BAB II

TINJAUAN PUSTAKA

2.1 Pengenalan Karakter Huruf Jepang

Bahasa Jepang adalah salah satu bahasa yang paling unik di dunia. Hal tersebut dikarenakan dalam metode penulisan bahasa jepang sendiri menggabungkan berbagai macam bentuk karakter huruf. Huruf-huruf tersebut adalah Hiragana, Katakana dan Kanji (Das dan Banerjee 2014). Setiap jenis huruf atau karakter tersebut memiliki fungsi dan peranan tersendiri. Kindaichi yang merupakan salah seorang pakar linguistik bahasa Jepang mengklasifikasikan karakteristik-karakteristik bahasa Jepang ke dalam 5 kelompok besar yaitu hatsuon, moji, goi, bunpo dan hyogen (yang secara berurutan dalam bahasa Indonesia adalah ucapan, huruf, kosa kata, tata bahasa dan ekspresi). Unsur-unsur bahasa Jepang meliputi kanji, cara baca kanji, hitsujun (cara penulisan kanji), bushu (bagian kanji yang menentukan arti), rikusho (pembentukan dan pemakaian kanji), hiragana, dan katakana serta fungsinya masing-masing Dalam penulisannya bahasa Jepang menggabungkan huruf-huruf atau karakter sebagai berikut :

2.1.1 Hiragana

Kana mencakup 2 macam huruf yaitu hiragana dan katakana. Kedua macam huruf tersebut memiliki fungsi yang berbeda dalam penggunaannya. Huruf hiragana melambangkan suku kata tunggal, dan digunakan untuk menulis kata-kata yang berasal dari Jepang asli. Huruf hiragana terbentuk dari modifikasi dan penyederhanaan kanji, sehingga huruf hiragana coretannya melengkung dan tidak bersudut tajam. Huruf hiragana berjumlah 46 huruf di dalamnya mencakup 5 vokal yaitu a i u e o sisanya adalah suku kata dengan deretan ka, sa, ta, na, ha, ma, ya ra, wa dan satu konsonan yaitu n serta satu kata bantu o yang kadangkala diucapkan wo. 46 huruf hiragana tersebut di atas disebut sei on, selain itu dalam penggunaannya terdapat dakuon, handakuon dan yoo on. Huruf hiragana yang termasuk sei on (Brenda Carqua, Sunneng Sandino Berutu, Majalah Ilmiah Vol. 07, No. 02, Juli 2015).

Tabel 1.1 Hiragana Sei On

HURUF HIRAGANA					CARA BACA				
あ	い	う	え	お	a	i	U	e	o
か	き	く	け	こ	ka	ki	Ku	ke	ko
さ	し	す	せ	そ	sa	shi	Su	se	so
た	ち	つ	て	と	ta	chi	tsu	te	to
な	に	ぬ	ね	の	na	ni	Nu	ne	no
は	ひ	ふ	へ	ほ	ha	hi	Fu	he	ho
ま	み	む	め	も	ma	mi	mu	me	mo
や		ゆ		よ	ya		Yu		yo
ら	り	る	れ	ろ	ra	ri	Ru	re	ro
わ				を	wa				wo
ん					n				

Daku on berjumlah 20 huruf hiragana. Huruf hiragana yang termasuk dalam daku on ditunjukkan pada gambar tabel berikut.

Tabel 1.2 Hiragana Daku On

HURUF HIRAGANA					CARA BACA				
が	ぎ	ぐ	げ	ご	ga	gi	Gu	ge	go
ざ	じ	ぜ	ず	ぞ	za	ji	Zu	ze	zo
だ	ぢ	づ	で	ど	da	ji	Zu	de	do
ば	び	ぶ	べ	ぼ	ba	bi	Bu	be	bo

Handaku on berjumlah 5 huruf hiragana. Tabel 1.3 menunjukkan huruf hiragana yang termasuk dalam handaku on.

Tabel 1.3 Hiragana Handaku On

HURUF HIRAGANA					CARA BACA				
ぱ	ぴ	ぷ	ぺ	ぽ	pa	pi	Pu	pe	po

Yoo on berjumlah 36 huruf hiragana. Tabel 1.4 menunjukkan huruf hiragana yang termasuk dalam yoo on.

Tabel 1.4 Hiragana Yoo On

HURUF HIRAGANA			CARA BACA		
きゃ	きゅ	きょ	kya	kyu	kyo
しゃ	しゅ	しょ	sha	shu	sho
ちゃ	ちゅ	ちょ	cha	chu	cho
にゃ	にゅ	にょ	nya	nyu	nyo
ひゃ	ひゅ	ひょ	hya	hyu	hyo
みゃ	みゅ	みょ	mya	myu	myo
りゃ	りゅ	りょ	rya	ryu	ryo
ぎゃ	ぎゅ	ぎょ	gya	gyu	gyo
じゃ	じゅ	じょ	ja	ju	jo
びゃ	びゅ	びょ	bya	byu	byo
ぴゃ	ぴゅ	ぴょ	pya	pyu	pyo

2.1.2 Katakana

Huruf katakana sama seperti huruf hiragana yaitu melambangkan suku kata tunggal, tetapi mempunyai fungsi yang berbeda dengan huruf hiragana. Huruf katakana selain digunakan untuk menulis kata-kata yang berasal dari bahasa asing, juga digunakan untuk penekanan suatu kata yang berasal dari Jepang asli. Huruf katakana juga terbentuk dari modifikasi kanji dengan cara mengambil salah satu bagian kanji, sehingga di dalam huruf katakana tidak akan ditemukan coretan yang melengkung seperti hiragana. Katakana memiliki bentuk huruf yang terkesan kaku, karena setiap coretannya bersudut tajam. Pada Tabel 1.5 akan ditunjukkan huruf katakana yang termasuk dalam sei on.

Tabel 1.5 Katakana Sei On

HURUF KATAKANA					CARA BACA				
ア	イ	ウ	エ	オ	a	i	U	e	o
カ	キ	ク	ケ	コ	ka	ki	Ku	ke	ko
サ	シ	ス	セ	ソ	sa	shi	Su	se	so
タ	チ	ツ	テ	ト	ta	chi	tsu	te	to
ナ	ニ	ヌ	ネ	ノ	na	ni	Nu	ne	no
ハ	ヒ	フ	ヘ	ホ	ha	hi	Fu	he	ho
マ	ミ	ム	メ	モ	ma	mi	mu	me	mo
ヤ		ユ		ヨ	ya		Yu		yo
ラ	リ	ル	レ	ロ	ra	ri	Ru	re	ro
ワ				ヲ	wa				wo
ン					n				

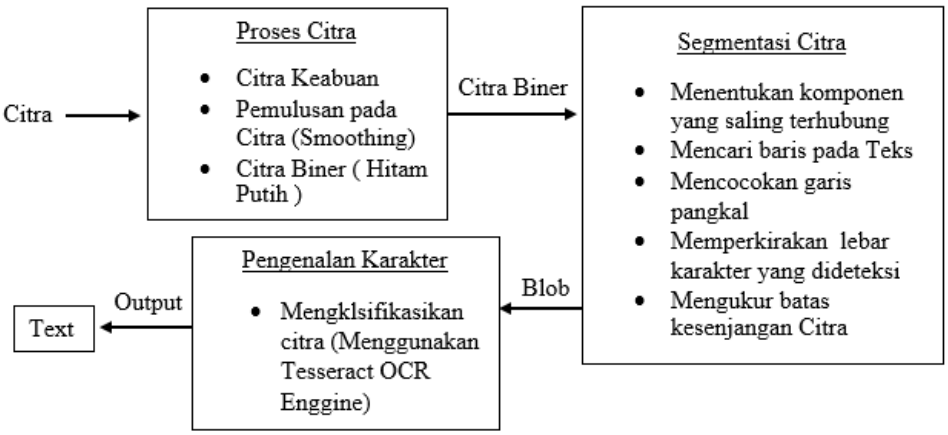
2.1.3 Kanji

Kanji adalah seperangkat simbol yang berasal dari sistem penulisan bahasa Cina. Masing-masing simbol tersebut mewakili satu atau lebih suku kata didalamnya, namun tidak seperti hanya mengeja kata, simbol-simbol ini juga memiliki makna sendiri. Bahkan jika ada dua kata yang dieja memiliki sama bunyi dan dengan makna yang berbeda, maka kanji yang digunakannya pun akan berbeda (Das dan Banerjee 2014). Macam-macam huruf kanji dapat dilihat pada Gambar 1.6.

Gambar 1.6 Huruf Kanji

2.2 Proses pengenalan Karakter

Proses pengenalan karakter Huruf Jepang dalam Penelitian ini menggunakan *Tessract OCR Enggine*. Proses pengenalan karakter Jepang menggunakan *Tesseract OCR* dapat dilihat pada Gambar 2.1 (Hanny Rindiani 2015).



Gambar 2.1 Proses Pengenalan Karakter

2.3 Image Processing

Image processing merupakan suatu bentuk pengolahan atau pemrosesan sinyal dengan input berupa gambar (image) dan ditransformasikan menjadi gambar lain sebagai keluarannya dengan teknik tertentu. Image processing terdiri atas 3 proses yaitu *grayscale*, *smoothing* (Pemulusan Citra), *tresholding* (Citra Biner).

2.3.1 Citra Grayscale

Citra grayscale merupakan citra digital yang hanya memiliki satu nilai kanal pada setiap pikselnya, artinya nilai dari Red = Green = Blue. Nilai-nilai tersebut digunakan untuk menunjukkan intensitas warna. Citra yang ditampilkan dari citra jenis ini terdiri atas warna abu-abu, bervariasi pada warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat. *Citra grayscale* berbeda dengan citra ”hitam-putih”, dimana pada konteks komputer, citra hitam putih hanya terdiri atas 2 warna saja yaitu ”hitam” dan ”putih” saja. *Citra grayscale* seringkali merupakan perhitungan dari intensitas cahaya pada setiap piksel pada spektrum elektromagnetik single band. Pada proses pengubahan, output dalam tahap ini akan digunakan untuk binerisasi gambar. Program akan melakukan pengulangan untuk grayscaling gambar per-piksel sebesar ukuran panjang dan lebar gambar.

2.3.2 Smoothing

Smoothing atau sering disebut *Blurring* adalah metode untuk mengkaburkan objek. Tujuannya adalah untuk mengurangi noise, sehingga mempermudah pemrosesan citra, terutama proses pendeteksian tepi. Hal ini karena proses *smoothing* dapat menghaluskan atau menyatukan bagian citra yang terpisah. Filter yang digunakan untuk smoothing adalah linear dengan 1 dimensi. *Smoothing* dilakukan dengan *Gaussian blur* dengan *matriks konvulsi* 3 x 3.

Gausian blur sendri adalah metode yang banyak digunakan dalam menghaluskan gambar, biasanya untuk mengurangi gangguan gambar dan mengurangi detail. blur halus yang menyerupai tampilan gambar melalui layar tembus cahaya, sangat berbeda dari efek bokeh yang dihasilkan oleh lensa yang tidak fokus atau bayangan objek di bawah penerangan biasa. *Gaussian smoothing* juga digunakan sebagai tahap pra-pemrosesan dalam algoritma visi komputer untuk meningkatkan struktur gambar pada skala yang berbeda-lihat representasi ruang skala dan implementasi skala ruang.

Secara matematis, penerapan *Gaussian blur* ke gambar sama dengan mengkonvolrasikan gambar dengan fungsi *Gaussian*. Ini juga dikenal sebagai *Transformasi Weierstrass* dua dimensi. Sebaliknya, dikonvolusi oleh lingkaran (yaitu, kotak lingkaran kabur) akan lebih akurat mereproduksi efek bokeh. Karena *Transformasi Fourier* dari *Gaussian* adalah *Gaussian* lain, menerapkan *Gaussian blur* memiliki efek mengurangi komponen frekuensi tinggi gambar; *gaussian blur* adalah *filter low pass*. *The Gaussian blur* adalah jenis filter gambar-blurring yang menggunakan fungsi *Gaussian* (yang juga mengungkapkan distribusi normal dalam statistik) untuk menghitung transformasi untuk diterapkan ke setiap piksel dalam gambar.

Di mana x adalah jarak dari titik awal dalam sumbu horizontal, y adalah jarak dari titik awal dalam sumbu vertikal, dan σ adalah deviasi standar distribusi *Gaussian*. Ketika diterapkan dalam dua dimensi, rumus ini menghasilkan permukaan yang konturnya adalah lingkaran konsentris dengan distribusi *Gaussian* dari titik pusat. Nilai dari distribusi ini digunakan untuk membangun matriks konvolusi yang diterapkan pada gambar asli. Proses konvolusi ini diilustrasikan secara visual pada gambar di sebelah kanan. Setiap nilai baru piksel diatur ke rata-rata tertimbang dari lingkungan piksel itu. Nilai piksel asli menerima bobot terberat (memiliki nilai Gaussian tertinggi) dan piksel yang berdekatan menerima bobot yang lebih kecil karena jaraknya ke piksel asli meningkat. Ini menghasilkan blur yang mempertahankan batas dan tepi lebih baik daripada filter lain yang lebih seragam dan kabur.

Secara teori, fungsi *Gaussian* pada setiap titik pada gambar tidak akan nol, yang berarti bahwa seluruh gambar perlu dimasukkan dalam perhitungan untuk setiap piksel. Dalam prakteknya, ketika menghitung pendekatan diskrit fungsi *Gaussian*, piksel pada jarak lebih dari 3σ memiliki pengaruh yang cukup kecil untuk dianggap nol secara efektif. Selain bersifat sirkuler simetris, *blur Gaussian* dapat diterapkan pada gambar dua dimensi sebagai dua penghitungan satu dimensi independen, dan karenanya disebut filter terpisah. Artinya, efek penerapan matriks dua dimensi juga dapat dicapai dengan menerapkan serangkaian *matriks Gaussian* satu dimensi dalam arah horizontal, kemudian mengulangi proses dalam arah vertikal.

Efek blur Gaussian biasanya dihasilkan dengan mengkonvolrasikan gambar dengan kernel nilai Gaussian. Dalam praktiknya, yang terbaik adalah memanfaatkan properti Gaussian blur yang dapat dipisahkan dengan membagi proses menjadi dua laluan. Pada pass pertama, kernel satu dimensi digunakan untuk mengaburkan gambar hanya dalam arah horizontal atau vertikal. Pada

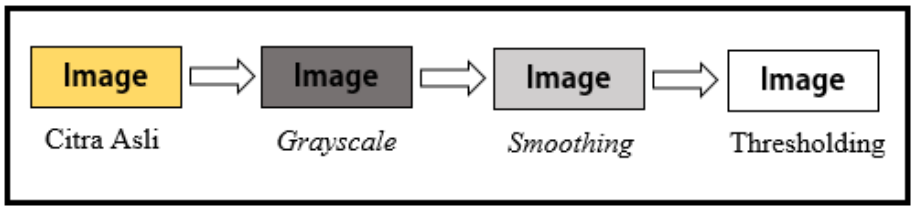
lintasan kedua, kernel satu dimensi yang sama digunakan untuk memburamkan dalam arah yang tersisa. Efek yang dihasilkan adalah sama dengan convolving dengan kernel dua dimensi dalam satu pass, tetapi membutuhkan lebih sedikit perhitungan.

Diskritisasi biasanya dicapai dengan sampling kernel filter Gaussian pada titik-titik diskrit, biasanya pada posisi yang sesuai dengan titik tengah setiap piksel. Hal ini mengurangi biaya komputasi tetapi, untuk kernel filter yang sangat kecil, titik sampling fungsi Gaussian dengan sangat sedikit sampel mengarah ke kesalahan besar. Saat mengonversi nilai berkelanjutan Gaussian menjadi nilai diskrit yang diperlukan untuk kernel, jumlah nilai akan berbeda dari 1. Ini akan menyebabkan penggelapan atau pencerahan gambar. Untuk memperbaiki ini, nilai-nilai dapat dinormalkan dengan membagi setiap term dalam kernel dengan jumlah dari semua istilah di kernel.

Gaussian blurring umumnya digunakan ketika mengurangi ukuran gambar. Ketika meng-downsampling gambar, adalah umum untuk menerapkan filter *low-pass* ke gambar sebelum *resampling*. Ini adalah untuk memastikan bahwa informasi frekuensi tinggi palsu tidak muncul dalam gambar *downsampled* (aliasing). *Gaussian blur* memiliki sifat yang bagus, seperti tidak memiliki tepi tajam, dan dengan demikian tidak memperkenalkan dering ke gambar yang di filter.

2.3.3 Thresholding (Citra Biner)

Thresholding merupakan proses mengubah citra berderajat keabuan menjadi citra biner atau hitam putih sehingga dapat diketahui daerah mana yang termasuk objek dan background dari citra secara jelas. Citra hasil *thresholding* biasanya digunakan lebih lanjut untuk proses pengenalan obyek serta ekstraksi fitur. Pada tahap ini *thresholding* dapat mengubah gambar berwarna maupun *grayscale* menjadi *binary image* dengan mengubah masing-masing pixel dalam kisaran tertentu. Gambar 2.2 menunjukkan tahap *Image Preprocessing* pada citra karakter.



Gambar 2.2 Tahap Image Processing

2.4 Segmentasi

Setelah tahap *Image processing* selesai, tahap selanjutnya dalam pengenalan karakter huruf Jepang adalah tahap Segmentation. Pada tahap Segmentation ini terdapat 2 proses yaitu *Connected component labeling* dan *Chopping Characters.*

2.4.1 Connected Component Labeling

Proses untuk mendeteksi component-component karakter yang saling terhubung antara satu karakter dengan karakter yang lain. Pada proses ini *Tesseract* melakukan pencarian sepanjang citra kemudian mengidentifikasi piksel latar depan atau *outline*, proses segmentasi akan terus dilakukan sampai semua piksel terluar ditandai sebagai *outline karakter*. *Outline* dari setiap karakter akan dikumpulkan menjadi blob. Proses ini membutuhkan deteksi tepi biner canny untuk mengetahui batas tepian dari citra yang akan dihubungkan dalam gambar tepian biner. (Ray Smith : 2007). Dapat dilihat pada gambar 2.3 tersebut.

Gambar 2.3 Proses Connected Componen Labeling (Ray Smith 2007)

2.4.2 Chopping Characters

Pada tahap ini dilakukan pemotogan karakter sehingga karakter pada kalimat akan terpotong menjadi teks kata dengan garis persegi panjang yang menunjukkan potongan kandidat titik pemisahan dengan garis pemotong vertikal dan Horizontal. pemotongan terpilih sebagai sebuah garis melewati *outline*. Pemotongan dilakukan sesuai dengan urutan prioritas. Setiap pemotongan yang gagal tidak sepenuhnya dibuang tetapi disimpan oleh associator sehingga jika pemotongan dapat digunakan kembali jika dibutuhkan (Ray Smith 2007). Dapat dilihat pada gambar 2.4 Tahap Chopping Character.

Gambar 2.4 Tahap Chopping Character

2.5 Kalsifikasi (*Tesseract OCR Enggine*)

Setelah tahap *segmentation* citra sudah selesai dilakukan, pada tahap selanjutnya adalah tahap *klasifikasi*. Pada tahap *klasifikasi* ini dilakukan dengan mencari kombinasi jarak terbaik dari data training dan data uji. Disini *Tesseract OCR Enggine* sudah menyediakan data training yang dapat diunduh sesuai jenis bahasa yang ingin dikenali. *Tesseract OCR Enggine* melakukan *ekstraksi tropological fitur* sebagaimana yang dilakuka oleh (Shillman et al : 1974) dengan *aproksimasi segmen poligon* (Ray Smith 1987). *Klasisfikasi adaptif* dan membandingkan fitur citra yaitu hasil normaslisasi *baseline dan moment*, dengan data training. menunjukkan citra karakter yang telah dinormalisasi pada baseline dan moment untuk selanjutnya diklasifikasi menggunakan *Tesseract OCR Enggine*.(Hanny Rindiani : 2015).

2.5.1 Tesseract OCR Enggine

Mesin *Tesseract* sendiri awalnya dikembangkan sebagai perangkat lunak yang dikembangkan di laboratorium Hewlett Packard di Bristol, Inggris dan Greeley, Colorado antara 1985 dan 1994, dengan beberapa perubahan yang dilakukan pada tahun 1996 ke port ke Windows. dan beberapa migrasi dari C ke C++ pada tahun 1998. Banyak kode ditulis dalam C, dan kemudian beberapa lagi ditulis dalam C++. Sejak saat itu semua kode telah diubah untuk setidaknya mengkompilasi dengan kompiler C++. Kemudian di tahun 2005, Tesseract dikembangkan menjadi *open source* oleh HP dan UNLV (Ray Smith : 2007). Secara garis besar *Tesseract OCR Enggine* merupakan mesin *open source* pengenalan karakter yang dapat digubakan oleh berbagai sistem operasi.

Pada penelitian (Ray Smith : 2007) *Tesseract OCR Enggine* telah memberikan hasil yang cukup baik untuk tiap pengenalan karakter dengan rata-rata tingka kesalahan hanya mencapai 3,77% atau dengan kata lain mampu mengenali karakter-karakter mencapai 96,33% yang mampu mengenali karakter dengan baik.

2.6 Pengenalan Teks ke Suara Pengucapan (*Cloud Text To Speech API*)

Setelah melalui proses pengengenalan karakter huruf *Jepang* (Hiragana ,Katakana dan Kanji) akan menghasilkan output berupa teks bahasa *Jepang* yang sudah dikenali dari citra *karakter huruf Jepang*. Setelah dikenali ke teks bahasa jepang, tahap selanjutnya adalah melakukan pengenalan ke pengucapan suara dengan menggunakan bantuan *Cloud Text To Speech API* yang bisa di dapatkan

di Google Developher untuk diaplikasikan ke dalam program *Aplikasi Text To Speech Pengucapan kata* dalam *Bahasa Jepang* berbasis *Android*.

2.6.1 Text To Speech

Text To Speech merupakan sistem yang bisa merubah teks (bahasa tulisan) menjadi ucapan atau suara yang sesua dengan kalimat yang di ingin diucapkan. Sistem *Text To Speech* memproduksi sinyal ucapan secara otomatis melalui transkripsi teks ke fonem pada kalimat yang diberikan. Phonem atau fonem adalah istilah linguistik dan merupakan satuan terkecil dalam sebuah bahasa yang masih bisa menunjukkan perbedaan makna. Bagian konverter teks ke fonem berfungsi untuk mengubah kalimat masukan dalam suatu bahasa tertentu yang berbentuk teks menjadi rangkaian kode-kode bunyi yang biasanya direpresentasikan dengan kode *phonem*, durasi serta pitch-nya. Hal inilah yang membedakan sistem *Text To Speech* dengan mesin bicara lainnya. Sistem voice response systems misalnya, bekerja dengan merangkai susunan kata terpisah (isolated word), hanya sesuai untuk aplikasi dengan jumlah kosa kata yang terbatas. Dalam konteks sistem *Text To Speech*, sangat tidak mungkin untuk menyimpan seluruh kata dari satu bahasa. (Samsudin, Riko Yopi Putra : 2014) Sistem Text to Speech pada prinsipnya terdiri dari dua subsistem dasar, yaitu :

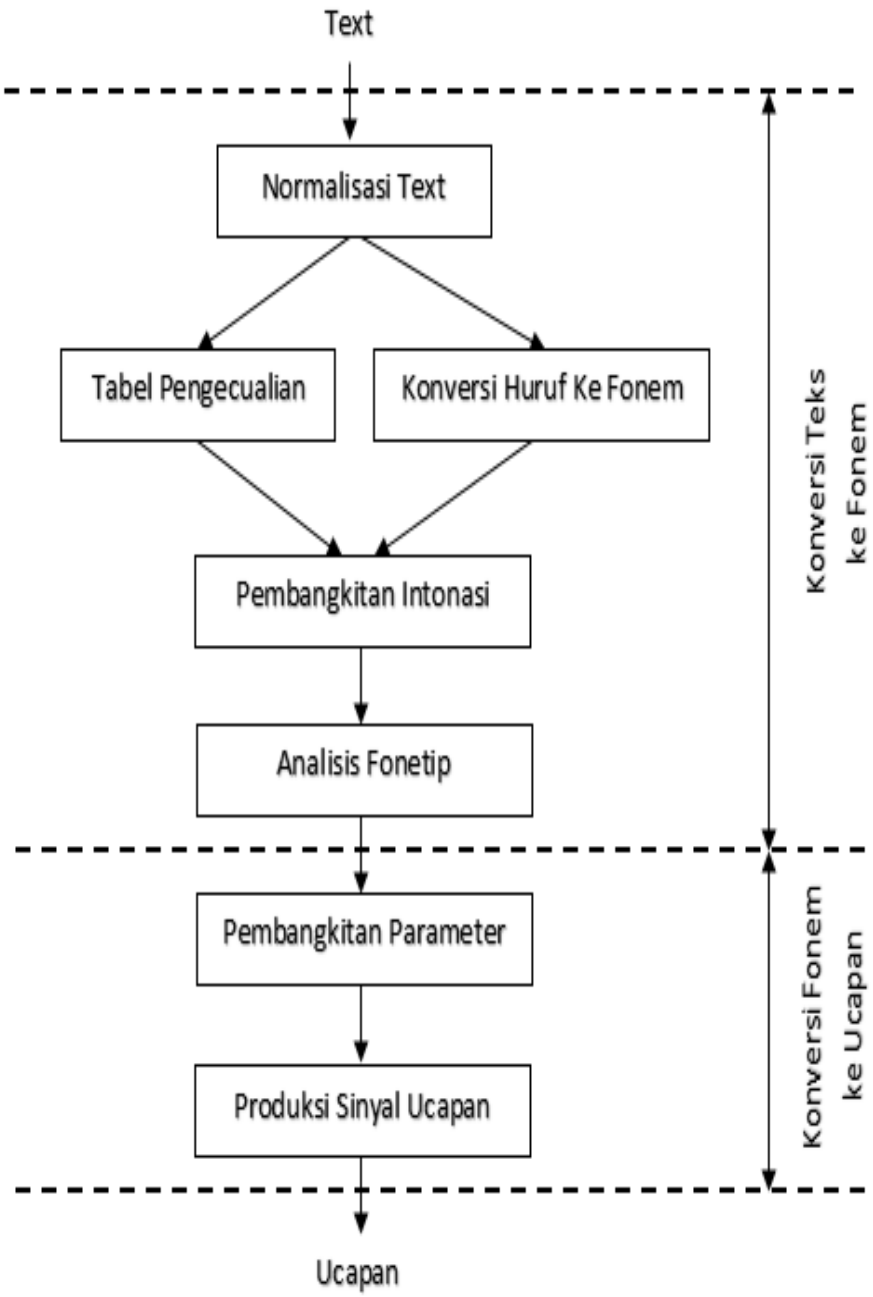
1. Subsistem konverter teks ke fonem

Subsistem konverter teks ke *fonem* yang memiliki dua fungsi utama. Pertama adalah mengambil kalimat masukan dalam suatu bahasa tertentu yang berbentuk barisan teks dan mengubah beberapa hal seperti nomor dan tanda ke dalam tulisan sesuai dengan bunyi yang seharusnya, sering disebut dengan *normalisasi teks* (text normalization). Kemudian menentukan kode *fonetik* (phonetic transcriptions) untuk tiap kata beserta durasi dan nadanya. Kode fonem adalah kode yang merepresentasikan unit bunyi yang ingin diucapkan. Pengucapan kata atau kalimat pada prinsipnya adalah urutan bunyi atau secara simbolik adalah urutan kode fonem. (Samsudin, Riko Yopi Putra : 2014)

2. Subsistem konverter fonem ke ucapan

Subsistem konverter *fonem* ke ucapan yang akan menerima masukan kode-kode *fonem* serta nada dan durasi yang telah dihasilkan oleh bagian sebelumnya. Berdasarkan kode-kode tersebut bagian ini akan menghasilkan bunyi atau sinyal *ucapan* yang sesuai dengan kalimat yang

ingin diucapkan. (Samsudin, Riko Yopi Putra : 2014) Urutan proses konversi Teks Ke Suara dapat dilihat pada Gambar 2.5



Gambar 2.5 Proses Konversi Teks Ke Suara Pengucapan

2.6.2 Cloud Text To Speech API

Google Developher menyediakan berbagai API (*Application Programming Interface*) yang sangat berguna bagi pengembang aplikasi Android, web maupun aplikasi desktop untuk memanfaatkan berbagai fitur yang disediakan oleh *Google Developher*. API secara sederhana bisa diartikan sebagai kode program yang merupakan antarmuka atau penghubung antara aplikasi atau web yang dibuat dengan fungsi-fungsi yang dikerjakan. (swahyudi, Catur, 2010) di *Google cloud developher* sudah menyediakan layanan *API* yang akan kita gunakan untuk mengubah teks ke suara yaitu mengunakan *Cloud Text To Speech API.* Algoritma untuk merubah Teks ke suara adalah sebagai berikut :(Chaw Su Thu, Theingi Zin, Mandalay Technological University, 2014)

1. Pertama periksa kondisi jika *Cloud Text To Speech API* itu tersedia di komputer atau tidak. Jika tidak tersedia maka error akan dihasilkan dan Perpustakaan *Cloud Text To Speech API* harus dimuat di komputer.
2. Mendapat objek suara dari *Cloud Text To Speech API.*
3. Membandingkan string input/ text dengan *Cloud Text To Speech API.*
4. Ekstrak suara dengan terlebih dahulu dan pilih suara yang ada tersedia di perpustakaan *Cloud Text To Speech API.*
5. Pilih kecepatan suara.
6. Menginisialisasi wave player untuk mengubah teks menjadi suara.
7. Output berupa suara pengucapan yang sesuai dengan teks inputan.

2.7 Perangkat Pengembangan

Pemodelan (modeling) merupakan proses perancang piranti lunak sebelum melakukan pengkodean (coding). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

2.7.1 Unified Modelling Language (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis

dalam bahasa pemrograman apapun. (Sri Dharwiyanti, IlmuKomputer.Com : 2003)

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML syntax mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (Object-Oriented Design), Jim Rumbaugh OMT (Object Modeling Technique), dan Ivar Jacobson OOSE (Object-Oriented Software Engineering). Desain UML yang digunakan dalam penelitian ini adalah use case diagram dan activity diagram.

2.7.2 Konsepsi Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML. Sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam gambar 2.6 tersebut.

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Gambar 2.6 Komsep Dasar UML(Sri Dharwiyanti, IlmuKomputer.Com:2003)

Abstraksi konsep dasar UML yang terdiri dari structural classification, dynamic behavior, dan model management, bisa kita pahami dengan mudah apabila kita melihat gambar 2.10 tersebut dari Diagrams. Main concepts bisa kita pandang sebagai term (hubungan) yang akan muncul pada saat kita membuat diagram. Dan view adalah kategori dari diagaram tersebut.

Desain UML yang digunakan dalam penelitian ini adalah Use Case diagram dan Actvity diagram.

1. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. Use case merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. (Sri Dharwiyanti, IlmuKomputer.Com : 2003). Terdapat 3 elemen dalam use case diagram, yaitu :

- Use Case

Merupakan proses-proses yang terjadi dalam sebuah sistem. Use case mendiskripsikan fungsi pada sistem yang mudah dipahami. Use case dipresentasikan dalam bentuk elips dengan keterangan di dalamnya.

- Actor

Merupakan pengguna yang berinteraksi dalam suatu sistem, dimana setiap pengguna menjalankan proses-proses tertentu dalam sebuah sistem. Actor dipresentasikan dengan gambar stickman dengan nama dan peraturan tertentu.

- Relationship

Relationship digambarkan sebagai garis antara dua simbol pada diagram use case. Arti dari relationships dapat berbeda tergantung pada bagaimana garis ditarik dan apa jenis simbol yang menghubungkan mereka.

Tabel 1 – Simbol UML Use Case Diagram

Gambar	Nama	Fungsi
	Package	Menambahkan paket baru dalam diagram
	Actor	Menambah aktor dalam diagram
	Use case	Menambahkan *use case* pada diagram
	Unidirectional association	Menggambarkan relasi antara aktor dengan *use case*
	Dependencies or Instantiates	Menggambarkan kebergantungan *(dependencies)* antar item dalam diagram
	Generalization	Mengggambarkan relasi lanjut antar *use case* atau menggambarkan struktur pewarisan antar actor

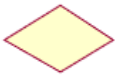
2. Activity Diagram

Activity diagrams menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. Activity diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan state diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-trigger oleh selesainya state sebelumnya (internal processing). Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu use case atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. (Sri Dharwiyanti, IlmuKomputer.Com : 2003).

Tabel 2 – Simbol UML Activity Diagram

Gambar	Nama	Fungsi
	State	Menambahkan *state* untuk suatu objek
	Activity	Menambahkan aktivitas baru pada diagram
	Start state	Memperlihatkan dimana aliran kerja berawal
	End state	Memperlihatkan dimana aliran kerja berakhir
	State transition	Menambah transisi dari suatu aktivitas ke aktivitas yang lainnya
	Transition to self	Menambah transisi rekursif
	Horizontal synchronization	Menambahkan sinkronisasi *horizontal* pada diagram
	Vertical synchronizations	Menambahkan sinkronisasi *vertikal* pada diagram
	Decisions points	Menambahkan titik keputusan pada aliran kerja

2.8 Penelitian Terdahulu

2.8.1 Penelitian Hanny Rindiani (2015)

Pada proses pengenalan karakter nenurut penelitian (Rindiani, H. 2015, "*Aplikasi Android untuk Pengenalan Citra Karakter Jepang dengan Library Tesseract*", Ilmu Komputer IPB, Bogor) Pada penelitian ini sistem pengenalan karakter Jepang berbasis Android dirancang untuk mengenali citra karakter Jepang dan menerjemahkannya menjadi bahasa Indonesia menggunakan library Tesseract OCR (Optical Character Recognition). Pada penelitian ini terdapat 5 kebutuhan fungsional pengguna yaitu mengambil gambar dengan kamera, mengambil gambar dari galeri, melakukan konversi dari citra ke teks, mengedit teks Jepang hasil OCR, dan menampilkan hasil terjemahan teks Jepang. Berdasarkan 10 sampel karakter Jepang yang telah diuji, nilai akurasi yang di dapat dari pengujian gambar dari kamera sekitar 80% dan gambar dari galeri 94%. Selain itu pegujian aplikasi menggunakan kuesioner mencapai 91%.

2.8.2 Penelitian Samsudin Dkk (2014)

Pada penelitian (Samsudin, Riko, Y.P. 2014, "*Perancangan Aplikasi Text to Speech Pengenalan kalimat dalam Bahasa Inggris Menggunakan Metode Linear Predictive Coding*", Teknik Informatika, UIN Sumatera Utara) Linier predictive coding adalah subjek yang penting dalam kaitannya dengan text-processing. Secara sederhana konsep linier predictive coding dapat diterjemahkan sebagai sebuah cara untuk mencari string yang sama dalam sebuah kumpulan teks (dokumen) atau database. Hasil dri peneletian ini dapat mengenal kalimat bahasa inggris dengan baik. Kata yang akan dibaca akan terblok dengan sendirinya sehingga user dapat mengetahui kata yang sedang dibaca. Hasil penggunaan aplikasi terhadap kalimat bahasa Inggris baku dan non baku bisa terdengar dengan baik terutama kalimat bahasa Inggris.

2.8.3 Penelitian Nessa Putri Anandayu (2013)

Pada penelitian (Anandayu, N.P. 2013, "*Perancangan Text To Speech Converter Enggine Dalam pengucapan kata berbahasa Arab sehari-hari*", Program Studi Teknik Informatika, Jurusan Teknik Elektro Fakultas Teknik Universitas Tanjungpura, Pontianak) Penelitian ini berfokus pada pengembangan suatu algoritma sederhana dalam teks Bahasa Arab ke proses ucapan dengan menggabungkan kata atau kalimat yang akan diucapkan. Berdasarkan hasil pengujian dengan sampel 40 kata mendapatkan persentase keberhasilan sekitar 95%, dengan kondisi karakter hijaiyah yang memiliki harakat, tidak ada simbol,

angka, dan tanda baca pada teks input, dan teks input yang mengandung bacaan aturan atau tajwid tidak dapat diproses untuk diucapkan dengan tajwid.

2.8.4 Penelitian Pande Made Mahendri Pramadewi Dkk (2013)

Pada penelitian (Pramadewi, P.M.M., Kesiman, M.W.A., Darmawiguna, I.G.M. 2013. "*Pengembangan Aplikasi Text To Speech untuk Bahasa Bali*". Universitas Pendidikan Ganesha Singaraja, Bali)Penelitian ini bertujuan, merancang aplikasi Text to Speech untuk bahasa Bali, mengimplementasikan rancangan aplikasi Text to Speech untuk bahasa Bali. Metode penelitian yang digunakan adalah pengembangan (development). Proses konversi teks ke ucapan, dirancang dan diimplementasikan memiliki dua tahapan proses yaitu: proses konversi teks ke fonem; proses konversi fonem ke ucapan yang menggunakan metode diphone concatenation dengan memanfaatkan MBROLA dan diphone database id1 yang dibangkitkan oleh eSpeak. Hasil penelitian ini adalah aplikasi Text to Speech untuk bahasa Bali yang dapat mengonversi teks dalam format bahasa Bali dengan huruf latin, menjadi ucapan sesuai dengan pembacaan teks dalam bahasa Bali dan menyimpan ucapan yang dihasilkan ke dalam bentuk file keluaran berupa audio file berekstensi *.wav dan phoneme file dengan format input MBROLA berekstensi *.pho. Aplikasi tersebut diberi nama "Bali Text to Speech" yang disingkat sebagai "BaliTTS" dan diimplementasikan dengan bahasa pemrograman Java. Hasil uji coba menunjukkan bahwa aplikasi BaliTTS telah dapat melakukan fungsinya dengan baik dari segi fungsional dan konseptual/ struktural.