                // into a kuaternion before turning it into the rotation matrix.
29                float thetaOverTwo = omegaMagnitude * dT / 2.0f;
30                float sinThetaOverTwo = sin(thetaOverTwo);
31                float cosThetaOverTwo = cos(thetaOverTwo);
32                deltaRotationVector[0] = sinThetaOverTwo * axisX;
33                deltaRotationVector[1] = sinThetaOverTwo * axisY;
34                deltaRotationVector[2] = sinThetaOverTwo * axisZ;
35                deltaRotationVector[3] = cosThetaOverTwo;
36           }
37           timestamp = event.timestamp;
38           float[] deltaRotationMatrix = new float[9];
39           SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
40           // User code should concatenate the delta rotation we computed with the current rotation
41           // in order to get the updated rotation.
42           // rotationCurrent = rotationCurrent * deltaRotationMatrix;
43      }
44      }
```

Dalam prakteknya, gyroscope *noise* dan *offset* akan menyebabkan beberapa kesalahan yang harus dikompensasi. Cara untuk mengkompensasinya biasanya dilakukan dengan menggunakan informasi dari sensor lain.

- **TYPE_GRAVITY**

  Sensor ini menunjukkan arah dan besarnya vektor gaya gravitasi. Sensor ini mengembalikan nilai dengan satuan m/s^2. Sistem koordinat sama seperti sistem koordinat yang umum digunakan sensor percepatan.

  Catatan: Bila perangkat sedang diam, maka keluaran dari sensor gravitasi harus identik dengan accelerometer.

- **TYPE_LINEAR_ACCELERATION**

  Sensor yang menunjukkan percepatan pada setiap sumbu perangkat, tidak termasuk percepatan yang terjadi karena gravitasi. Nilai diberikan dalam satuan m/s^2. Sistem koordinat yang digunakan sama seperti sistem koordinat yang digunakan sensor percepatan. Keluaran

dari sensor accelerometer, gravitasi dan percepatan linear harus mengikuti aturan berikut:

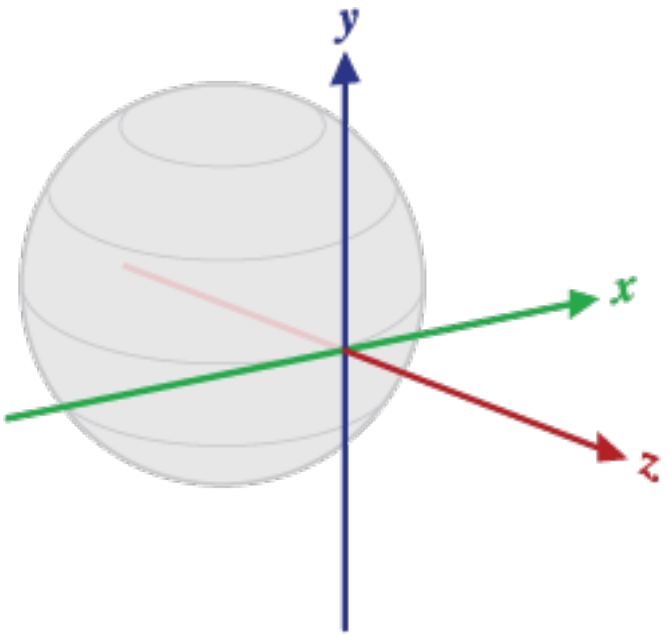
$$percepatan = gravitasi + percepatanlinear$$

- **TYPE_ORIENTATION**
  Semua nilai adalah sudut dalam derajat.
  - values[0]: Azimuth, sudut diantara arah magnetik utara dengan sumbu y, sekitar sumbu z (0 sampai 359). 0 = Utara, 90 = Timur, 180 = Selatan, 270 = Barat
  - values[1]: Pitch, rotasi sekitar sumbu x (-180 sampai 180), dengan nilai positif ketika sumbu x bergerak menuju sumbu y.
  - values[2]: Roll, perputaran sekitar sumbu y (-90 sampai 90) pada kondisi potrait, sensor akan bernilai 0. Pada kondisi landscape ke kanan sensor akan bernilai 90 dan sebaliknya yaitu kondisi landscape ke kiri sensor akan bernilai -90.

  Catatan: Definisi ini berbeda dengan definisi yaw, pitch, dan roll yang digunakan pada aviasi yang sumbu X adalah sepanjang sisi bidang.

  Catatan: Sensor ini sudah tidak digunakan lagi(deprecated), yang digunakan sekarang adalah sensor rotasi vector.

- **TYPE_ROTATION_VECTOR**
  Sensor ini merepresentasikan orientasi perangkat dengan kombinasi dari sumbu dan sudut. Perangkat akan di putar sebesar sudut θ mengelilingi sumbu x, y, z. Tiga elemen dari vektor rotasi adalah $(x\sin(\frac{\theta}{2}), y\sin(\frac{\theta}{2}), z\sin(\frac{\theta}{2}))$, sehingga besarnya vektor rotasi sama dengan $\sin(\frac{\theta}{2})$, dan arah vektor rotasi sama dengan sumbu rotasi. Tiga elemen dari vektor rotasi sama dengan tiga komponen terakhir pada unit kuaternion$(\cos(\frac{\theta}{2}), x\sin(\frac{\theta}{2}), y\sin(\frac{\theta}{2}), z\sin(\frac{\theta}{2}))$ yang dijelaskan pada subbab 2.3. Elemen dari vektor rotasi tak memiliki satuan. Sistem koordinat yang digunakan sama dengan sistem koordinat yang digunakan pada sensor percepatan. Referensi koordinat didefinisikan sebagai basis orthonormal, yaitu:
  - X didefinisikan sebagai perkalian dot product **Y.Z**
  - Y merupakan tangensial ke tanan pada lokasi perangkat saat ini dan menunjuk ke arah utara.
  - Z menghadap ke langit dan tegak lurus dengan tanah. Untuk lebih jelasnya dapat dilihat pada Gambar 2.5
  - values[0]: $x\sin(\frac{\theta}{2})$
  - values[1]: $y\sin(\frac{\theta}{2})$
  - values[2]: $z\sin(\frac{\theta}{2})$
  - values[3]: $\cos(\frac{\theta}{2})$
  - values[4]: Perkiraan akurasi (dalam radians) (-1 jika tidak tersedia)



Gambar 2.5: Sistem koordinat sensor rotasi vektor terhadap Bumi

2.2 Google VR SDK

Google VR SDK[9] digunakan untuk membantu dalam pembuatan aplikasi Virtual Reality pada *smartphone*. Google VR SDK memberikan beberapa fitur sebagai berikut :

- **Binocular rendering**: Fitur untuk tampilan layar terpisah untuk masing-masing dalam pandangan VR.
- **Spatial audio**: Fitur untuk mengeluarkan suara yang datang dari daerah-daerah tertentu dari dunia VR.
- **Head movement tracking**: Fitur untuk mendapatkan memperbaharui pengelihatan dunia VR yang sesuai dengan gerakan kepala pengguna.
- **Trigger input**: Fitur untuk memberikan input pada dunia VR dengan menekan tombol.

Ada beberapa persyaratan untuk menggunakan Google VR SDK, persyaratan tersebut adalah:

- Android Studio versi 1.0 atau lebih.
- Android SDK versi 23
- Gradle versi 23.0.1 atau lebih. Android Studio akan membantu meningkatkan versinya jika versinya terlalu rendah.
- Perangkat Android fisik yang menjalankan Android versi 4.4 (KitKat) atau lebih.

Dalam membuat aplikasi Google Cardboard VR membutuhkan beberapa API(Application Program Interface) dari Google VR SDK. API-API umum yang akan digunakan adalah sebagai berikut.

- API audio: API untuk mengimplementasikan ***Spatial Audio*** (Metode untuk menspasialisasikan sumber suara dalam ruang tiga dimensi).

- API base: API untuk fondasi dari suatu aplikasi Google VR.

2.2.1 API Audio

[9] API ini membantu developer untuk menspasialisasikan sumber suara dalam tiga dimensi, termasuk jarak dengan tinggi isyarat sumber suara. Pada API ini hanya terdapat satu class utama yaitu **GvrAudioEngine**. **GvrAudioEngine** mampu memutarkan suara secara spasial dalam dua cara yang berbeda :

- Metode pertama dikenal sebagai *Sound Object rendering*. Metode ini memungkinkan pengguna membuat sumber suara virtual dalam ruang tiga dimensi.
- Metode kedua memungkinkan pengguna untuk memutar kembali rekaman *Ambisonic soundfield*. Rekaman *Ambisonic soundfield* adalah file audio *multi-channel* yang telah terspasialisasi.

API ini juga dapat memutarkan suara secara *stereo*. Kelas **GvrAudioEngine** memiliki tiga buah *nested class* yaitu:

- GvrAudioEngine.DistanceRolloffModel: Kelas ini mendefinisikan konstanta-konstanta yang merepresentasikan perbedaan jarak dari efek model-model rolloff.
- GvrAudioEngine.MaterialName: Kelas ini mendefinisikan konstanta-konstanta yang merepresentasikan bahan permukaan ruangan untuk disesuaikan dengan efek suara pada suatu ruangan.
- GvrAudioEngine.RenderingMode: Kelas ini mendefinisikan konstanta-konstanta untuk menyesuaikan dengan mode rendering. Semakin baik kualitas renderin akan semakin besar penggunaan CPU(Central Processing Unit).

2.2.2 API Base

[9] API ini digunakan sebagai fondasi dari suatu aplikasi Google VR. Fitur-fitur Binocular rendering, Head movement tracking, dan Trigger input diimplementasikan pada API ini. Kelas-kelas penting yang ada di API ini adalah AndroidCompat, Eye, GvrActivity, GvrView, HeadTransform, Viewport.

- AndroidCompat
  Kelas ini merupakan kelas utilitas untuk menggunakan fitur VR. Fitur-fitur ini mungkin tidak tersedia pada semua versi android. Kelas ini memiliki method-method sebagai berikut:
  - setSustainedPerformanceMode(Activity activity, boolean enabled):
    Method ini digunakan untuk mengubah window android ke mode performa secara berkelanjutan.
  - public static void setVrModeEnabled (Activity activity, boolean enabled):
    Mengatur pengaturan yang tepat untuk "mode VR" pada suatu Activity. Method ini tidak digunakan karena hanya dapat digunakan pada Android N+.
  - public static boolean trySetVrModeEnabled (Activity activity, boolean enabled): Method ini kegunaanya sama dengan method **setVrModeEnabled (Activity activity, boolean enabled)**, namun mengembalikan boolean true jika berhasil dan sebaliknya.

- Eye
  Kelas ini mendefinisikan detil perenderan stereoskopik mata. Method penting yang dimiliki kelas ini adalah **public float[] getEyeView ()**. Method ini mengembalikan matriks yang mentransformasikan camera virtual ke mata. Transformasi yang diberikan termasuk melacak rotasi kepala, perubahan posisi dan perubahan IPD(interpupillary distance).

- GvrActivity
  Kelas ini merupakan Activity dasar yang menyediakan integrasi yang mudah dengan headset Google VR. Kelas ini mengekspos kejadian untuk berinteraksi dengan headset Google VR dan menangani detil-detil yang biasa diperlukan saat membuat suatu Activity untuk perenderan VR. Activity ini membuat layar tetap menyala selama perangkat android bergerak. Jika tidak ada pergerakan dari perangkat android maka layar reguler (*wakeclock*) akan ditampilkan. Pada kelas ini terdapat method **onCardboardTrigger ()** untuk mendeteksi ketika Cardboard Trigger sedang ditarik dan dilepaskan (Magnet yang berada pada sisi Google Cardboard).

- GvrView
  Kelas ini merupakan kelas View yang menyediakan perenderan VR. Kelas ini didesain untuk berkerja pada mode layar penuh dengan orientasi *landscape* atau *reverse landscape*. Kelas View ini dapat digunakan dengan mengimplements salah satu Interface perenderan. Interface-interface tersebut adalah:

  - GvrView.StereoRenderer: Interface untuk perenderan detil stereoskopik secara abstrak oleh perender.
  - GvrView.Renderer: Interface untuk mesin yang kompleks yang membutuhkan untuk menangai semua detil perenderan.

  Kelas GvrView.Renderer jarang digunakan dan sebaiknya tidak digunakan jika tidak sangat dibutuhkan. Ketika suatu kelas mengimplement Kelas GvrView.StereoRenderer, kelas tersebut harus mengimplementasikan method-method berikut ini:

  - **public void onNewFrame(HeadTransform headTransform)**
    method ini terpanggil ketika Framebaru akan digambar. Method ini memungkinkan untuk membedakan antara perenderan pandangan mata dan frame-frame yang berbeda. Setiap operasi per-frame harus tidak spesifik pada satu tampilan saja.
  - **public abstract void onDrawEye (Eye eye)**
    Method ini meminta untuk menggambar suatu konten dari sudut pandang mata.
  - **public abstract void onFinishFrame (Viewport viewport)**
    Method ini dipanggil ketika suatu frame telah selesai. Dengan pemanggilan ini, konten frame telah di gambar dan jika koreksi distorsi diaktifkan, koreksi distrosi akan diterapkan. Setiap perendereran pada tahap ini relatif terhadap seluruh permukana, tidak terhadap satu pandangan mata tunggal.
  - **public abstract void onRendererShutdown ()**
    Method ini dipanggil ketika thread perender sedang menutup. Melepaskan sumber GL(Graphics Library) dan sedang melakukan penutupan operasi pada thread perender. Dipanggil hanya jika sebelumnya ada pemanggilan method onSurfaceCreated.

  - **public abstract void onSurfaceChanged (int width, int height)**
    Dipanggil ketika ada perubahan dimensi permukaan. Semua nilai adalah relatif ke ukuran yang dibutuhkan untuk merender sebuah mata.
  - **public abstract void onSurfaceCreated (EGLConfig config)**
    Method ini dipanggil ketika suatu permukaan dibangun atau dibangun ulang.

- HeadTransform
  Method ini mendeskripsikan transformasi kepala secara independen dari setiap parameter mata. Kelas ini digunakan di kelas GvrView.StereoRenderer sebagai parameter pada method **onNewFrame**. Method-method yang perlu diperhatikan pada kelas ini adalah:
  - **public void getHeadView (float[] headView, int offset)**
    Method ini digunakan untuk mendapatkan matriks transformasi dari camera virtual ke kepala. Kepala disini didefinisikan sebagai titik tengah diantara kedua mata. Matriks yang didapatkan akan disimpan pada parameter **headView**.
  - **public void getQuaternion (float[] quaternion, int offset)**
    Method ini digunakan untuk mendapatkan kuaternion yang merepresentasikan rotasi kepala.

- Viewport
  Kelas ini didefinisikan sebagai *viewport*(area pandang) berbentuk persegi.

2.3 Teori Kuaternion

Pada Android SDK **SensorEvent.values** [8] tipe sensor **Sensor.TYPE_ROTATION_VECTOR**, yaitu tipe sensor yang mendeteksi vektor perputaran pada *smartphone*. Tipe sensor ini dijelaskan akan mengembalikan nilai-nilai dari komponen kuaternion. Kuaternion[10] adalah objek penggabungan dari suatu skalar dengan suatu vektor, sesuatu yang tidak dapat didefinisikan dalam aljabar linear biasa. Kuaternion ditemukan oleh William Rowan Hamilton dengan memperpanjang notasi dari bilangan kompleks menjadi Kuaternion.

2.3.1 Struktur Ajabar

Karena Kuaternion merupakan bilangan kompleks yang diperpanjang notasinya, struktur aljabar Kuaternion hampir mirip dengan bilangan kompleks. Untuk mengerti struktur-struktur aljabar kuaternion, diperlukan untuk mengerti bilangan kompleks terlebih dahulu. Berikut adalah penjelasan singkat tentang bilangan kompleks.

Bilangan Kompleks

[10] Bilangan kompleks adalah bilangan yang merupakan gabungan dari bilangan imajiner dengan bilangan riil. Notasi umum dari bilangan kompleks adalah :

$$a + bi \tag{2.1}$$

Pada notasi 2.1 bilangan a dengan b merupakan bilangan riil, dan i merupakan bilangan imajiner tertentu yang memiliki sifat $i^2 = -1$. Bilangan kompleks juga dapat beroperasi dengan bilangan kompleks lainnya seperti penjumlahan, perkalian, pengurangan, dan pembagian. Berikut adalah contoh-contoh operasi pada bilangan kompleks 2.1 dengan bilangan kompleks $c + di$:

- Penjumlahan

$$(a + bi) + (c + di) = (a + c) + i(b + d)$$

- Perkalian

$$(a + bi)(c + di) = (acbd) + (bc + ad)i$$

- Pengurangan

$$(a + bi) - (c + di) = (a - c) + i(b - d)$$

- Pembagian

$$\frac{(a + bi)}{(c + di)} = \frac{(ac + bd)}{c^2 + d^2} + i\frac{bc - ad}{c^2 + d^2}$$

Pada operasi penjumlahan dan perkalian untuk kedua bilangan kompleks tersebut memiliki hukum assosiatif dan komutatif. Notasi 2.2 menunjukkan bagaimana bagaimana kedua hukum tersebut berlaku pada penjumlahan.

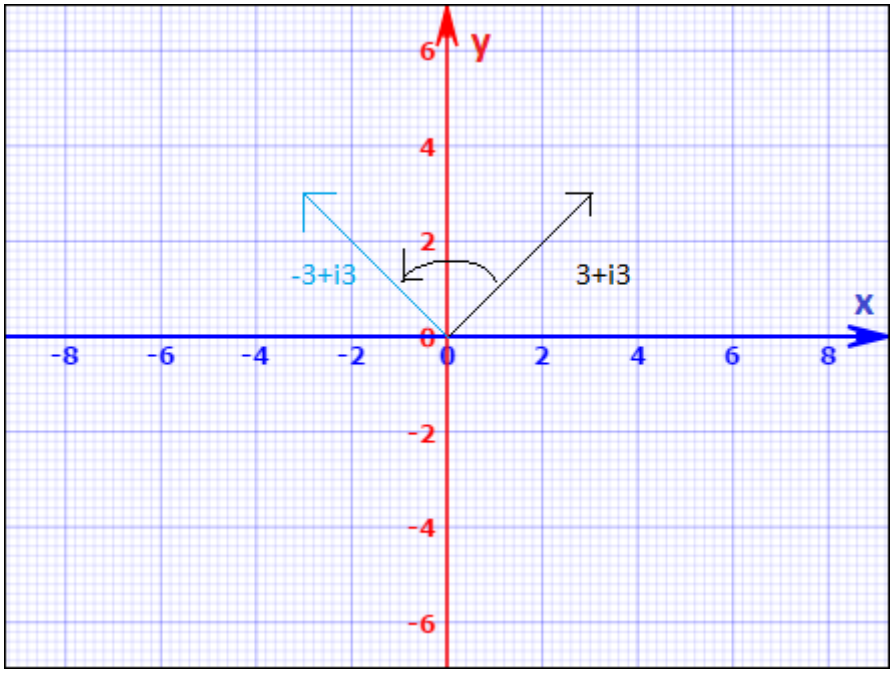
$$\begin{aligned}(a + ib) + (c + id) = (a + c) + i(b + d) = \\ (c + id) + (a + ib) = (c + a) + i(d + b)\end{aligned} \tag{2.2}$$

Suatu bilangan dapat dikatakan konjugasi kompleks dari suatu bilangan kompleks jika nilai bilangan riilnya sama, tetapi nilai bilangan imajinernya berlawanan dengan nilai pada bilangan kompleks tersebut. Maka konjugasi kompleks dari bilangan kompleks 2.1 adalah $a - bi$.

Bilangan kompleks ini dapat digunakan untuk rotasi vektor pada bidang dua dimensi. Rotasi ini dapat dilakukan dengan mengalikan suatu vektor dengan bilangan imajiner i. Mengalikan suatu vektor dengan bilangan imajiner i akan memutar vektor sebesar 90° berlawanan arah jarum jam. Mengalikan suatu vektor dengan bilangan imajiner i^2 akan memutar vektor sebesar 180° berlawanan arah jarum jam. Untuk memperjelas perputaran dengan bilangan kompleks diberikan contoh berikut: Sebuah vektor $v = 3 + i3$ akan diputar 90° berlawanan arah jarum jam dengan mengalikan vektor tersebut dengan bilangan imajiner i. Maka vektor hasil perputarannya(v') adalah :

$$\begin{aligned}v' =& i(3 + i3) \\ =& i3 + i^2 3 \\ =& i3 + (-1)3 \\ =& -3 + i3\end{aligned} \tag{2.3}$$

Dengan bilangan pada bilangan riil diasumsikan sebagai nilai pada sumbu x dan bilangan yang



Gambar 2.6: Contoh perputaran dengan bilangan kompleks

dikalikan dengan bilangan i diasumsikan pada sumbu y$(x + iy)$ Seperti yang ditunjukkan pada Gambar 2.6. Oleh karena itu rumus perputaran menggunakan bilangan kompleks dapat di rumuskan sebagai berikut:

$$v' = v \times (\cos\theta + i\sin\theta)$$

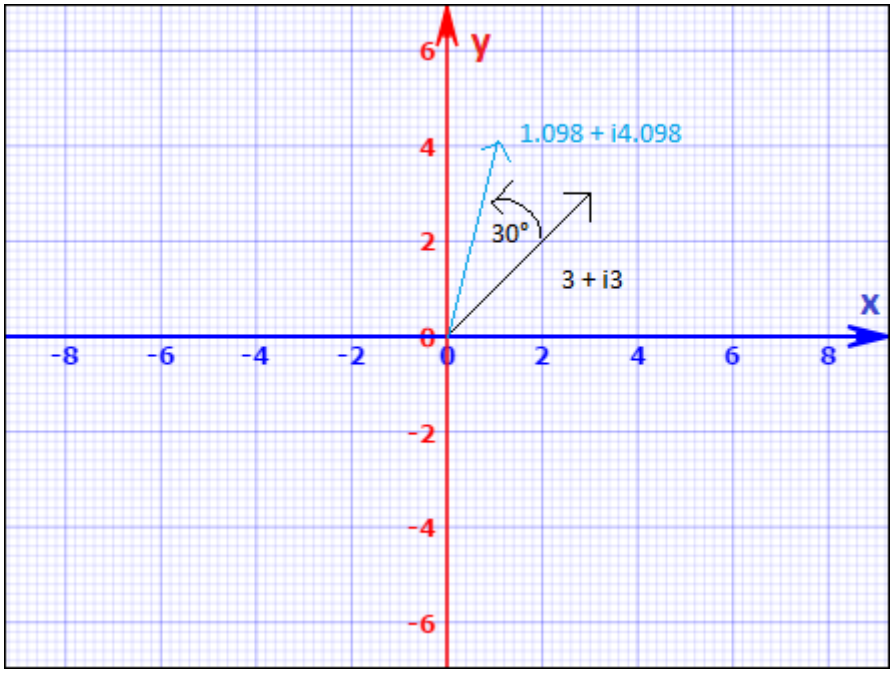
dengan θ adalah besar sudut perputaran. Jika vektor $v = 3+i3$ diputar 30° berlawanan arah jarum jam menggunakan konsep diatas, akan menghasilkan vektor berikut:

$$\begin{aligned} v' =& (3 + i3)(\cos 30° + i\sin 30°) \\ =& (3 + i3)(\frac{\sqrt{3}}{2} + i\frac{1}{2}) \\ =& \frac{3\sqrt{3} - 3}{2} + i\frac{3\sqrt{3} + 3}{2} \\ =& 1.098 + i4.098 \end{aligned} \tag{2.4}$$

Dari persamaan tersebut dapat divisualisasikan pada Gambar 2.7.

2.3.2 *Aljabar Kuaternion* dan Operasi-operasi pada Kuaternion

[10] Kuaternion ditemukan oleh ahli matematika dan astronomi Inggris, William Rowan Hamilton, dengan memperpanjang aritmatika dari bilangan kompleks. Dari penemuan tersebut William Rowan Hamilton menemukan bahwa dia tidak hanya membutuhkan bilangan imajiner i saja untuk melakukan rotasi pada ruang tiga dimensi. Dia menemukan bahwa dia juga membutuhkan tiga komponen imajiner lainnya yaitu i, j dan k. Persamaan umum Kuaternion memiliki empat bilangan



Gambar 2.7: Contoh perputaran tiga puluh derajat dengan bilangan kompleks

riil atau skalar. Persamaan tersebut adalah

$$q = q_0 + iq_1 + jq_2 + kq_3 \tag{2.5}$$

Ketiga komponen tersebut memiliki relasi sebagai berikut:

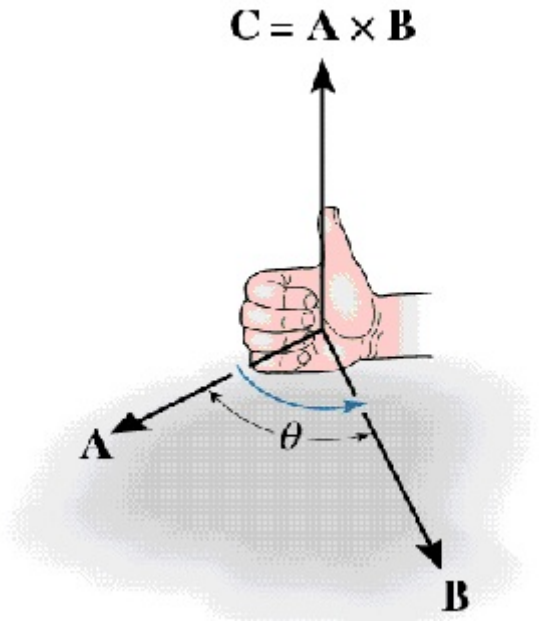
$$i^2 = j^2 = k^2 = ijk = -1$$

Hasil dari perkalian dua *kuaternion* memiliki aturan yang lebih rumit, sehingga memiliki aturan-aturan khusus. Berikut aturan-aturan khususnya :

$$\begin{aligned} ij &= k = -ji \\ jk &= i = -kj \\ ki &= j = -ik \end{aligned} \tag{2.6}$$

Ketiga persamaan diatas mirip dengan aturan tangan kanan (*right-hand rule*)pada perkalian cross product dari suatu vector. Pada Gambar 2.8, A berperan sebagai i, B berperan sebagai j, dan C berperan sebagai k.

Seperti pada bilangan kompleks, kuaternion juga memiliki konjugasinya. Konjugasi kuaternion ini akan digunakan untuk melakukan operasi rotasi tiga dimensi(Akan dijelaskan pada subsubbab berikut). Sama dengan konjugasi pada bilangan kompleks, kuaternion yang bilangan riilnya sama, dan bilangan imajinernya berlawanan dengan kuaternionnya disebut dengan konjugasi kuaternion.



Gambar 2.8: Right-hand rule dalam *cross product* vektor

Oleh karena itu konjugasi kuaternion dari notasi kuaternion 2.5 adalah:

$$q = q_0 - iq_1 - jq_2 - kq_3$$

Operasi pada Kuaternion

Dua buah kuaternion dapat dikatakan identik jika dan hanya jika kedua kuaternion memiliki komponen yang identik.

$$p = p_0 + +ip_1 + jp_2 + kp_3$$

dan,

$$q = q_0 + iq_1 + jq_2 + kq_3$$

maka p = q jika dan hanya jika

$$\begin{aligned} p_0 &= q_0 \\ p_1 &= q_1 \\ p_2 &= q_2 \\ p_3 &= q_3 \end{aligned} \tag{2.7}$$

Penjumlahan dari kedua kuaternion diatas dapat didefinisikan sebagai komponen penjumlahan yaitu:

$$(p + q) = (p_0 + q_0) + i(p_1 + q_1) + j(p_2 + q_2) + k(p_3 + q_3)$$

Perkalian dari kedua kuaternion diatas dapat didefinisikan sebagai komponen perkalian yaitu:

$$pq = (p_0 + ip_1 + jp_2 + kp_3)(q_0 + iq_1 + jq_2 + kq_3)$$

Begitu pula untuk komponen pengurangan dengan pembagian. Dari keempat operasi kuaternion tersebut, operasi kuaternion yang digunakan untuk rotasi bidang tiga dimensi adalah operasi perkalian. Fungsi rotasi vektor dapat menggunakan operasi kuaternion seperti pada bilangan kompleks, dengan rumus:

$$v' = qvq^*$$

dengan,

- $v = 1 + x_v i + y_v j + z_v k$
- $q = q_0 + iq_1 + jq_2 + kq_3$
- $q^* = q_0 - iq_1 - jq_2 - kq_3$
- $v' = 1 + x_{v'} i + y_{v'} j + z_{v'} k$

Seperti pada bilangan kompleks, bilangan q_0, iq_1, jq_2, kq_3 akan memiliki nilai sebagai berikut jika suatu kuaternion ingin digunakan untuk rotasi tiga dimensi:

$$\begin{aligned} q_0 &= \cos(\frac{\theta}{2}) \\ q_1 &= \sin(\frac{\theta}{2}) x_f \\ q_2 &= \sin(\frac{\theta}{2}) y_f \\ q_3 &= \sin(\frac{\theta}{2}) z_f \end{aligned} \tag{2.8}$$

dengan vektor f (x_f, y_f, z_f) merupakan sumbu perputaran dan θ merupakan besar sudut putar berlawanan arah dengan jarum jam.

BAB 3

ANALISIS

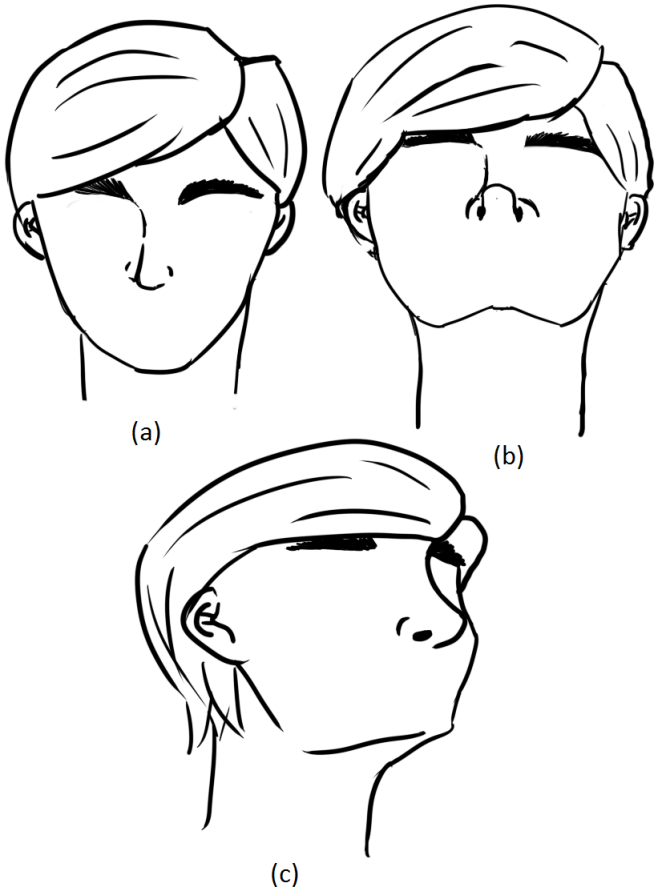
Pada bab ini akan dijelaskan mengenai analisis grafik dari data sensor-sensor pada *smartphone* ketika sedang mengangguk dan menggeleng, analisis aplikasi-aplikasi sejenis, dan analisis metode pendeteksi gerakan kepala.

3.1 Perekaman Data Sensor

Pada analisis ini akan dilakukan perekaman mengganguk dan menggeleng dengan sensor-sensor pada Android. Perekaman-perekaman ini akan dilakukan pada tiga kondisi muka pengguna. Kondisi muka yang pertama adalah kondisi muka pengguna ketika menghadap ke depan, digambarkan dengan Gambar 3.1 bagian (a). Kondisi muka yang kedua adalah kondisi muka pengguna keita menghadap ke atas sekitar 45° dari pandangan muka menghadap ke atas digambarkan dengan Gambar 3.1 bagian (b). Kondisi muka yang ketiga adalah kondisi muka ketika menghadap serong ke kiri atas digambarkan dengan Gambar 3.1 bagian (c). Anggukan yang dilakukan oleh pengguna hanya sebanyak satu kali mengangguk ke bawah saja. Sedangkan dalam menggeleng akan bergerak ke kiri terlebih dahulu dan ke kanan setelahnya dan diakhiri pada posisi muka kembali ke posisi awal.

Grafik-grafik yang akan ditunjukkan pada bab ini akan memiliki beberapa karakteristik. Sumbu y pada grafik yang akan ditunjukkan akan merepresentasikan besar nilainya, Sedangkan sumbu x akan merepresentasikan waktunya. Grafik yang ditunjukkan akan memiliki beberapa nilai, tergantung dari jumlah nilai yang dikembalikan untuk setiap sensornya. Contohnya pada sensor *accelerometer* yang memiliki tiga jenis nilai, sehingga pada grafik akan terbentuk tiga buah garis nilai. Aplikasi akan merekam beberapa sensor secara langsung ketika pengguna mengangguk atau menggeleng, sehingga nilai wakunya akan sama untuk kondisi muka yang sama walaupun sensornya berbeda.

Analisis grafik data sensor-sensor pada Android dilakukan dengan membuat suatu aplikasi perekam sensor-sensor yang ada pada *smartphone* Android terlebih dahulu. Aplikasi ini akan merekam nilai-nilai yang dihasilkan dari sebagian sensor-sensor pada android setiap ada perubahan. Pada skripsi ini, nilai sensor-sensor yang dibutuhkan adalah sensor *accelerometer*, *gyroscope*, *rotation vector*, dan *geomagnetic rotation*. Aplikasi menyimpan nilai sensor-sensor menggunakan format CSV *(Comma Separated Values)*. Dari data yang diperoleh oleh apliasi tersebut dibuatkan grafiknya menggunakan aplikasi Microsoft Excel. Penjelasan dari setiap grafik akan dijelaskan pada subbab-subbab berikut.



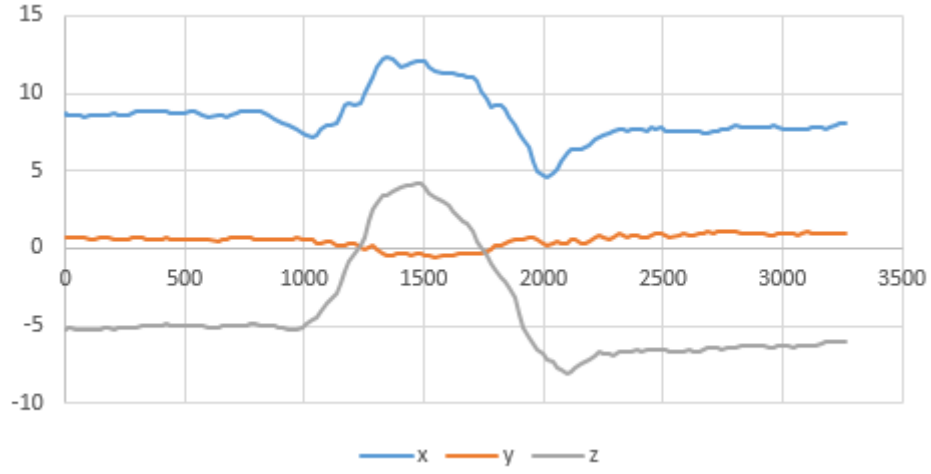
Gambar 3.1: Gambar untuk mendeskripsikan posisi dari muka sebelum melakukan gerakan menggeleng atau mengangguk. Gambar (a) adalah kondisi muka ketika sedang menghadap ke depan. Gambar (b) adalah kondisi muka ketika sedang menghadap ke atas. Gambar(c) adalah kondisi muka ketika sedang menghadap ke serong kiri atas.

3.1.1 Analisis Grafik Sensor *Accelerometer*

Seperti yang sudah dijelaskan pada bab sebelumnya, sensor *accelerometer* akan mendeteksi seluruh percepatan yang terjadi pada perangkat Android. Perekaman ini perangkat android akan diletakkan di depan muka pengguna, sehingga percepatan yang mempengaruhi perangkat Android hanya percepatan gravitasi dengan percepatan yang dilakukan oleh gerakan kepala pengguna. Gambar 3.2 merupakan grafik yang terbentuk dari nilai yang di terima oleh sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke depan.

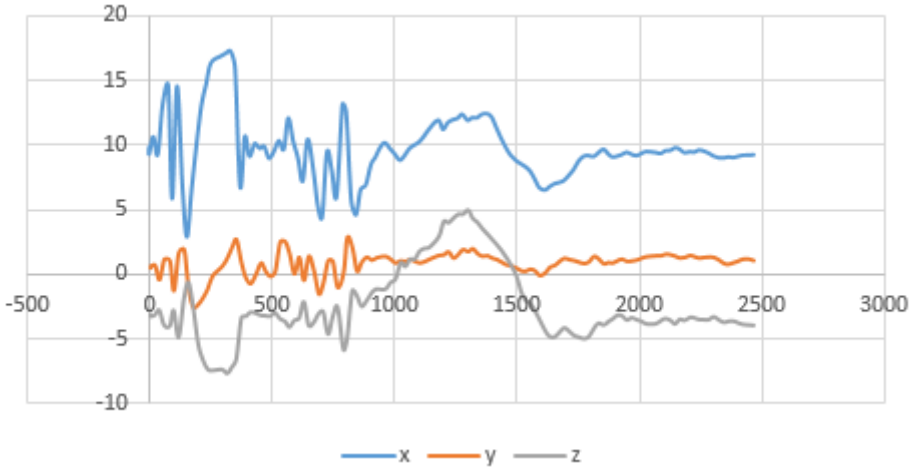
Gambar 3.2: Gambar grafik nilai sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke depan.

Pada Gambar 3.2 terlihat nilai z menaik dan nilai x menurun ketika sedang mengangguk. Tetapi nilai x kembali menaik ketika nilai z sudah hampir mencapai nilai tertinggi. Sedangkan nilai y terlihat cukup konstan di sekitar angka 0. Kemudian Gambar 3.3 merupakan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke atas.



Gambar 3.3: Gambar grafik nilai sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke atas.

Pada Gambar 3.3 terlihat nilai x dengan y memiliki pola yang serupa ketika mengangguk. Kedua nilai menaik ketika pengguna sedang mengangguk. Nilai x bermulai dari nilai sekitar sebesar 9 sedangkan nilai z bernilai sekitar sebesar -5. Sama seperti pada Gambar 3.2 nilai y terlihat konstan di sekitar angka 0. Selanjutnya Gambar 3.4 merupakan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke kiri atas.



Gambar 3.4: Gambar grafik nilai sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke kiri atas.

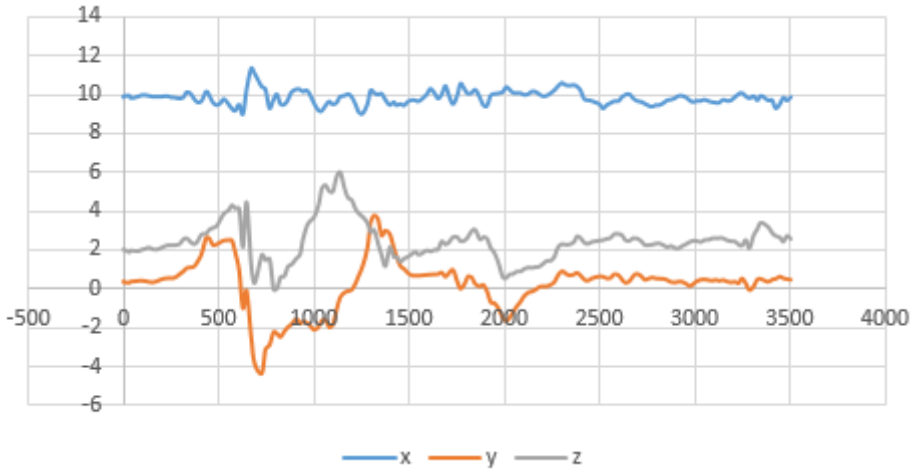
Pada Gambar 3.4 grafik tersebut terlihat berantakan. Pada Grafik ini sulit untuk membedakan kondisi kapan pengguna sedang mengangguk. Goncangan yang terjadi terhadap *smartphone*, seperti munculnya notifikasi mungkin dapat menyebabkan hal ini. Namun pada waktu mencapai 1000 milidetik terlihat cukup stabil. Pada Gambar 3.4 juga menunjukkan bahwa nilai x dengan z memiliki pola yang sama hingga akhir, dan nilai y konstan di sekitar angka 0. Hasil nilai tersebut serupa dengan kasus ketika menghadap ke atas. Gambar 3.5 merupakan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke depan.

Pada grafik di Gambar 3.5, nilai x konstan di sekitar angka 10. Nilai y dengan z menaik dan menurun ketika pengguna menggelengkan kepala. Berbeda dengan kasus pada Gambar 3.3 dan pada Gambar 3.4 yang nilai x dengan z memiliki pola yang serupa, sedang pada kasus ini nilai y dan z tidak memiliki pola yang serupa. Kemudian pada Gambar 3.6 menunjukan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke atas.

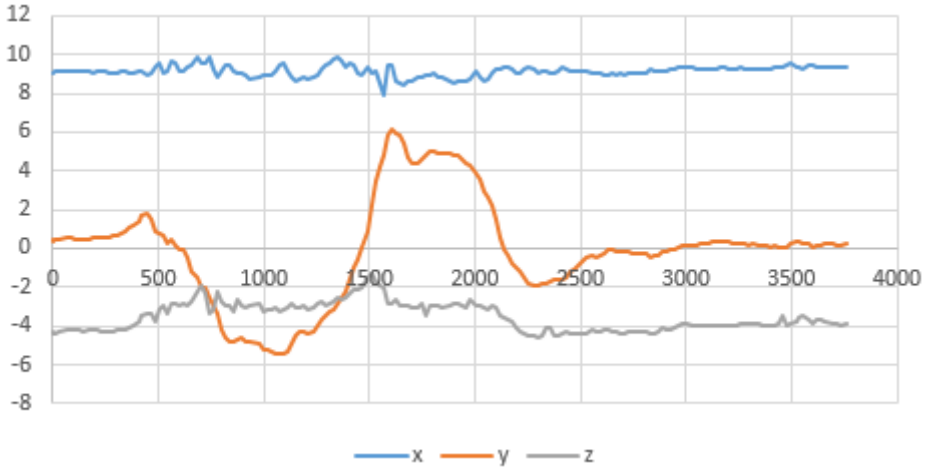
Berbeda dengan kasus pada Gambar 3.5, nilai x pada grafik di Gambar 3.6 terlihat konstan di sekitar nilai 10 dan nilai z di sekitar nilai -3. Nilai y mengalami kenaikan dan penurunan saat menggeleng. Gambar 3.6 menunjukan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke kiri atas.

Pada grafik di Gambar 3.7 nilai x konstan di sekitar nilai 10. Nilai y menaik dan menurun, tetapi tidak beraturan. Nilai pada z juga mengalami sedikit kenaikan dan penurunan.

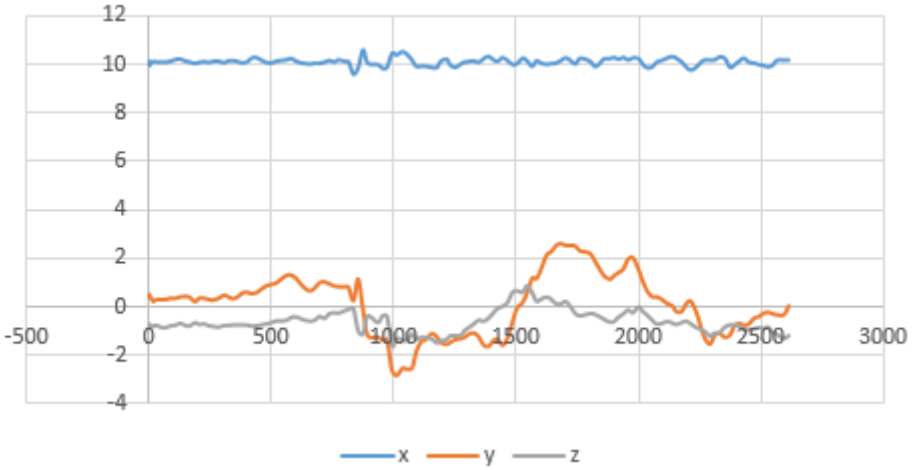
Dari keenam grafik tersebut terlihat bahwa nilai yang terpengaruh ketika pengguna sedang



Gambar 3.5: Gambar grafik nilai sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke depan.



Gambar 3.6: Gambar grafik nilai sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke atas.

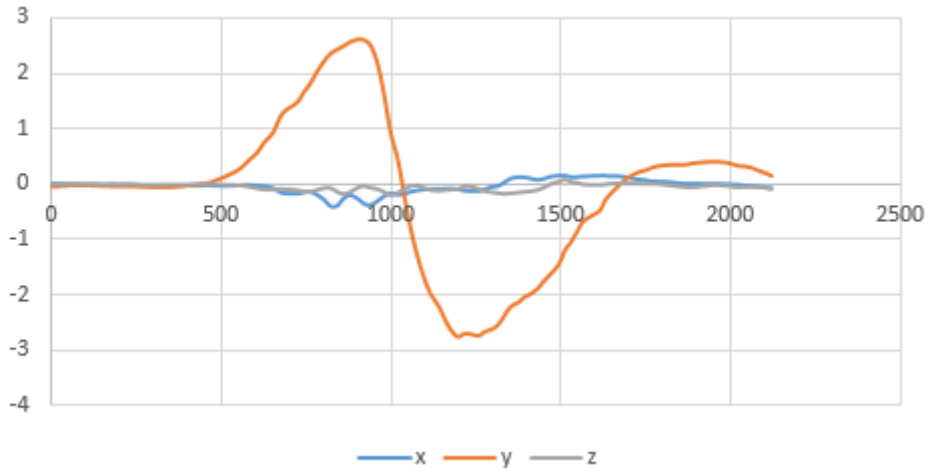


Gambar 3.7: Gambar grafik nilai sensor *accelerometer* ketika pengguna menggeleng dan sedang menghadap ke kiri atas.

mengangguk adalah nilai x dengan z. Nilai y tidak berpengaruh karena nilai y cenderung konstan. Nilai yang terpengaruh ketika pengguna sedang menggeleng adalah nilai y. Nilai x terlihat konstan pada setiap grafik, tetapi nilai z mengalami sedikit pergerakan ketika pengguna sedang mengangguk sehingga tidak dapat dipastikan bahwa nilai z terpengaruhi gerakan menggeleng.

3.1.2 Analisis Grafik Sensor *Gyroscope*

Perekaman menggunakan sensor *gyroscope* akan mendapatkan percepatan angular yang terjadi setiap waktunya. Berbeda dengan sensor *accelerometer* yang dapat terpengaruhi oleh lingkungan sekitar seperti gravitasi dan percepatan lainnya, sensor ini hanya akan merekam perputaran yang terjadi pada perangkat saja. Hal ini sangat menguntungkan dalam mendeteksi suatu gerakan karena tidak harus memperdulikan kasus dari pengaruh luar. Gambar 3.8 menunjukkan nilai-nilai sensor *gyroscope* ketika pengguna sedang mengangguk dan menghadap ke depan.



Gambar 3.8: Gambar grafik nilai sensor *gyroscope* ketika pengguna mengangguk dan sedang menghadap ke depan.

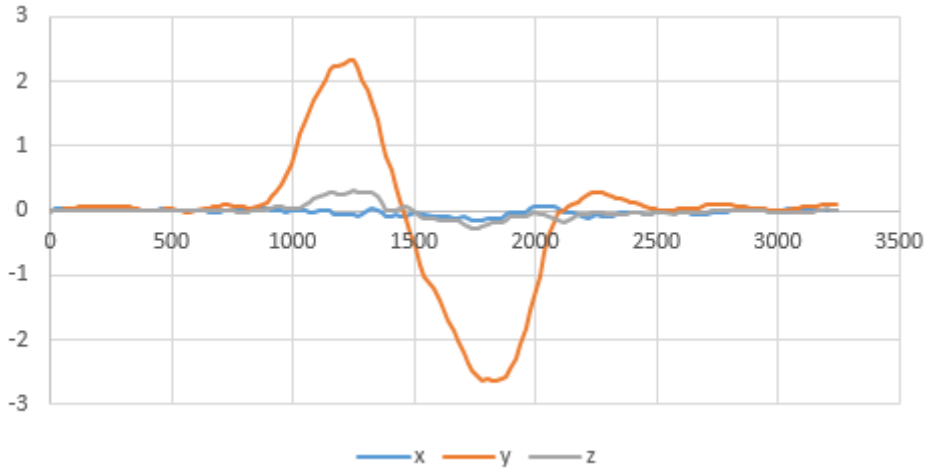
Grafik pada Gambar 3.8 terbentuk 1 buah bukit dan lembah pada nilai y. Bukit yang terjadi disini menunjukkan ketika pengguna menggerakan kepalanya kebawah, dan ketika kepala pengguna kembali ke posisi semula percepatan angularnya berbalik arah sehingga menimbulkan lembah. Nilai x dengan z cenderung bernilai 0. Gambar 3.9 menunjukkan nilai-nilai sensor *gyroscope* ketika pengguna sedang mengangguk dan menghadap ke atas.

Grafik pada Gambar 3.8 mirip seperti grafik pada Gambar 3.9. Begitu pula pada grafik di Gambar 3.10 yang memiliki pola yang serupa dengan yang grafik-grafik sebelumnya.

Pada grafik di Gambar 3.11 nilai yang mengalami kenaikan dan penurunan adalah nilai x dan nilai-nilai lainnya cenderung berada pada nilai 0. Nilai x membentuk 2 buah bukit dan 1 buah lembah. Bukit pertama terjadi ketika pengguna menggerakan kepalanya ke kiri. Lembah pertama terjadi ketika pengguna menggerakan kepalanya ke kanan. Bukit kedua terjadi ketika pengguna menggerakan kepalanya kembali ke posisi semula.

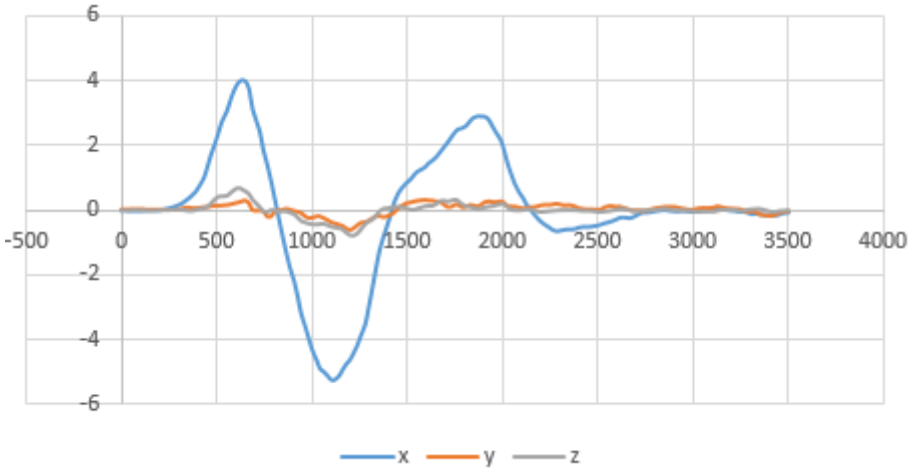
Pada grafik di Gambar 3.12 dengan grafik di Gambar 3.13 menunjukkan pola grafik yang serupa dengan grafik pada Gambar 3.11.

Dari hasil-hasil tersebut dapat disimpulkan bahwa arah pandang pengguna tidak mempengaruhi sensor *gyroscope* dalam mendeteksi gerakan kepala. Nilai-nilai yang dikembalikan oleh sensor

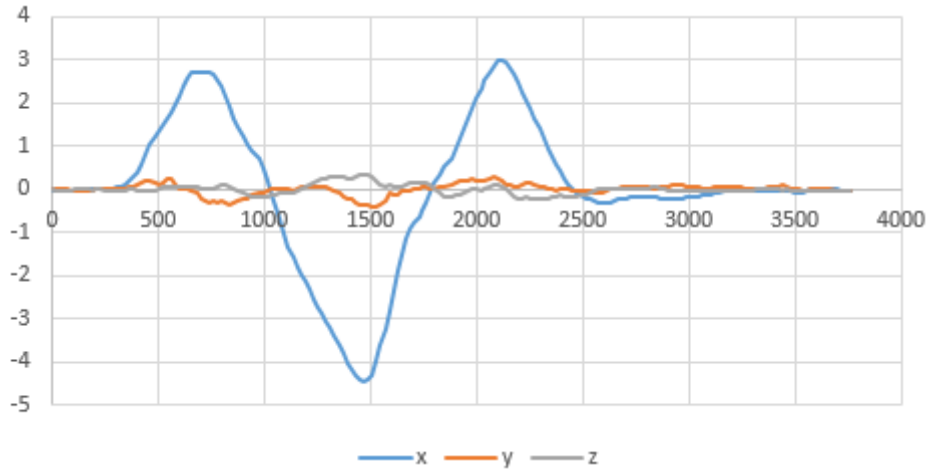


Gambar 3.9: Gambar grafik nilai sensor *gyroscope* ketika pengguna mengangguk dan sedang menghadap ke atas.

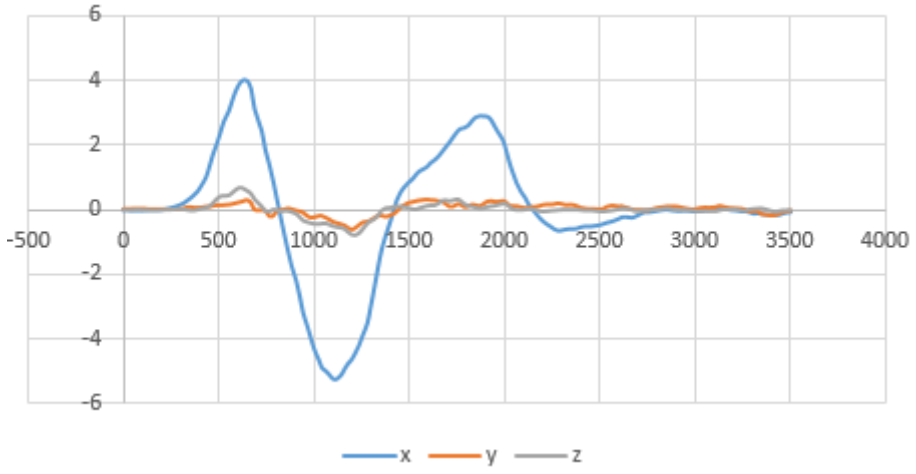
Gambar 3.10: Gambar grafik nilai sensor *gyroscope* ketika pengguna mengangguk dan sedang menghadap ke kiri atas.



Gambar 3.11: Gambar grafik nilai sensor *gyroscope* ketika pengguna menggeleng dan sedang menghadap ke depan.



Gambar 3.12: Gambar grafik nilai sensor *gyroscope* ketika pengguna menggeleng dan sedang menghadap ke atas.

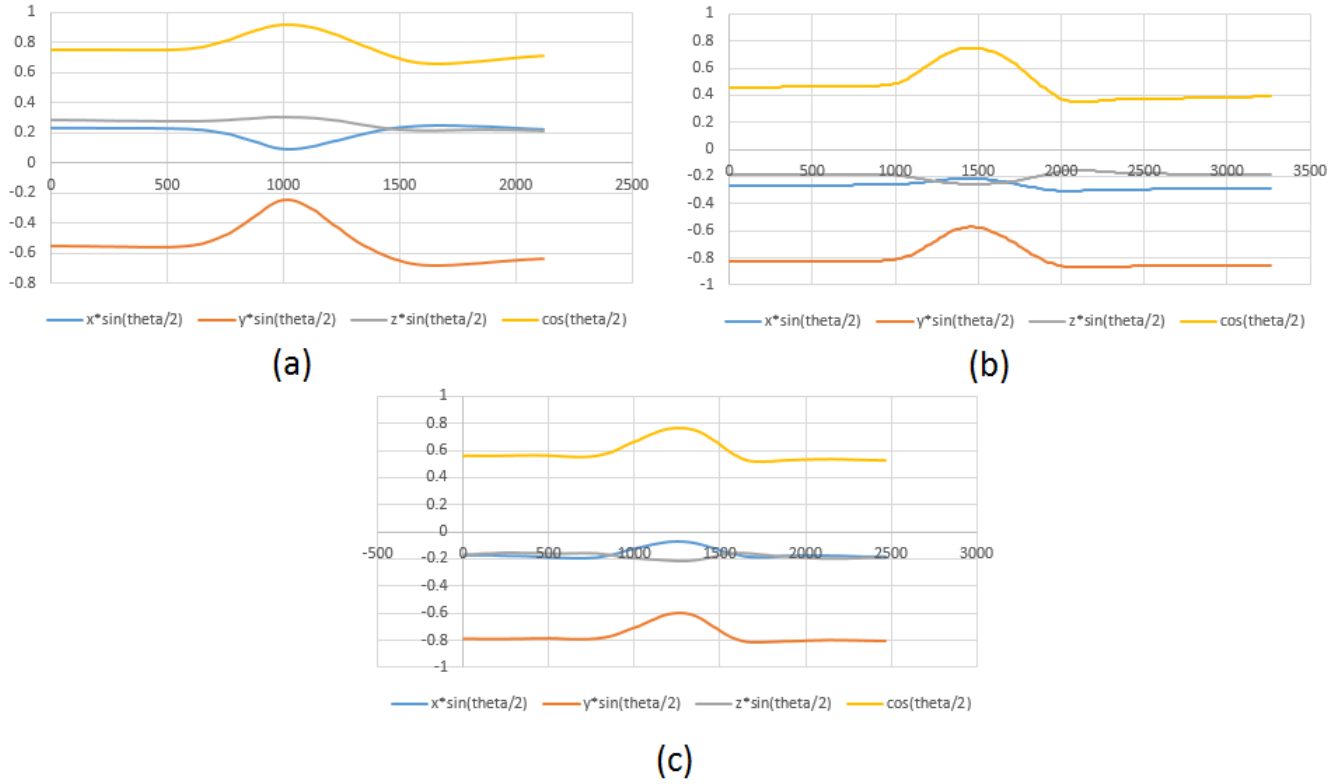


Gambar 3.13: Gambar grafik nilai sensor *gyroscope* ketika pengguna menggeleng dan sedang menghadap ke kiri atas.

gyroscope memiliki pola grafik yang jauh lebih rapih dibandingkan grafik-grafik yang dihasilkan oleh sensor *accelerometer*. Selain itu sensor gyroscope hanya menggunakan 1 jenis nilai yang dipengaruhi oleh pergerakan kepala, sedangkan accelerometer ada 2 jenis nilai yang di pengaruhi gerakan kepala pada saat mengangguk. Oleh karena itu sensor gryoscope lebih baik dalam mendeteksi gerakan yang terjadi pada perangkat android.

3.1.3 Analisis Grafik Sensor *Rotation Vector*

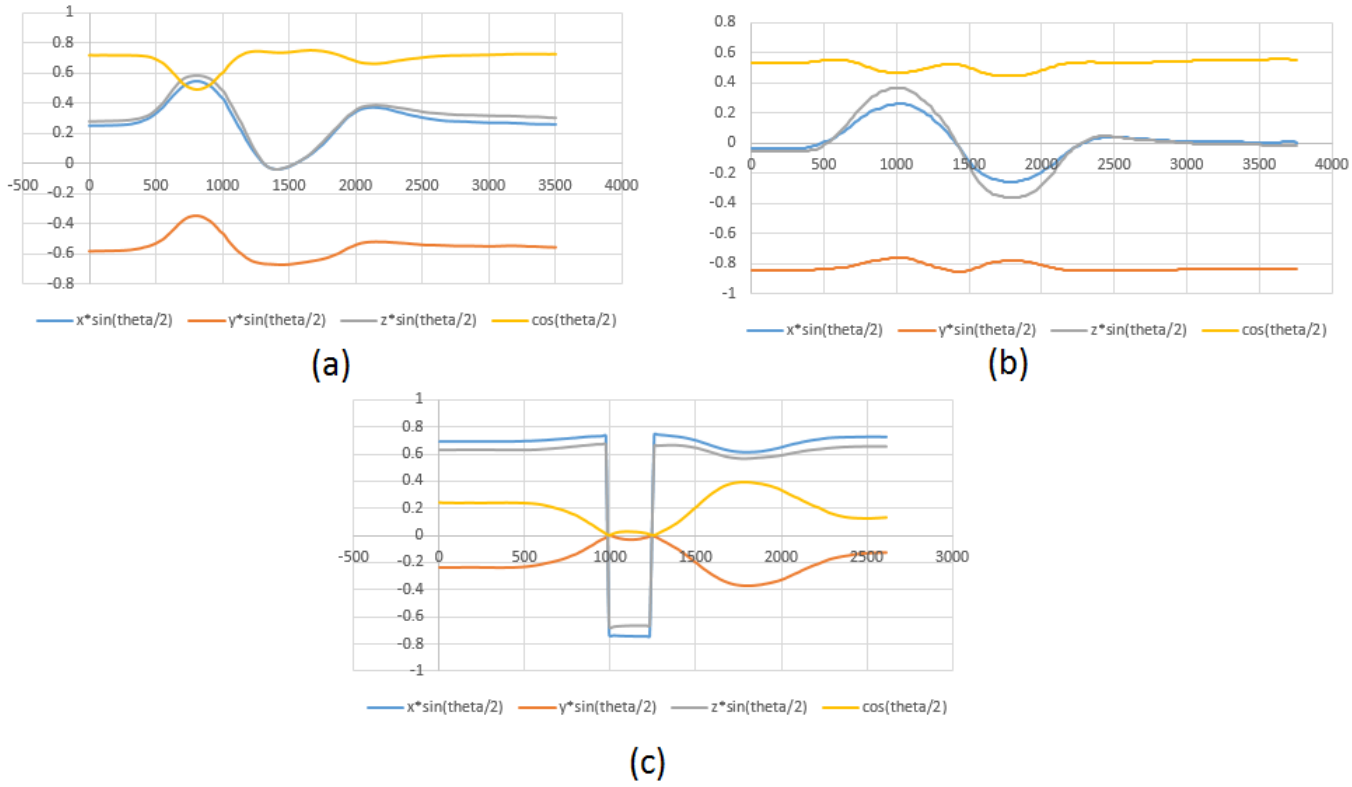
Perekaman menggunakan sensor *rotation vector* akan mendapatkan sebuah kuaternion yang merepresentasikan perputaran yang terjadi pada perangkat Android. Perputaran ini akan dideskripsikan dengan suatu vektor sebagai sumbu putarnya dan sudut perputarannya. Berbeda dengan sensor *gyroscope* yang merekam kecepatan perputaran yang terjadi pada suatu waktu, sensor *rotation vector* akan mengembalikan nilai kuaternion untuk mendefinisikan suatu kondisi putaran pada saat itu.



Gambar 3.14: Grafik nilai kuaternion dari sensor *rotation vector* ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

Gambar 3.14 menunjukkan nilai-nilai sensor *rotation vector* ketika pengguna sedang mengangguk. Bagian (a) pada Gambar 3.14 menunjukkan grafik nilai-nilai kuaternion dengan posisi muka awal menghadap ke depan. Pada grafik ini terlihat terbentuk sebuah bukit pada garis berwarna jingga dengan kuning, dan lembah pada garis berwarna biru. Garis yang berwarna abu cenderung stabil di angka 0.3. Bagian (b) pada Gambar 3.14 menunjukkan grafik nilai-nilai kuaternion dengan posisi muka awal menghadap ke atas. Grafik ini menunjukkan pola yang mirip pada bagian (a) namun berbeda nilainya saja. Pada bagian (a) garis berwarna kuning dimulai pada angka sekitar

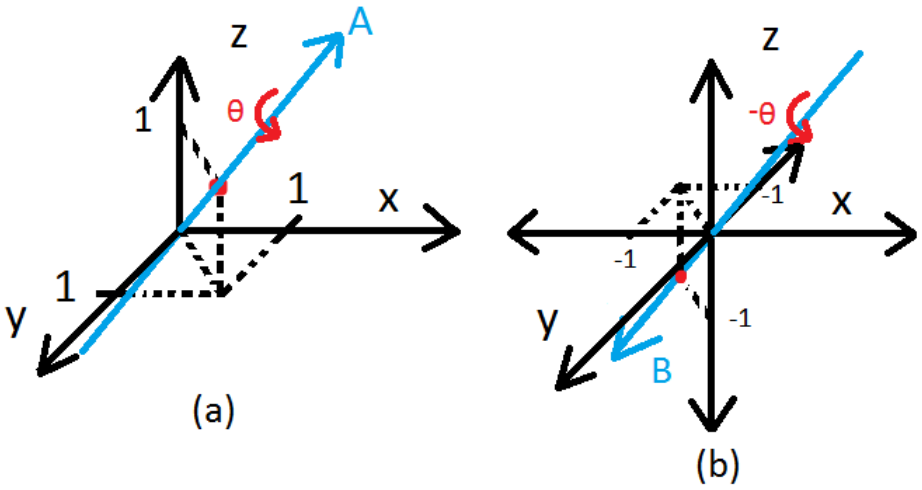
0.7, sedangkan pada bagian (b) garis berwarna jingga dimulai pada angka sekitar 0.4. Garis biru pada grafik ini tidak membentuk sebuat lembah seperti pada grafik pada bagian (a). Garis berwarna abu cenderung konstan pada nilai -0.2, sedangkan pada bagian (a) cenderung konstan di sekitar 0.3. Garis berwarna jingga memiliki pola yang sama dengan garis berwarna kuning, hanya berbeda pada nilainya saja. Bagian (c) pada Gambar 3.14 menunjukkan grafik nilai-nilai kuaternion dengan posisi muka awal menghadak ke kiri atas. Seperti pada grafik-grafik sebelumnya, grafik pada bagian (c) ini memiliki pola yang sama dengan grafik lainnya. Grafik ini juga hanya nilainya saja yang berbeda dengan grafik lainnya. Kemiripan pola ini memungkinkan mempermudah pendeteksian gerakan kepala.



Gambar 3.15: Grafik nilai kuaternion dari sensor *rotation vector* ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

Gambar 3.15 menunjukkan nilai-nilai sensor *rotation vector* ketika pengguna sedang menggeleng. Bagian (a) menunjukkan nilai-nilai kuaternion ketika posisi muka awal menghadap ke depan. Pada grafik ini terjadi bukit yang diikuti dengan lembah pada garis berwarna biru dengan abu-abu. Garis berwarna kuning membentuk suatu lembah yang setelahnya cenderung konstan. Berbeda pada garis kuning yang membentuk bukit kemudian setelahnya cenderung konstan. Grafik bagian (b) menunjukkan nilai-nilai kuaternion ketika posisi muka awal menghadap ke atas. Mirip dengan grafik pada bagian (a), garis biru dengan garis abu-abu memiliki pola yang sama yaitu membentuk bukit dengan lembah ketika pengguna menggelengkan kepala. Garis kuning dengan jingga cenderung konstan, berbeda dengan grafik pada bagian (a) yang membentuk bukit atau lembah. Pada bagian (c) garis-garis membentuk suatu pola yang tidak normal. Garis kuning membentuk sebuah lembah, tetapi membentuk suatu bukit kecil ketika nilainya mencapai nilai 0 pada milidetik ke 1000. Garis

jingga memiliki pola yang berlawanan dengan garis kuning. Pada saat garis kuning dengan garis jingga mencapai angka 0 perubahan drastis pun terjadi pada garis biru dengan abu-abu. Pada milidetik ke 1000 garis biru dengan abu mengalami perubahan nilai yang sangat drastis. Kedua nilai tersebut berubah dari nilai yang berkisar diantara 0.6 sampai 0.7 menjadi berkisar diantara -0.7 sampai -0.8. Kasus ini tidak berarti sensor sedang mengalami kegagalan akurasi (*accuracy fail*), tetapi memang seperti itulah karakteristik dari perputaran menggunakan kuaternion pada android. Perputaran yang terjadi pada batas mendekati sebelum terjadinya dengan setelah perubahan nilai yang drastis memiliki hasil perputaran yang sama. Hal ini disebabkan karena melakukan perputaran dengan vektor sebagai sumbu dapat memiliki 2 buah nilai yang sejenis. Dua buah nilai tersebut akan menghasilkan perputaran yang sama ketika arah vektor dengan arah putarnya di balikkan seperti yang ditunjukkan pada Gambar 3.16. Sepertinya pada sistem android nilai cos($theta/2$) didesain agar tidak bernilai negatif, sehingga nilai-nilai yang lainnya akan mengalami perubahan nilai yang drastis ketika nilai cos($theta/2$) mencapai angka 0.



Gambar 3.16: Dua buah rotasi yang identik dengan nilai kuaternion yang berbeda.

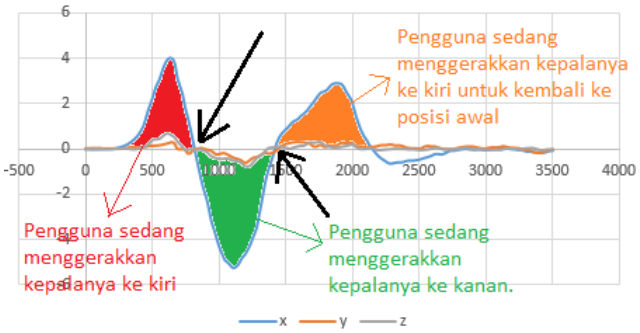
3.2 Analisis Data Sensor untuk Mendeteksi Gerakan Kepala

Dari ketiga hasil pecobaan pada sensor-sensor pada bab 3.1 dapat disimpulkan bahwa sensor *gyroscope* adalah sensor yang terbaik untuk mendeteksi gerakan kepala. Data dari sensor *accelerometer* akan susah untuk digunakan dalam mendeteksi gerakan kepala karena terganggu dengan aktivitas-aktivitas diluar gerakan pengguna yang juga ikut terekam oleh sensor *accelerometer*. Data dari sensor *rotation vector* juga akan lebih rumit dibandingkan sensor *gyroscope*. Hal ini karena perputaran yang di rekam oleh sensor *rotation vector* merekam kondisi putar pada suatu saat. Hasil rekaman ini akan mempersulit pada saat pendeteksian gerakan kepala karena harus melakukan proses untuk menghitung kecepatan kepala bergerak, agar dapat membedakan gerakan mengangguk atau menggeleng atau sekedar menoleh biasa. Dalam mendeteksi gerakan mengangguk dengan menggeleng banyak batas-batas yang perlu diperhatikan. Batas-batas tersebut untuk mengetahui apakah pengguna benar-benar mengangguk atau menggeleng atau sekedar menoleh biasa. Menurut pengamatan saya, berikut adalah batas-batas yang perlu diperhatikan :

- Kecepatan pengguna menoleh.
- Jarak waktu pada saat pengguna melakukan melawan arah gerakan.
- Simpangan terbesar kepala saat mengangguk atau menggeleng.

Kecepatan pengguna menjadi batas karena gerakan kepala yang kecepatannya cenderung pelan biasanya bukan merupakan gerakan mengangguk ataupun menggeleng. Kecepatan ini dapat langsung diperoleh menggunakan sensor *gyroscope*. Kecepatan yang dibutuhkan adalah kecepatan perputaran maksimum yang dilakukan oleh pengguna. Kecepatan maksimum dapat diperoleh dengan mengambil nilai puncak tertinggi dengan terendah. Jika nilai puncaknya mencapai kecepatan tertentu, gerakan tersebut dapat diperkirakan merupakan gerakan mengangguk ataupun menggeleng.

Gerakan menggeleng atau mengangguk biasanya memiliki selang waktu yang sangat sempit, karena pengguna biasanya langsung melawan arah secara langsung ketika sedang mengangguk ataupun menggeleng. Nilai ini dapat diperoleh dengan menghitung jarak antara bukit dengan lembah yang terbentuk pada grafik seperti yang dijelaskan pada Gambar 3.17. Idealnya gerakan menggeleng tidak memiliki rentang waktu ini, namun mungkin sebagian orang masih menghasilkan rentang waktu yang cukup sedikit. Oleh karena itu mungkin batas waktu yang ditentukan di sini adalah sekitar 100 milidetik. Jika rentang waktunya melebihi batas waktu tersebut, maka gerakan tersebut mungkin bukan merupakan gerakan mengangguk ataupun gerakan menggeleng.



Gambar 3.17: Deskripsi grafik pada saat pengguna menggeleng. Yang ditunjuk oleh panah berwarna hitam adalah selang waktu yang terjadi saat pengguna melawan arah gerakan kepala.

Simpangan terbesar ini juga penting untuk dijadikan batasan-batasan dalam mendeteksi gerakan mengangguk ataupun menggeleng. Simpangan kepala yang sangat kecil dapat diragukan untuk dianggap sebagai gerakan mengangguk atau menggeleng. Simpangan ini dapat diperoleh dengan menghitung luas yang dibentuk dari bukit atau lembah yang terbentuk pada grafik. Contoh pada Gambar 3.17 simpangan terbesar yang terjadi ketika pengguna menggerakkan kepalanya pertama kali ke kiri adalah luas pada bidang yang diarsir merah. Simpangan terbesar yang terjadi ketika pengguna menggerakaan kepalanya ke kanan adalah luas bidang yang diarsir berwarna hijau dikurangi dengan luas pada bidang yang diarsir merah. Pengurangan ini dilakukan karena luas bidang yang diarsir berwarna hijau merupakan simpangan yang terjadi setelah kepala sudah menghadap ke kiri. Begitu pula dengan luas bidang yang diarsir berwarna jingga yang akan dikurangi dengan hasil pengurangan luas sebelumnya.

DAFTAR REFERENSI

[1] T. Parisi, *Learning virtual reality: developing immersive experiences and applications for desktop, web, and mobile*. O'Reilly Media, 1 ed., 2015.

[2] G. J. Kim, *Designing virtual reality systems: the structured approach*. Springer, 2005.

[3] J. Vince, *Introduction to virtual reality*. Springer, 2004.

[4] "Google cardboard." https://vr.google.com/cardboard/. [Online; diakses 10-September-2016].

[5] "Sensor types." https://source.android.com/devices/sensors/sensor-types.html. [Online; diakses 10-September-2016].

[6] G. Bleser and D. Stricker, "Advanced tracking through efficient image processing and visual–inertial sensor fusion," *Computers & Graphics*, vol. 33, no. 1, pp. 59–72, 2009.

[7] A. Developers, "What is android," 2011.

[8] "Android 7.0 nougat!." https://developer.android.com/. [Online; diakses 12-September-2016].

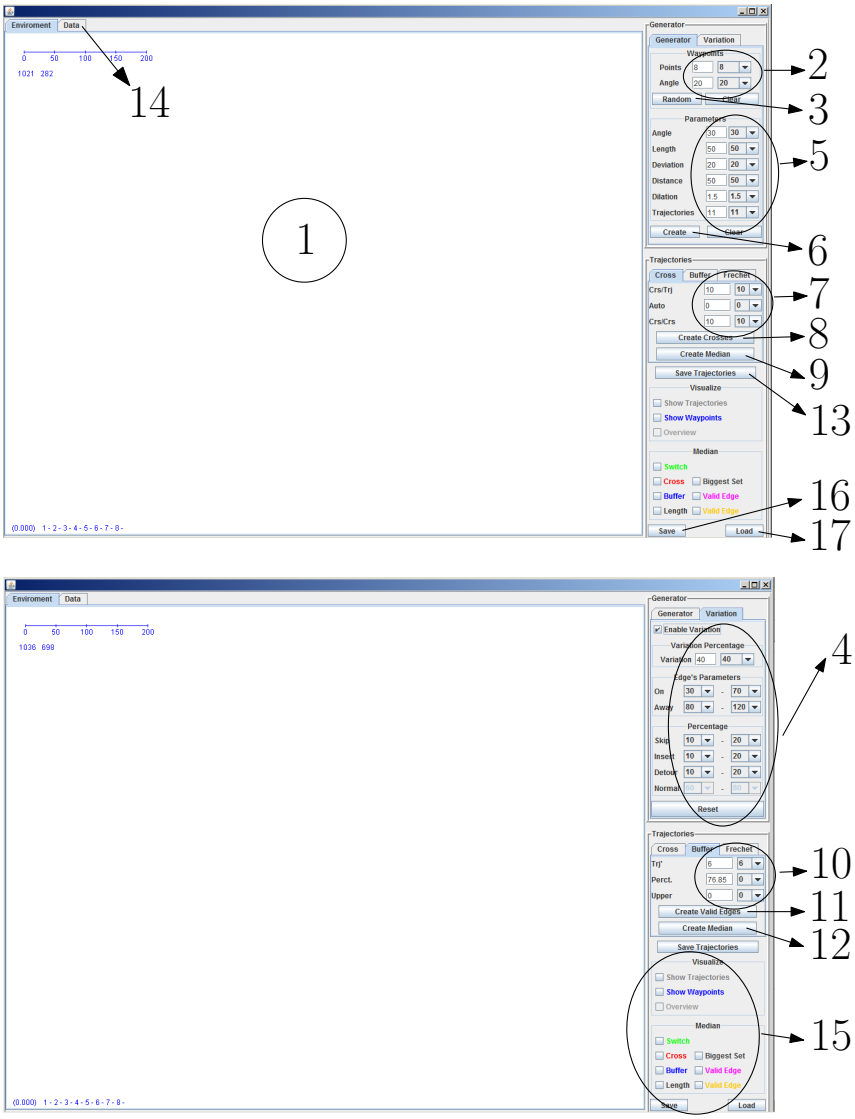
[9] "Google vr | google developers." https://developers.google.com/vr/. [Online; diakses 20-September-2016].

[10] J. B. Kuipers, *Quaternions and Rotation Sequences : A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 1998.

LAMPIRAN A

THE PROGRAM

The interface of the program is shown in Figure A.1:



Gambar A.1: Interface of the program

Step by step to compute the median trajectory using the program:

1. Create several waypoints. Click anywhere in the "Environment" area(1) or create them automatically by setting the parameters for waypoint(2) or clicking the button "Random"(3).

2. The "Variation" tab could be used to create variations by providing values needed to make them(4).

3. Create a set of trajectories by setting all parameters(5) and clicking the button "Create"(6).

4. Compute the median using the homotopic algorithm:

   - Define all parameters needed for the homotopic algorithm(7).
   - Create crosses by clicking the "Create Crosses" button(8).
   - Compute the median by clicking the "Compute Median" button(9).

5. Compute the median using the switching method and the buffer algorithm:

   - Define all parameters needed for the buffer algorithm(10).
   - Create valid edges by clicking the "Create Valid Edges"button(11).
   - Compute the median by clicking the "Compute Median"button(12).

6. Save the resulting median by clicking the "Save Trajectories" button(13). The result is saved in the computer memory and can be seen in "Data" tab(14)

7. The set of trajectories and its median trajectories will appear in the "Environment" area(1) and the user can change what to display by selecting various choices in "Visualize" and "Median" area(15).

8. To save all data to the disk, click the "Save"(16) button. A file dialog menu will appear.

9. To load data from the disk, click the "Load"(17) button.

LAMPIRAN B

THE SOURCE CODE

Listing B.1: MainActivity.java

```
1  package com.example.egaprianto.testingsensors;
2
3  import android.content.Intent;
4  import android.hardware.Sensor;
5  import android.hardware.SensorManager;
6  import android.support.v7.app.AppCompatActivity;
7  import android.os.Bundle;
8  import android.view.View;
9  import android.widget.TextView;
10
11 public class MainActivity extends AppCompatActivity {
12     TextView mTextView;
13     private SensorManager mSensorManager;
14     private Sensor mSensor;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_main);
20     }
21
22     @Override
23     public void onDestroy() {
24         super.onDestroy();
25         android.os.Debug.stopMethodTracing();
26     }
27
28     public void onClickLightSensorButton(View view){
29         Intent intent = new Intent(this, SensorAccelerometerActivity.class);
30         startActivity(intent);
31     }
32
33     public void onRotVecSensorButton(View view){
34         Intent intent = new Intent(this, QuaternionTheta.class);
35         startActivity(intent);
36     }
37
38     public void recordAllRAWSensorData(View view){
39         Intent intent = new Intent(this, RecordAllRAWSensorData.class);
40         startActivity(intent);
41     }
42
43     public void onGyroSensorClicked(View view){
44         Intent intent = new Intent(this, SensorGyroscopeActivity.class);
45         startActivity(intent);
46     }
47     public void onMagSensorClicked(View view){
48         Intent intent = new Intent(this, SensorGeomagnetoticRotationActivity.class);
49         startActivity(intent);
50     }
51 }
```

Listing B.2: RecordAllRAWSensorData.java

```
1  package com.example.egaprianto.testingsensors;
2
3  import android.Manifest;
4  import android.content.Context;
5  import android.content.pm.PackageManager;
6  import android.hardware.Sensor;
7  import android.hardware.SensorEvent;
8  import android.hardware.SensorEventListener;
9  import android.hardware.SensorManager;
10 import android.media.Ringtone;
11 import android.media.RingtoneManager;
12 import android.net.Uri;
13 import android.os.Build;
14 import android.os.Bundle;
15 import android.os.CountDownTimer;
16 import android.os.Environment;
```

```
17 import android.support.v4.app.ActivityCompat;
18 import android.support.v7.app.AppCompatActivity;
19 import android.text.format.DateFormat;
20 import android.util.Log;
21 import android.view.View;
22 import android.view.WindowManager;
23 import android.widget.TextView;
24
25
26 import java.io.BufferedWriter;
27 import java.io.File;
28 import java.io.FileWriter;
29 import java.io.IOException;
30 import java.util.ArrayList;
31 import java.util.List;
32
33 public class RecordAllRAWSensorData extends AppCompatActivity implements SensorEventListener {
34     private static final String FILENAME = "sensorRecord-";
35     private SensorManager mSensorManager;
36     private ArrayList<Sensor> sensorArrayList;
37     private TextView textViewCountDown;
38     private boolean isCapturing;
39     private File file;
40     private ArrayList<double[]> acceleroData;
41     private ArrayList<double[]> gyroData;
42     private long startCaptureTime;
43     private long runningTime;
44     private ArrayList<double[]> rotData;
45     private ArrayList<double[]> rotVecData;
46
47     @Override
48     protected void onCreate(Bundle savedInstanceState) {
49         super.onCreate(savedInstanceState);
50         this.sensorArrayList = new ArrayList<Sensor>();
51         setContentView(R.layout.activity_record_all_rawsensor_data);
52         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
53         textViewCountDown = (TextView) findViewById(R.id.textViewCountDown);
54         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
55             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_ACCELEROMETER));
56             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_GYROSCOPE));
57             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_ROTATION_VECTOR));
58         }
59
60         int REQUEST_EXTERNAL_STORAGE = 1;
61         String[] PERMISSIONS_STORAGE = {
62                 Manifest.permission.READ_EXTERNAL_STORAGE,
63                 Manifest.permission.WRITE_EXTERNAL_STORAGE
64         };
65         int permission = ActivityCompat.checkSelfPermission(this, Manifest.permission.
               WRITE_EXTERNAL_STORAGE);
66         if (permission != PackageManager.PERMISSION_GRANTED) {
67             ActivityCompat.requestPermissions(
68                     this,
69                     PERMISSIONS_STORAGE,
70                     REQUEST_EXTERNAL_STORAGE
71             );
72         }
73         isCapturing = false;
74         for (Sensor sensor :
75                 sensorArrayList) {
76             mSensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
77         }
78     }
79
80     @Override
81     protected void onResume() {
82         super.onResume();
83         for (Sensor sensor :
84                 sensorArrayList) {
85             mSensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
86         }
87     }
88
89     @Override
90     protected void onPause() {
91         super.onPause();
92         mSensorManager.unregisterListener(this);
93     }
94
95     @Override
96     public void onSensorChanged(SensorEvent event) {
97         Sensor sensor = event.sensor;
98         if (isCapturing) {
99             float[] copyData = new float[4];
100            System.arraycopy(event.values, 0, copyData, 1, 3);
101            copyData[0] = System.currentTimeMillis() - startCaptureTime;
102            if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
103                this.acceleroData.add(convertFloatsToDoubles(copyData));
104            } else if (sensor.getType() == Sensor.TYPE_GYROSCOPE) {
105                this.gyroData.add(convertFloatsToDoubles(copyData));
106            }else if (sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {
107                copyData = new float[6];
108                System.arraycopy(event.values, 0, copyData, 1, event.values.length);
109                copyData[0] = System.currentTimeMillis() - startCaptureTime;
110                this.rotData.add(convertFloatsToDoubles(copyData));
111                double theta = (Math.acos(event.values[3]))*2;
112                double x = event.values[0]/ Math.sin(theta/2);
113                double y = event.values[1]/ Math.sin(theta/2);
114                double z = event.values[2]/ Math.sin(theta/2);
```

```
115                 double [] vecData = new double[5];
116                 vecData[0] = copyData[0];
117                 vecData[1] = x;
118                 vecData[2] = y;
119                 vecData[3] = z;
120                 vecData[4] = theta;
121                 this.rotVecData.add(vecData);
122             }
123             runningTime = (System.currentTimeMillis() - startCaptureTime);
124             this.textViewCountDown.setText("Time : " + runningTime / 1000 + "sec");
125         }
126     }
127
128     @Override
129     public void onAccuracyChanged(Sensor sensor, int accuracy) {
130
131     }
132
133     public void onStartButtonClicked(View view) throws InterruptedException {
134         Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
135         Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), notification);
136         acceleroData = new ArrayList<double[]>();
137         gyroData = new ArrayList<double[]>();
138         rotData = new ArrayList<double[]>();
139         rotVecData = new ArrayList<double[]>();
140         String time = DateFormat.format("MM-dd-yyyy-h|mm|ss-aa", System.currentTimeMillis()).toString();
141         File root = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS
                ) + File.separator + "DataSensors");
142         root.mkdirs();
143         file = new File(root, FILENAME + time + ".csv");
144         try {
145             file.createNewFile();
146         } catch (IOException e) {
147             e.printStackTrace();
148         }
149         new CountDownTimer(10000, 1000) {
150
151             public void onTick(long millisUntilFinished) {
152                 textViewCountDown.setText("Count Down = " + millisUntilFinished / 1000);
153             }
154
155             public void onFinish() {
156                 textViewCountDown.setText("Capturing Data!");
157                 Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
158                 Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), notification);
159                 r.play();
160                 isCapturing = true;
161                 getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
162                 startCaptureTime = System.currentTimeMillis();
163             }
164         }.start();
165
166     }
167
168     private Sensor sensorGetter(int input) {
169
170         List<Sensor> sensorList = mSensorManager.getSensorList(input);
171         Sensor sensor = null;
172         for (int i = 0; i < sensorList.size(); i++) {
173             sensor = sensorList.get(i);
174         }
175         return sensor;
176     }
177
178     private void writeData(BufferedWriter bw, ArrayList<double[]> data, String title, String columnTitle)
            throws IOException {
179         bw.write(title);
180         bw.newLine();
181         bw.write(columnTitle);
182         bw.newLine();
183         for (int i = 0; i < data.size(); i++) {
184             for (int j = 0; j < data.get(i).length; j++) {
185                 bw.write(data.get(i)[j] + ",");
186             }
187             bw.newLine();
188         }
189         bw.newLine();
190     }
191
192     private double[] convertFloatsToDoubles(float[] input)
193     {
194         if (input == null)
195         {
196             return null;
197         }
198         double[] output = new double[input.length];
199         for (int i = 0; i < input.length; i++)
200         {
201             output[i] = input[i];
202         }
203         return output;
204     }
205
206     public void onStopButtonClicked(View view) {
207         isCapturing = false;
208         getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
209         try {
210             BufferedWriter bw = new BufferedWriter(new FileWriter(this.file));
211             writeData(bw, acceleroData, "Accelerometer", "time,x,y,z");
```

```
212             writeData(bw, gyroData, "Gyroscope", "time,x,y,z");
213             writeData(bw, rotData, "Rotation Vector", "time,x*sin(theta/2),y*sin(theta/2),z*sin(theta/2),
                    cos(theta/2),akurasi (dalam radians) (-1 jika tidak ada)");
214             writeData(bw, rotVecData, "Rotation Vector by Vector and Angle", "time,x,y,z,theta");
215             bw.newLine();
216             bw.write("Running Time : " + runningTime + "ms");
217             bw.flush();
218             bw.close();
219             file.createNewFile();
220             textViewCountDown.setText("Data Captured! at " + file.getAbsolutePath());
221             Log.d("File", "File is located at " + file.getAbsolutePath());
222         } catch (IOException e) {
223             e.printStackTrace();
224         }
225
226     }
227 }
```

Listing B.3: SensorAccelerometerActivity.java

```
1  package com.example.egaprianto.testingsensors;
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.hardware.Sensor;
6  import android.hardware.SensorEvent;
7  import android.hardware.SensorEventListener;
8  import android.hardware.SensorManager;
9  import android.os.Build;
10 import android.os.Bundle;
11 import android.widget.TextView;
12
13 import java.util.List;
14
15 public class SensorAccelerometerActivity extends Activity implements SensorEventListener {
16     private SensorManager mSensorManager;
17     private Sensor mAccelSensor;
18     private TextView textViewXData;
19     private TextView textViewYData;
20     private TextView textViewZData;
21     private TextView textViewDebug;
22     private TextView textViewAccuracy;
23
24     @Override
25     public final void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_accelero_sensor);
28         textViewXData = (TextView) findViewById(R.id.textViewXData);
29         textViewYData = (TextView) findViewById(R.id.textViewYData);
30         textViewZData = (TextView) findViewById(R.id.textViewZData);
31         textViewDebug = (TextView) findViewById(R.id.textViewDebug);
32         textViewAccuracy = (TextView) findViewById(R.id.textViewAccuracy);
33         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
34         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
35             mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
36             List<Sensor> accelerometerSensors = mSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
37             mAccelSensor = null;
38             if (accelerometerSensors != null) {
39                 for (int i = 0; i < accelerometerSensors.size(); i++) {
40                     mAccelSensor = accelerometerSensors.get(i);
41                 }
42             }
43         }
44     }
45
46     @Override
47     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
48     }
49
50     @Override
51     public final void onSensorChanged(SensorEvent event) {
52         float x = event.values[0];
53         float y = event.values[1];
54         float z = event.values[2];
55         textViewXData.setText("x =" + x);
56         textViewYData.setText("y =" + y);
57         textViewZData.setText("z =" + z);
58     }
59
60     @Override
61     protected void onResume() {
62         super.onResume();
63         mSensorManager.registerListener(this, mAccelSensor, SensorManager.SENSOR_DELAY_NORMAL);
64     }
65
66     @Override
67     protected void onPause() {
68         super.onPause();
69         mSensorManager.unregisterListener(this);
70     }
71
72
73 }
```

Listing B.4: SensorGyroscopeActivity.java

```
1  package com.example.egaprianto.testingsensors;
```

```
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.hardware.Sensor;
6  import android.hardware.SensorEvent;
7  import android.hardware.SensorEventListener;
8  import android.hardware.SensorManager;
9  import android.os.Build;
10 import android.os.Bundle;
11 import android.widget.TextView;
12
13 import java.util.List;
14
15 public class SensorGyroscopeActivity extends Activity implements SensorEventListener {
16     private SensorManager mSensorManager;
17     private Sensor mGryoSensor;
18     private TextView textViewXData;
19     private TextView textViewYData;
20     private TextView textViewZData;
21     private TextView textViewDebug;
22     private TextView textViewAccuracy;
23
24     @Override
25     public final void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_gyroscope_sensor);
28         textViewXData = (TextView) findViewById(R.id.textViewXData);
29         textViewYData = (TextView) findViewById(R.id.textViewYData);
30         textViewZData = (TextView) findViewById(R.id.textViewZData);
31         textViewDebug = (TextView) findViewById(R.id.textViewDebug);
32         textViewAccuracy = (TextView) findViewById(R.id.textViewAccuracy);
33         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
34         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
35             mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
36             List<Sensor> gyroscopeSensors = mSensorManager.getSensorList(Sensor.TYPE_GYROSCOPE);
37             mGryoSensor = null;
38             if (gyroscopeSensors != null) {
39                 for (int i = 0; i < gyroscopeSensors.size(); i++) {
40                     mGryoSensor = gyroscopeSensors.get(i);
41                 }
42             }
43         }
44     }
45
46     @Override
47     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
48     }
49
50     @Override
51     public final void onSensorChanged(SensorEvent event) {
52         float x = event.values[0];
53         float y = event.values[1];
54         float z = event.values[2];
55         textViewXData.setText("x = " + x);
56         textViewYData.setText("y = " + y);
57         textViewZData.setText("z = " + z);
58     }
59
60     @Override
61     protected void onResume() {
62         super.onResume();
63         mSensorManager.registerListener(this, mGryoSensor, SensorManager.SENSOR_DELAY_NORMAL);
64     }
65
66     @Override
67     protected void onPause() {
68         super.onPause();
69         mSensorManager.unregisterListener(this);
70     }
71
72
73 }
```