BAB 2 LANDASAN KEPUSTAKAAN

2.1 *Head Movement Control System* (HEMOCS)

Head Movement Control System (HEMOCS) adalah suatu sistem kendali yang memanfaatkan data dari sensor gerak yang terdapat dalam perangkat bergerak yang digunakan sebagai acuan dalam mendeteksi pergerakan kepala pengguna. Sistem ini digunakan sebagai alternatif dalam interaksi manusia dengan komputer bagi seseorang yang memiliki disabilitas pada anggota gerak tangan, yang ingin berinteraksi dengan sistem maupun aplikasi yang diprogram untuk bisa mengenali pergerakan kepala pengguna.

2.1.1 Kajian Pustaka

Penelitian pertama yang terkait adalah penelitian dengan judul "*Mobile Devices Based 3D Image Display Depending on User's Actions and Movements"* yang membahas tentang kontrol kamera pada citra gambar 3 dimensi melalui pergerakan dan aksi pengguna (Arai & Tolle, 2013). Kontrol kamera pada citra 3 dimensi ini memanfaatkan data dari *gyroscope* dan *accelerometer* yang diolah dengan *framework Core Motion.* Ketika kepala pengguna menghadap ke atas, maka citra 3 dimensi tadi akan menampilkan citra 3 dimensi di bagian atas, begitu seterusnya sesuai dengan posisi menghadap dari kepala pengguna.

Penelitian kedua dengan judul "*Design of Head Movement Controller System (HEMOCS) for Control Mobile Application through Head Pose Movement Detection*" yang membahas tentang pengembangan metode pendeteksian pergerakan kepala dengan sensor internal pada perangkat iOS, yang diolah dengan menggunakan *framework Core Motion.* Pendeteksian gerakan kepala memanfaatkan pola data dari sensor gerak yang dapat direpresentasikan melalui 3 sumbu, yaitu sumbu *pitch,* sumbu *roll,* dan sumbu *yaw* (Tolle & Arai, 2016). Sumbu *yaw* digunakan sebagai acuan dalam mengenali pergerakan kepala secara horizontal ke kiri maupun ke kanan. Sumbu *roll* digunakan sebagai acuan dalam mengenali pergerakan kepala secara vertikal ke atas maupun ke bawah, sedangkan sumbu *pitch* digunakan sebagai acuan dalam mengenali pergerakan kepala secara miring ke kiri maupun ke kanan (Safii, Tolle, & Kharisma, 2016).

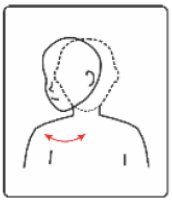
Gambar 2.1 *Degrees of Freedom* pada *Head Mounted Display*

Sumber: Tolle & Arai, 2016

Ada 6 jenis pergerakan kepala yang digunakan sebagai kontrol dalam sistem ini, diantaranya:

a. Menoleh ke kanan

Pada gerakan ini, pengguna menoleh ke arah kanan, lalu kembali berputar secara horizontal ke titik awal (menghadap depan).

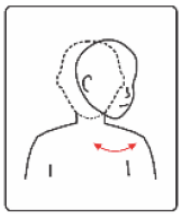


Gambar 2.2 Gerakan menoleh ke kanan

Sumber: Tolle & Arai, 2016

b. Menoleh ke kiri

Pada gerakan ini, pengguna menoleh ke arah kanan, lalu kembali berputar secara horizontal ke titik awal (menghadap depan).

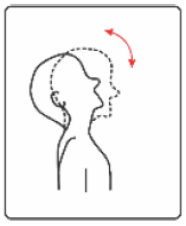


Gambar 2.3 Gerakan menoleh ke kiri

Sumber: Tolle & Arai, 2016

c. Melihat ke atas

Pada gerakan ini, pengguna melihat ke arah atas secara vertikal, lalu kembali berputar secara vertikal ke titik awal (menghadap depan).

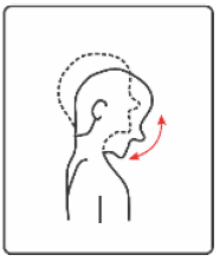


Gambar 2.4 Gerakan melihat ke atas

Sumber: Tolle & Arai, 2016

d. Melihat ke bawah

Pada gerakan ini, pengguna melihat ke arah bawah secara vertikal, lalu kembali berputar secara vertikal ke titik awal (menghadap depan).

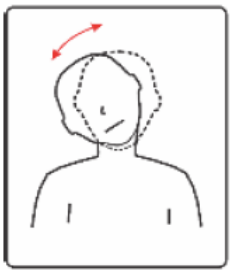


Gambar 2.5 Gerakan melihat ke bawah

Sumber: Tolle & Arai, 2016

e. Miring ke kanan

Pada gerakan ini, pengguna memiringkan kepala ke kanan, lalu menegakkan kepala ke titik awal.

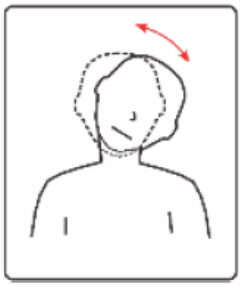


Gambar 2.6 Gerakan miring ke kanan

Sumber: Tolle & Arai, 2016

f. Miring ke kiri

Pada gerakan ini, pengguna memiringkan kepala ke kiri, lalu menegakkan kepala ke titik awal.



Gambar 2.7 Gerakan miring ke kiri

Sumber: Tolle & Arai, 2016

Penelitian terkait yang ketiga berjudul "*Design of Keyboard Input Control for Mobile Application using Head Movement Control Systems* (HEMOCS)" yang membahas tentang perancangan sebuah aplikasi *keyboard* yang masukannya berasal dari pergerakan kepala. Pada penelitian ini, pergerakan kepala pengguna berdasarkan pergerakan kepala linear yang memanfaatkan data yang diperoleh dari sensor berdasarkan sumbu *pitch, yaw,* dan *roll* (Tolle, Aknuranda, Ananta, Brata, & Az-Zahra, 2017). Ada 8 buah pergerakan kepala yang digunakan dalam penelitian ini, terdapat dalam tabel sebagai berikut:

Tabel 2.1 Tabel Pergerakan Kepala pada Desain Keyboard HEMOCS

Kode	**Tipe Gerakan Kepala**	**Fungsi**
H1	*Axial Rotation Left*	Pindah seleksi sebelumnya
H2	*Axial Rotation Right*	Pindah seleksi setelahnya
H3A	*Small Extension*	Pindah seleksi ke atas
H3B	*Extension*	Backspace
H4A	*Small Flexion*	Pindah seleksi ke bawah
H4B	*Flexion*	Memilih tombol
H5	*Lateral bending left*	Menghapus semua karakter
H6	*Lateral bending right*	Insert spasi

Sumber: Tolle et al., 2017

Penelitian terkait yang keempat berjudul "Pengembangan Aplikasi *Health Communication Board* Berbasis Kendali Pergerakan Linear HEMOCS (Head Movement Control System)" yang membahas tentang pengembangan aplikasi *Health Communication Board* yang digunakan oleh dokter kepada pasien dalam membantu mereka menyampaikan informasi yang diinginkan, yang dikembangkan untuk perangkat bergerak berbasis iOS, dan dikendalikan dengan pergerakan kepala, serta memadukan teknologi *Augmented Reality* dengan *Head Mounted Device* (Pratama, Tolle, & Az-zahra, 2017).

Penelitian kelima berjudul "*Real Time Head Pose Estimation for Mobile Device*" membahas tentang pengestimasian pose kepala yang menggunakan kamera depan atau belakang sebagai penangkap citra pose kepala yang nantinya akan diolah menggunakan *image processing*. Jika wajah terdeteksi pada citra yang telah diambil, maka citra akan dilakukan *preprocessing* dengan *illumination correction*, dan posisi tersebut dijadikan sebuah input dari sebuah *tracker* (Neto et al., 2013).

Penelitian terakhir, adalah yang berjudul "Implementasi Pengendalian Quadcopter Dengan Prinsip Virtual Reality Menggunakan Google Cardboard".

Penelitian ini membahas tentang pengendalian Parrot AR.Drone menggunakan pergerakan kepala dengan konsep Virtual Reality (Pribadi, Jonemaro, & Setyawan, 2017). Dalam pendeteksian pergerakan kepala, peneliti tersebut menggunakan sensor *gyroscope* yang datanya dihitung dengan menggunakan *quaternion* pada perangkat Android. Pada penelitian tersebut, terdapat 7 pergerakan yang digunakan, yaitu *takeoff, hover, landing, ascending, descending, rotate left,* dan *rotate right.*

Dari keenam penelitian tersebut, pengembangan aplikasi dengan menggunakan konsep *Head Movement Control System* menggunakan sensor berupa *accelerometer* dan *gyroscope* yang diolah sesuai dengan *platform* pengembangannya. Untuk *platform* iOS, pengolahan data tersebut sedikit lebih mudah karena dalam *Core Motion framework,* telah menyediakan data *device motion* yang telah diproses berdasarkan data *gyroscope* dan *accelerometer*, salah satunya data *device attitude* yang berisikan data *pitch, yaw, roll* yang nantinya akan dipetakan sesuai dengan pergerakan kepala yang ada. Seluruh informasi tadi akan dijadikan acuan dalam pengembangan aplikasi kendali kemudi kursi roda berbasis HEMOCS.

2.1.2 Algoritme *Head Movement Control System*

Pada sistem *Head Movement Control System* ini, terdapat algoritme yang digunakan sebagai *core* dalam aplikasi kendali kemudi kursi roda. Algoritme ini memiliki beberapa tahap, diantaranya:

1. Mendapatkan data dari sensor gerak

   Yang pertama dilakukan adalah mendapatkan data dari sensor gerak pada perangkat, lalu data diproses untuk digunakan di tahapan selanjutnya

2. Pengecekan *threshold* sudut

   Setelah mendapatkan data, lalu dilakukan pengecekan *threshold* sudut pegerakan pada sumbu-sumbu yang terkait. Jika nilai sudut melebihi atau sama dengan nilai *threshold* yang telah ditetapkan sebelumnya, maka algoritme akan melakukan penghitungan interval pergerakan kepala, jika tidak melebihi *threshold* yang ditentukan, maka algoritme akan menganggap tidak ada pergerakan kepala sama sekali.

3. Mendeteksi jenis sumbu dari sensor gerak

   Setelah mendapatkan data dari sensor gerak pada perangkat, maka dilakukan pengecekan perubahan data pada sumbu tertentu, apakah sumbu *pitch, yaw* atau *roll.* Jika sumbu *pitch* mengalami perubahan, maka pergerakan kepala yang terjadi adalah pergerakan miring ke kanan atau kiri. Jika sumbu *yaw* yang mengalami perubahan, maka pergerakan kepala yang terjadi adalah pergerakan menoleh ke kanan atau kiri. Jika sumbu *roll* yang mengalami perubahan, maka pergerakan kepala yang terjadi adalah pergerakan menghadap ke atas atau ke bawah.

4. Menghitung panjang interval pergerakan kepala

Setelah pergerakan kepala telah terdeteksi berdasarkan jenisnya, maka lakukan pengecekan panjang interval pergerakan kepala, pengecekan ini digunakan untuk melakukan validasi pergerakan, jika pergerakan tersebut masuk kedalam interval minimum dan maksimum, maka pergerakan dianggap valid.

5. Pemetaan pergerakan kepala

Setelah dilakukan validasi panjang interval pergerakan kepala, maka tahap selanjutnya adalah memetakan pergerakan kepala sesuai data yang telah didapat sebelumnya. Hasil pemetaan ini digunakan sebagai acuan sistem untuk menjalankan aksi yang sesuai.

6. Sistem menjalankan aksi sesuai hasil pemetaan pergerakan kepala

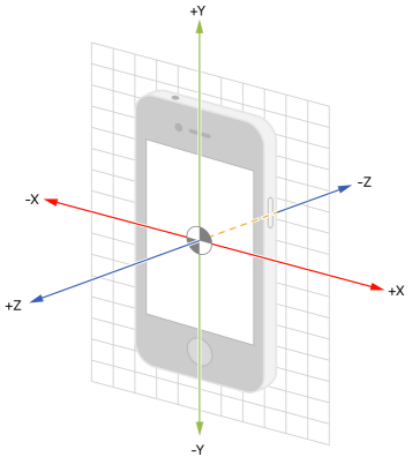
Setelah mendapatkan hasil pemetaan pergerakan kepala dari proses pengolahan data di tahap-tahap selanjutnya, maka sistem akan menjalankan aksi sesuai dengan hasil pemetaan sebelumnya, seperti contoh pergerakan kepala untuk memerintahkan kemudi untuk berputar ke kanan.

2.2 Sensor Gerakan Pada Perangkat Bergerak iOS

Pada perangkat bergerak berbasis iOS, terdapat dua buah sensor gerak, yaitu *accelerometer* dan *gyroscope.*

2.2.1 Accelerometer Sensor

Sensor *accelerometer* digunakan untuk mengukur percepatan pergerakan pada perangkat berdasarkan salah satu sumbu. Pada perangkat berbasis iOS, terdapat 3 buah sumbu yang menyediakan nilai dari *percepatan* pergerakan perangkat pada masing-masing sumbu (Apple, 2017).

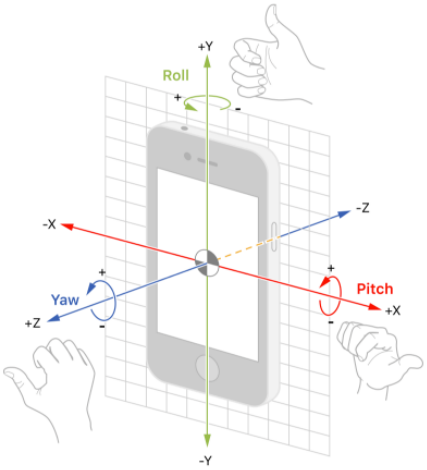


Gambar 2.8 Sumbu pada *accelerometer sensor*

Sumber: Core Motion Documentation, Apple

2.2.2 Gyroscope Sensor

Sensor *gyroscope* digunakan untuk mengukur *rotation rate* perangkat bergerak yang berputar pada sumbu spasial. Pada tiap perangkat iOS, terdapat 3 buah sumbu pada *gyroscope,* diantaranya adalah sumbu *pitch, yaw,* maupun *roll* (Apple 2017).



Gambar 2.9 Sumbu pada *gyroscope sensor*

Sumber: Core Motion Documentation, Apple

2.3 Pemrograman iOS Dengan Swift 3

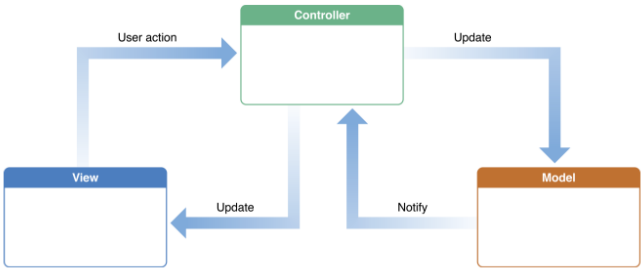
Swift adalah bahasa pemrograman baru untuk mengembangkan aplikasi iOS, macOS maupun tvOS yang dibangun dengan bahasa C dan Objective-C, tanpa batasan kompatibilitas dengan bahasa C. Swift mengadopsi pola *safe programming* dan menambahkan fitur-fitur baru yang membuat proses pemrograman jauh lebih mudah, lebih fleksibel dan lebih menyenangkan (Apple, 2017). Beberapa fitur dari Swift adalah:

- *Closure*
- Tipe data berupa *tuple* dan fungsi bisa mengembalikan *multiple return value*
- Mendukung adanya *Generic type*
- Dapat melakukan iterasi *range* dan *collections* dengan cepat
- *Struct* yang mendukung adanya *method, extension,* dan *protocol*
- Mendukung pola pemrograman fungsional
- Memiliki error handling yang mumpuni
- Memiliki *control flow* seperti *do*, *guard*, *defer*, dan *repeat*.

2.3.1 Arsitektur Aplikasi iOS

Secara *default,* arsitektur aplikasi yang digunakan dalam pengembangan iOS adalah arsitektur MVC (*Model-View-Controller).* Arsitektur ini memisahkan tanggung jawab antara *model* sebagai *layer* yang menangani data dan manipulasi data, *view* sebagai *layer* tampilan aplikasi atau antar muka pengguna (UI), dan *controller* sebagai penengah antara *view* dan *model,* yang berfungsi untuk mengubah model dan menginterpretasikan input atau aksi dari pengguna dari *view layer*.

Seiring berkembangnya waktu, banyak arsitektur yang dikembangkan oleh arsitektur ini, seperti arsitektur MVP maupun MVVM, serta arsitektur yang berdiri sendiri (tidak terpengaruh dengan varian arsitektur MV*) seperti VIPER.



Gambar 2.10 Diagram Arsitektur MVC

Sumber: Apple, 2017

2.3.2 *Application Lifecycle*

Aplikasi iOS tentunya memiliki daur hidup atau *lifecycle,* mulai dari inisialisasi aplikasi saat pertama kali dijalankan hingga selesai dieksekusi (*terminated*). Berikut ini tabel yang menjelaskan daur hidup dari aplikasi iOS.

Tabel 2.2 iOS *Application Lifecycle*

Daur Hidup	Penjelasan
application:willFinishLaunchingWithOptions	Method ini digunakan untuk menjalankan *initialization code* ketika proses eksekusi aplikasi di kesempatan pertama
application:didFinishLaunchingWithOptions	Method ini digunakan untuk menjalankan *initialization code* sebelum aplikasi ditampilkan ke pengguna
applicationDidBecomeActive:	Method ini dijalankan ketika aplikasi berada di *foreground*
applicationWillResignActive:	Method ini dijalankan ketika aplikasi akan meninggalkan mode *foreground*
applicationDidEnterBackground:	Method ini dijalankan ketika aplikasi telah memasuki mode *background*
applicationWillEnterForeground:	Method ini dijalankan ketika aplikasi akan memasuki mode *foreground*
applicationWillTerminate:	Method ini dijalankan ketika aplikasi akan dihentikan dan dihapus dari memori.

Sumber: Apple, 2017

2.4 Core Motion Framework

Core motion framework menyedakan data pergerakan pada perangkat bergerak berbasis iOS, yang meliputi sensor *gyroscope, accelerometer, pedometer, magnetometer* dan *barometer. Framework* ini digunakan untuk mengakses data yang telah digenerate oleh *hardware* lalu digunakan di aplikasi yang sedang dibangun (Apple, 2017).

Berikut penjelasan dari beberapa klas dalam *Core Motion Framework* yang akan digunakan dalam penelitian ini:

2.4.1 CMMotionManager

Klas CMMotionManager digunakan untuk mengakses data dari sensor *accelerometer, gyroscope* ataupun data *device motion*. Setelah melakukan instansiasi dari klas ini, aplikasi dapat menggunakannya untuk mengambil data

mentah dari *accelerometer* maupun *gyroscope* ataupun mengambil data hasil olahan yang berupa data *device motion.*

Tabel 2.3 Fungsionalitas & Deskripsi *Instance variable/method* dari CMMotionManager

Fungsionalitas	***Instance Variable/Method***	**Deskripsi**
Mengetahui adanya layanan dari sensor	`var isDeviceMotionAvailable: Bool`	Nilai boolean yang menyatakan ketersediaan dari *device motion*
	`var isAccelerometerAvailable: Bool`	Nilai boolean yang menyatakan ketersediaan dari sensor *accelerometer*
	`var isGyroAvailable: Bool`	Nilai boolean yang menyatakan ketersediaan dari sensor *gyroscope*
	`var isMagnetometerAvailable: Bool`	Nilai boolean yang menyatakan ketersediaan dari *magnetometer*
Mengetahui bahwa layanan telah aktif	`var isDeviceMotionActive: Bool`	Nilai boolean yang menyatakan bahwa aplikasi sedang menerima *update* dari *device motion service*
	`var isAccelerometerActive: Bool`	Nilai boolean yang menyatakan bahwa

		accelerometer data sedang diperbarui
	`var isGyroActive: Bool`	Nilai boolean yang menyatakan bahwa *gyroscope* data sedang diperbarui
	`var isMagnetometerActive: Bool`	Nilai boolean yang menyatakan bahwa *magnetometer* data sedang diperbarui
Mengelola pembaruan dari *device motion*	`var showsDeviceMovementDisplay: Bool`	Variabel kontrol yang digunakan untuk menampilkan pergerakan device di layar
	`var deviceMotionUpdateInterval: TimeInterval`	Jeda waktu dalam memberikan pembaruan data *device motion*
	`func startDeviceMotionUpdates(using: CMAttitudeReferenceFrame, to: OperationQueue, withHandler: CMDeviceMotionHandler)`	Menjalankan pembaruan *device motion* berdasarkan *attitude reference frame* yang spesifik pada *operation queue* dengan *handler* spesifik
	`func startDeviceMotionUpdates(to: OperationQueue, withHandler: CMDeviceMotionHandler)`	Menjalankan pembaruan *device motion* pada *operation*

		queue dengan *handler* spesifik
	`func startDeviceMotionUpdates(using: CMAttitudeReferenceFrame)`	Menjalankan pembaruan *device motion* berdasarkan *attitude reference frame,* namun tanpa *handler*
	`func startDeviceMotionUpdates()`	Menjalankan pembaruan *device motion* namun tanpa *handler*
	`func stopDeviceMotionUpdates()`	Menghentikan pembaruan *device motion*
	`var deviceMotion: CMDeviceMotion?`	Sample data terkini dari *device motion*

Sumber: Core Motion Documentation, Apple, 2017

2.4.2 CMDeviceMotion

Hasil instansiasi dari klas CMDeviceMotion mengenkapsulasi data yang terkait dengan *device motion*, seperti *attitude, rotation rate,* maupun data *acceleration*

Tabel 2.4 Fungsionalitas & Deskripsi *Instance variable/method* dari CMDeviceMotion

Fungsionalitas	***Instance Variable/Method***	**Deskripsi**
Mendapatkan data *attitude* dan *rotation rate*	`var attitude: CMAttitude`	Data *attitude* dari perangkat
	`var rotationRate: CMRotationRate`	Data *rotation rate* dari perangkat
Mendapatkan data percepatan / *acceleration*	`var gravity: CMAcceleration`	Vektor percepatan gravitasi berdasarkan *reference frame* perangkat

	`var userAcceleration: CMAcceleration`	Nilai percepatan yang diberikan pengguna kepada perangkat
Mendapatkan data *magnetic field*	`var magneticField: CMCalibratedMagneticField`	Nilai medan magnet pada perangkat
Mendapatkan data *heading*	`var heading: Double`	Nilai *heading* perangkat yang dinyatakan dalam derajat

Sumber: Core Motion Documentation, Apple, 2017

2.4.3 CMAttitude

Klas CMAttitude merepresentasikan data tingkah laku dari perangkat bergerak (*attitude*), yang berisikan *quaternion*, nilai dari sumbu *pitch, yaw* dan *roll.* Prinsip *attitude* disini mirip dengan prinsip *attitude* pada *flight dynamics* pesawat terbang.

Tabel 2.5 Fungsionalitas & Deskripsi *Instance variable/method* dari CMAttitude

Fungsionalitas	***Instance Variable/Method***	**Deskripsi**
Mendapatkan data sudut Euler	`var roll: Double`	Nilai *roll* dari perangkat, dalam radian
	`var pitch: Double`	Nilai *pitch* dari perangkat, dalam radian
	`var yaw: Double`	Nilai *yaw* dari perangkat, dalam radian
Mendapatkan data *rotation matrix*	`var rotationMatrix: CMRotationMatrix`	Nilai *rotation matrix* yang merepresentasikan *attitude* perangkat
Mendapatkan data *quaternion*	`var quaternion: CMQuaternion`	Nilai *quaternion* yang merepresentasikan *attitude* perangkat
Mendapatkan perubahan *attitude*	`func multiply(byInverseOf: CMAttitude)`	Melakukan perubahan data *attitude* dengan *attitude* yang dilewatkan pada method tersebut

Sumber: Core Motion Documentation, Apple, 2017

2.5 Core Bluetooth Framework

Core Bluetooth framework menyediakan berbagai macam klas yang dibutuhkan oleh aplikasi iOS maupun Mac untuk berkomunikasi dengan perangkat yang dilengkapi dengan teknologi *Bluetooth Low Energy* seperti alat monitor detak jantung maupun termostat *digital* (Apple, 2017).

2.5.1 CBCentral

Klas CBCentral merepresentasikan *remote central device* yang terkoneksi dengan aplikasi lain yang berfungsi sebagai *bluetooth peripheral*. Ketika mengimplementasikan *peripheral role* melalui klas CBPeripheralManager, *central* yang terkoneksi dengan *peripheral* tersebut direpresentasikan sebagai objek CBCentral.

2.5.2 CBCentralManager

Objek yang diinstan dari klas CBCentralManager digunakan untuk mengelola *peripheral device* yang berhasil ditemukan maupun yang berhasil dikoneksikan, yang direpresentasikan sebagai objek CBPeripheral. Selain itu pula, digunakan untuk melakukan pemindaian, *discovery*, maupun untuk menyambungkan *central* ke *peripheral*.

2.5.3 CBPeripheral

Klas CBPeripheral merepresentasikan *remote peripheral device* yang telah berhasil ditemukan maupun yang berhasil terkoneksi dengan *central device. Peripheral* diidentifikasi dengan UUID (*Universally Unique Identifier) ,* yang direpresentasikan oleh objek NSUUID. *Peripheral* juga membawa informasi mengenai layanan yang tersedia pada *peripheral* dan kekuatan sinyal.

2.6 Head Mounted Display

Perangkat *Head Mounted Display* merupakan layar kecil atau teknologi proyeksi yang diintegrasikan pada kacamata, atau dipasangkan pada helm atau topi (Gartner, 2017). Penggunaan perangkat ini sangat erat dengan penggunaan teknologi *Virtual Reality* yang menawarkan pandangan dari layar perangkat yang dipasangkan pada HMD seolah pengguna melihat objek tersebut secara langsung.. Beberapa perangkat HMD yang populer saat ini adalah Google Cardboard maupun Oculus Rift.

Gambar 2.11 Google Cardboard

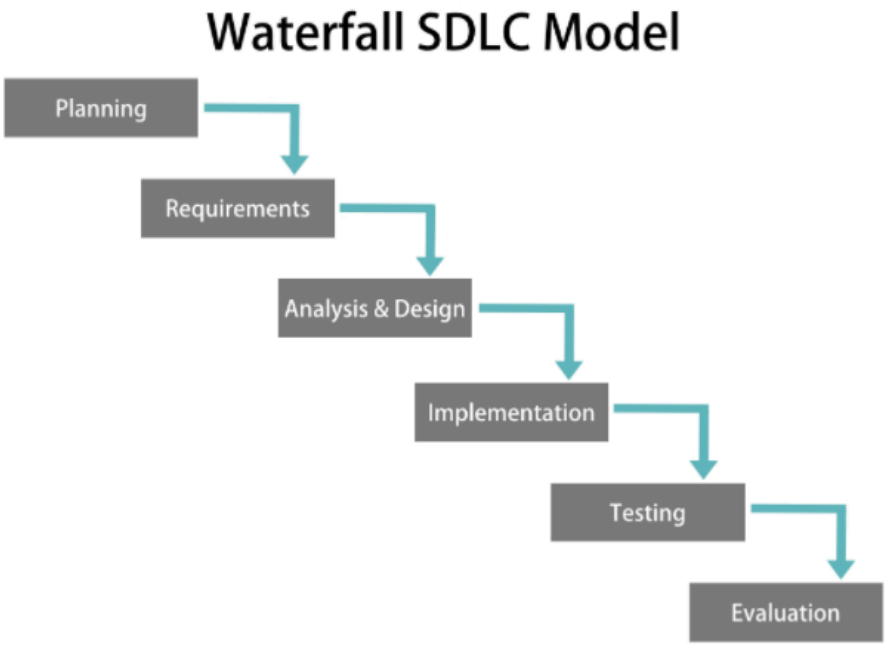
Sumber: Google VR, 2017

Gambar 2.12 Oculus Rift

Sumber: Oculus.com, 2017

2.7 Software Development Life Cycle: Waterfall

Model waterfall menggunakan teknik tradisional dalam perencanaan, pengujian dan teknik implementasi untuk merancang dan mengimplementasikan produk perangkat lunak yang baru. Model ini merepresentasikan urutan proses secara sekuensial yang mana prosesnya terlihat seperti aliran yang mengalir kebawah secara konstan. (CMU Open Learning Initiative, 2017)



Gambar 2.13 Aliran SDLC Waterfall