BAB II

STUDI PUSTAKA

2.1 Tinjauan Pustaka

2.1.2 *State Of The Art*

Terdapat beberapa penelitian yang telah dilakukan sebelumnya dalam studi kasus transliterasi aksara sunda kedalam aksara latin dengan beberapa metode ataupun algoritma. Dengan adanya hal tersebut maka perlu adanya suatu studi literatur terhadap penelitian sebelumnya yang memiliki korelasi dengan penelitian yang akan dilakukan,hal itu berguna untuk memposisikan penelitian lanjutan yang akan dilakukan peneliti. Berikut merupakan *state of the art* yang memiliki korelasi yang sama diantaranya :

1. Dwi Aprianto (2018), telah melakukan penelitian dengan judul "Implementasi *Optical Character Recognition* pada kamus aksara sunda-Indonesia menggunakan Algoritma *Feature Extraction* berbasis Android". Bertujuan untuk Transliterasi Aksara sunda kedalam aksara latin dengan algoritma *Feature Extraction* menggunakan *Optical Character Recognition* [3]. Masalah yang terdapat pada penelitian ini yaitu untuk menyikapi fenomena dari kurangnya pelestarian aksara sunda, penelitian ini juga berfokus pada pengembangan aplikasinya untuk bidang pendidikan, menggunakan lagoritma *feature extraction* dan juga *tesseract engine* untuk mengenali karakter huruf aksara sunda yang di transliterasikan ke aksara latin. Dimana terdapat beberapa tahapan dalam implementasi algoritma *feature extraction*, diantaranya menggunakan

algoritma *Nearest Neighbour Interpolation* hal ini dilakukan agar citra terbaca secara baik dan benar oleh tesseract. Hasil dari penelitian ini yaitu *feature Extraction* dan *tesseract engine* dapat membaca huruf aksara sunda dengan ketepatan 55,56% terbaca tepat, 42,96% terbaca, dan 1,48% tidak terbaca.

2. Derayan Bima Alamyah (2018), telah melakukan penelitian dengan judul "implementasi *text recognition* untuk mendeteksi *digital writing* dan *hand writing* dalam alfabet latin menggunakan OCR (*optical character recognition*)", Bertujuan untuk mendeteksi tulisan digital dengan tulisan tangan dengan output menghasilkan presentase dari total penulisan *digital writing* dan *hand writing* dalam suatu dokumen atau kalimat [1]. Penelitian ini menggunakan algoritma *template matching* untuk membaca huruf alphabet dengan kamera ponsel, yang dimana algoritma *template matching* ini dapat membandingkan *template* target dengan *template* data latih. Hasil dari penelitian ini yaitu mendapatkan akurasi sekitar 99.8% dari jumlah total 30 pengujian dan divariasikan dengan 3 model pengujian.
3. Suryo Hartanto (2014), telah melakukan penelitian dengan judul "*Optical Character Recognition* menggunakan algoritma *template matching correlation*", Bertujuan untuk mengkonversi suatu dokumen cetak ke dokumen digital dengan algoritma *template matching* [4]. Permasalahan bersumber dari perlunya pengenalan tulisan dengan berbagai karakter huruf, terdapat tahapan *preprocessing* yaitu dimana mengolah citra masukan seperti format .jpg atau .bmp dengan tahapan meliputi binerisasi, segmentasi, dan normalisasi. Dari hasil penelitian memiliki hasil akurasi

sebesar 92,90%, yang berarti algoritma *template matching correlation* cukup efektif dalam membangun sebuah sistem pengenalan karakter dengan OCR.

4. Dani Rohpandi dkk (2015), telah melakukan penelitian dengan judul "aplikasi pengolahan citra dalam pengenalan pola Huruf ngalagena menggunakan Matlab" bertujuan untuk mengenali aksara ngalagena [5]. Pada penelitian ini menggunakan algoritma *template matching* sebagai cara dalam mengenali setiap pola karakter aksara sunda, terdapat beberapa tahapan dalam implementasi algoritmanya yaitu diantaranya preprocessing meliputi Binerisasi, segmentasi dan juga *identifier object* untuk mengenali pola. Pengujian dari penelitian ini yaitu menggunakan 3 jenis citra, yaitu citra pola karakter yang sama dengan template, citra pola karakter yang berbeda dengan template dan citra pola karakter yang ditulis dengan tangan, dimana hasilnya citra pola karakter yang sama dengan template menghasilkan akurasi sebesar 88%, citra pola karakter yang berbeda dengan template menghasilkan akurasi sebesar 60,87 dan citra pola karakter yang ditulis dengan tangan menghasilkan akurasi sebesar 32%.
5. Raden Sofian Bahri (2014), telah melakukan penelitian dengan judul "Perbandingan algoritma *template matching* dan *feature Extraction* pada *Optical Character Recognition*" untuk membandingkan algoritma *template matching* dan *feature Extraction* pada tulisan latin [6]. Penelitian ini bertujuan untuk mengetahui perbandingan tingkat akurasi dari algoritma *template matching* dan *feature extraction* dalam studi kasus pengenalan aksara latin, dari hasil penelitian didapat algoritma feature

extraction memiliki tingkat akurasi yang lebih tinggi dari algoritma template matching, hal ini karena algoritma *template matching* hanya menyesuaikan dengan data latih saja sedangkan algoritma *feature extraction* lebih fleksibel dengan membaca berdasarkan ciri-ciri khusus dari suatu karakter aksara sunda.

Tabel 2.1 Merupakan Kesimpulan Dari hasil studi literatur yang telah diambil dari beberapa sumber.

Tabel 2.1 *State Of The Art*

No.	Judul	Masalah	Perancangan	Hasil
1.	Aplikasi pengolahan citra dalam pengenalan pola Huruf ngalagena menggunakan Matlab	Perlunya suatu algoritma untuk mengenali pola karakter aksara sunda pada matlab.	Algoritma *Template Matching,* Matlab, Aksara Sunda.	Citra yang sama dengan template memiliki akurasi 88%, berbeda dengan template memiliki akurasi 60,87%, dan tuisan tangan memiliki akurasai 32%.
2.	Implementasi Character Recognition pada kamus aksara sunda-Indonesia menggunakan Algoritma *Feature Extraction* berbasis Android	Masalah yang terdapat pada penelitian ini yaitu untuk menyikapi fenomena dari kurangnya pelestarian aksara sunda.	OCR, Android, Algoritma *Feature Extraction.* Aksara Sunda.	Hasil dari penelitian ini yaitu *feature Extraction* dan *tesseract engine* dapat membaca huruf aksara sunda dengan ketepatan 55,56% terbaca tepat, 42,96% terbaca, dan 1,48% tidak terbaca.

Tabel 2.1 *State Of The Art* (lanjutan)

No.	Judul	Masalah	Perancangan	Hasil
3.	Implementasi *text recognition* untuk mendeteksi digital writing dan handwriting dalam alfabet latin menggunakan ocr (*optical character recognition*)	Belum adanya sistem yang dapat menghitung persentase yang membedakan tulisan digital dan tulisan tangan.	OCR, Android, Algoritma *Template Matching,* Aksara latin.	Hasil dari penelitian ini yaitu mendapatkan akurasi sekitar 99.8% dari jumlah total 30 pengujian dan divariasikan dengan 3 model pengujian.
4.	Perbandingan Algoritma *Template matching* dan *feature Extraction* pada *Optical Character Recognition*	Belum adanya perbandingan *Template matching* dan *feature Extraction* pada OCR aksara latin.	OCR, Algoritma *feature extraction,* Algoritma *Template Matching,* Aksara latin.	hasil penelitian didapat algoritma *feature extraction* memiliki tingkat akurasi yang lebih tinggi dari algoritma *template matching*
5.	*Optical Character Recognition* Menggunakan Algoritma *Template Matching Correlation*	Permasalahan bersumber dari perlunya pengenalan tulisan dengan berbagai karakter huruf.	OCR, Android, Algoritma, *Template Matching,* Aksara latin.	Dari hasil penelitian memiliki hasil akurasi sebesar 92,90% pada pembacaan pola karakter.

Tabel 2.2 Posisi Penelitian

Peneliti	Judul	Masalah	Perancangan
Muhamad Farid Padilah (2019).	Perbandingan Algoritma *Template Matching* dan Algoritma *Feature Extraction* pada Aplikasi Transliterasi Aksara Sunda menggunakan *Optical Character Recognition* Berbasis Android	perlu di kembangkannya jurnal mengenai perbandingan *template matching* dengan *feature extraction* dalam studi transliterasi aksara sunda ke latin.	Algoritma *Template Matching*, Algoritma Feature Extraction, Android, OCR, Aksara Sunda.

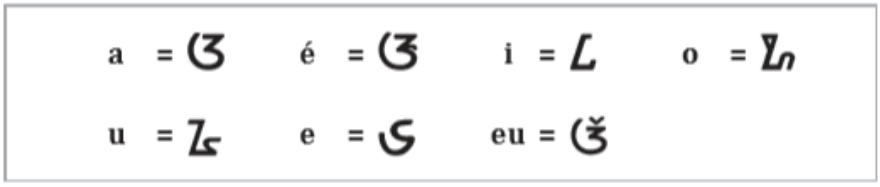
Berdasarkan penelitian sebelumnya yang berkaitan dengan penelitian ini, Algoritma yang banyak dipakai dalam citra digital yaitu Algoritma feature Extraction dan Algoritma Template Matching, Peneliti bermaksud untuk mengadakan penelitian lebih lanjut mengenai kedua Algoritma tersebut yang dimana peneliti membandingkan kinerja algoritma tersebut dari tingkat akurasi dan kecepatan dalam studi kasus pengenalan aksara sunda dengan menggunakan OCR (*Optical Character Recognition*) .

2.2 Landasan Teori

2.2.1 Aksara Sunda

a. Aksara Swara

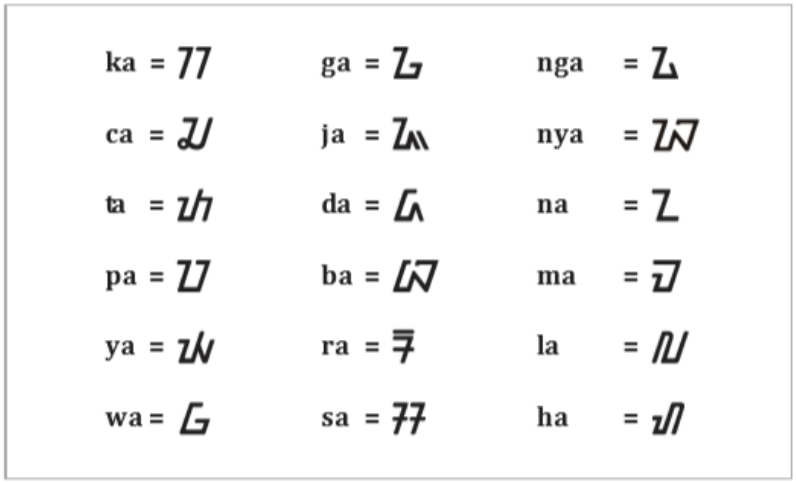
Aksara Swara merupakan vocal dari aksara sunda, terdapat 7 jenis aksara swara diantaranya yaitu a, i, u, e, o, eu, e [7]. seperti pada gambar 2.1 berikut.



Gambar 2.1 Aksara Swara

b. Aksara ngalagena dari bunyi sunda

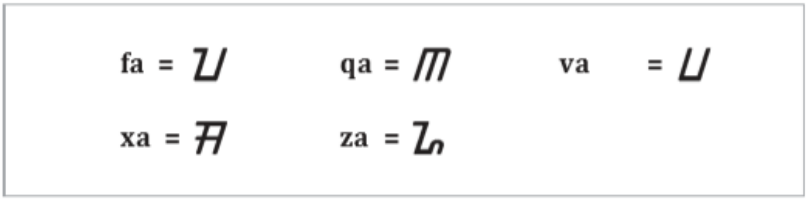
Ngalagena merupakan suatu konsonan dalam aksara sunda yang mengandung bunyi vokal “a”. Aksara ngalagena ini memiliki fungsi artikuasi sebagai alat ucap yang nantinya dapat disatukan dengan rarangken sunda. Terdapat 18 aksara ngalagena yaitu seperti pada gambar 2.2 dibawah ini.



Gambar 2.2 Aksara Ngalagena

c. Aksara ngalagena serapan

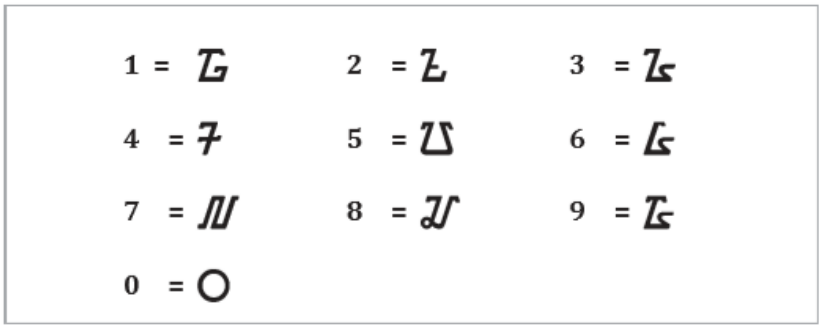
Terdapat 5 aksara sunda dari bunyi serapan yaitu seperti pada gambar 2.3 dibawah ini.



Gambar 2.3 Aksara Ngalagena Serapan

d. Angka dalam aksara sunda

Angka dalam aksara sunda yaitu sebagai seperti pada gambar 2.4 dibawah ini.



Gambar 2.4 Angka Aksara Sunda

e. Rarangkén dalam aksara sunda

Rarangkén adalah sebuah aturan dalam aksara sunda yang dimana mengatur dari setiap artikulasi dan vokalisasi aksara sunda. Terdapat 13 Rarangkén yang dapat dikategorikan berdasarkan letak penulisannya [7]. Diantaranya sebagai berikut.

1. Rarangken dibawah huruf

	panyuku, membuat vokal aksara *Ngalagena* dari [a] menjadi [u]. Contoh:  = ka →  = ku.
	panyakra, menambah konsonan [r] di tengah suku kata. Contoh:  = ka →  = kra.
	panyiku, menambah konsonan [l] di akhir suku kata. Contoh:  = ka →  = kla.

Gambar 2.5 rarangkén di bawah huruf

2. Rarangken diatas huruf

Terdapat 5 Rarangken diatas huruf yaitu :

◌ᩤ	*panghulu*, membuat vokal aksara *Ngalagena* dari [a] menjadi [i]. Contoh: ᨀ = ka → ◌ᩤ = ki.
◌ᩮ	*pamepet*, membuat vokal aksara *Ngalagena* dari [a] menjadi [ə]. Contoh: ᨀ = ka → ◌ᩮ = ke.
◌ᩯ	*paneuleung*, membuat vokal aksara *Ngalagena* dari [a] menjadi [ɤ]. Contoh: ᨀ = ka → ◌ᩯ = keu.
◌ᩬ	*panglayar*, menambah konsonan [r] pada akhir suku kata. Contoh: ᨀ = ka → ◌ᩬ = kar.
◌ᩰ	*panyecek*, menambah konsonan [ŋ] pada akhir suku kata. Contoh: ᨀ = ka → ◌ᩰ = kang.

Gambar 2.6 rarangkén di atas huruf

3. Rarangken sejajar dengan huruf

◌ᩮᨀ	*panéléng*, membuat vokal aksara *Ngalagena* dari [a] menjadi [ɛ]. Contoh: ᨀ = ka → ◌ᩮᨀ = ké.
◌ᨀᩣ	*panolong*, membuat vokal aksara *Ngalagena* dari [a] menjadi [ɔ]. Contoh: ᨀ = ka → ◌ᨀᩣ = ko.
◌ᩁ᩠ᨀ	*pamingkal*, menambah konsonan [j] di tengah suku kata. Contoh: ᨀ = ka → ◌ᩁ᩠ᨀ = kya.
◌ᩉ᩠ᨀ	*pangwisad*, menambah konsonan [h] di akhir suku kata. Contoh: ᨀ = ka → ◌ᩉ᩠ᨀ = kah.
◌᩵ᨀ	*patén* atau *pamaéh*, meniadakan vokal pada suku kata. Contoh: ᨀ = ka → pamaeh = k.

Gambar 2.7 rarangkén sejajar huruf

2.2.2 Java

a. Sejarah Singkat Java

Pelopor bahasa java di tahun 1991 adalah Mike Sheridan, Patrick Naughton, Chris Warth, Ed Frank, dan James Gosjilng. bahasa pemrograman ini pertamakali dinamai "Oak" yang selanjutnya berubah nama menjadi Java. Pada proses pengembangannya banyak orang yang terlibat diantaranya Frank Yellin, Jonathan Payne, Jonathan Payne, Arthur van Hoff, dan Bill joy. dan Tim Lindholm merupakan orang-orang yang menjadi kunci keberhasilan java [8].

b. Keunggulan Java

Java memiliki slogan yang unik yaitu *"write once run anywhere"* sehingga program java dapat dijalankan diberbagai platform seperti windows dan linux [9]. Berikut ini merupakan kelebihan dari bahasa pemrograman java diantaranya :

1. Bersifat *independent* dan *portable*

Syarat berjalannya bahasa pemrograman java di berbagai *platform* seperti windows dan linux yaitu harus tersedianya JVM pada *platform* tersebut.

2. Memori yang sedikit.

Java melakukan Pembuangan sampah yang tidak berguna sehingga alokasi dari memori menjadi kecil dan membuat bahasa pemrograman java terbebas dari masalah pengelolaan memori.

3. Bebas Arsitektur

Java sudah dirancang untuk bekerja pada macam-macam arsitektur prosesor dan berbagai sistem operasi. Java bersifat interpreter yaitu mengubah program java menjadi *byte-code.*

2.2.3 Android Studio

Android studio merupakan IDE (*Integrated Development Environment*) untuk sistem operasi Android. Android studio pertama kali diumumkan di Google I/O conference pada tanggal 16 Mei 2013. Ini merupakan tahap preview dari versi 0.1 pada Mei 2013, dan memasuki tahap beta sejak versi 0.8 dan mulai diliris pada Juni 2014. Beberapa fitur dukungan pada android studio adalah :

a. Dukungan gradle based build.
b. Terdapat template untuk membuat drawer dan macam-macam activity (*Template-based wizards*).
c. Memiliki editor untuk mengedit tata letak kompone UI dengan *drag and drop*, dan juga bisa melihat komponen tata letak dilayar.
d. Mendukung pengembangan android wear.
e. Terdapat emulator (Android *Virtual Device*).
f. Integrasi proguard dan kemampuan penanda tangan aplikasi.

2.2.4 Algoritma *Feature Extraction*

Algoritma *Feature Extraction* Merupakan Algoritma yang berfungsi untuk mengenali suatu *pattern* dalam suatu objek berdasarkan ciri-ciri khusus yang dimiliki objek tersebut. *Feature extraction* bertujuan untuk mengklasifikasi ciri – ciri yang dimiliki oleh suatu citra dengan melakukan perhitungan dan perbandingan.

Gambar 2.8 Ilustrasi *feature extraction*

Ciri-ciri dari suatu karakter akan disimpan sebagai *template*, lalu citra input yang akan dibandingkan akan di analisis berdasarkan ciri-ciri citra tersebut [6]. Berikut merupakan ciri-ciri dari *feature ekstraction* yaitu :

1. Dapat menghemat waktu komputasi karena ekstraksi yang jumlahnya sedikit.
2. Transformasi lebih fleksibel yang berarti tidak memiliki keterikatan dalam hal pergeseran, penskalaan, rotasi dan sebagainya.
3. Dapat membedakan antara objek masukan dengan objek masukan lainnya.
4. Memperoleh fitur dengan memperhatikan tingkat kompleksitas.

Untuk proses klasifikasi *Feature Extraction* dapat di lakukan dengan pendekatan K-NN (K-*Nearst Neighbor*). Yang dimana algoritma K-*Nearst Neighbor* ini dapat mengklasifikasikan objek berdasarkan ciri-ciri yang telah dipetakan oleh *feature extraction.* Dari hasil pemetaan tersebut maka akan di dapat diketahui nilai yang paling kecil (cocok) antara data *template* dengan data inputan [10]. Berikut merupakan rumus dari K-*Nearst Neighbor* :

$$knn = \sqrt{\sum_{i=1}^{p} (X2i - X1i)^2}$$

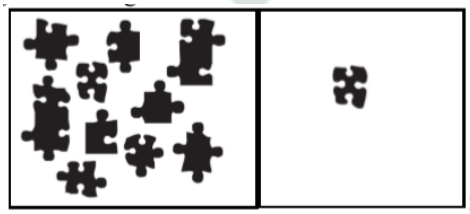
Keterangan : X2i = data ciri-ciri citra masukan.

X1i = data ciri-ciri citra *template*.

2.2.5 Algoritma *Template Matching*

Template matching adalah suatu pencocokan objek suatu citra dengan template yang menjadi acuan atau data latih, proses pencocokan dilakukan sampai kebagian-bagian terkecil objek gambar yang sudah dirubah kedalam bentuk biner. Dapat di analogikan bahwa template matching merupakan suatu cara manusia dalam mengenali pola suatu objek yang dilihat yang berdasarkan template dalam konteks rekognisi yang menunjuk pada konstruk internal, yang jika cocok maka akan dikenali oleh indra sebagai suatu objek yang sama.

Berikut merupakan ilustrasi dari template matching :



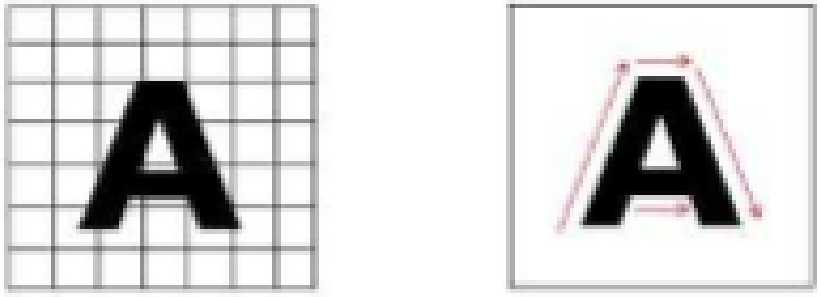
Gambar 2.9 Ilustrasi *Template Matching*

Template target merupakan suatu kumpulan dari studi kasus objek yang akan di input, yang kemudian nantinya akan disamakan dengan template inputan yang akan diperhitungkan kesesuaiannya. Pada proses penyesuaian ini template yang memiliki error terendah yang di anggap cocok dan sesuai dengan inputan [1]. Citra template berupa hasil matriks dan pencocokannya dikorelasikan secara *priority* (template yang medekati) dan *position.*

2.2.6 OCR (Optichal Character Recognition)

1. Definisi OCR (*Optichal Character Recognition*)

OCR merupakan proses pengenalan objek yang dikonversi dari karakter ASCII (*American Standad Code for Information Interchange*) yang dapat dikenali oleh komputer. Objek tersebut dapat berupa pindaian suatu huruf dalam dokumen, foto, gambar dan lain-lain. Terdapat dua tipe character recognition diantaranya yaitu *online character recognition* dan *offline character recognition.*

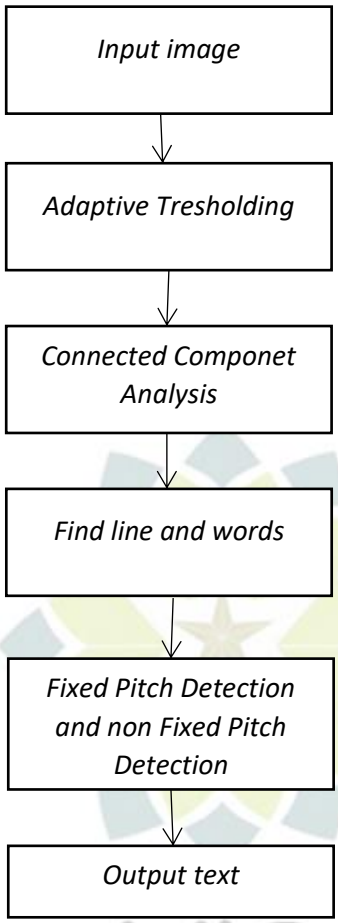


Gambar 2.10 Ilustrasi *OCR* dan *HCR*

Character recognition juga dapat dikategorikan menjadi dua tipe berdasarkan cara penulisan objectnya. Yaitu OCR (*Optichal character recognition*) dan HCR (*Handwriting Character Recognition*) [11]. Dalam segi tingkat akurasi antara HCR dengan OCR tentunya masih lebih tinggi OCR dimana OCR memindai tulisan static sedangkan HCR memindai tulisan bermacam-macam dan inkonsisten dari setiap karakternya [11].

2. Tesseract OCR (*Optichal Character Recognition*)

gambar 2.11 merupakan tahapan dalam pengenalan suatu objek *Tesseract engine* OCR.



Gambar 2.11 Alur *Tesseract engine*

a. Input *Image*

Pengenalan pada tesseract OCR yang pertama yaitu menginputkan *image* berupa pindaian dari dokumen, foto ataupun gambar.

b. Adaptive Tresholding

Adaptive Tresholding merupakan proses merubah gambar yang telah diinput menjadi biner [12].

c. *Connected Component Analysis*

Connected Component Analysis Merupakan proses pendefinisian panjang citra dengan mencari *outline* atau piksel latar depan. Proses ini

akan terus dilakukan sampai semua pixel menjadi BLOB (*Binary Long Object*) [12].

d. Find Line and Words

Find Line and Words merupakan tahapan pencarian yang dapat mengenali tulisan miring, dan pencarian baris yang cocok. Algoritma pada tahap ini diantaranya *line finding*, *blob filtering* dan *line construction* [12].

e. Fixed pitch Detection dan *Non Pixed pitch Detection*

Tahap *Fixed pitch Detection* ini merupakan tahap pendeteksian objek yang memiliki lebar yang tetap, apabila suatu objek terdeteksi maka tesseract akan melakukan *chopping* terhadap objek tersebut, apabila objek terrsebut tidak di temukan maka akan masuk ke *Non Pixed pitch Detection*, pada *Non Pixed pitch Detection* tesseract akan melakukan algoritma untuk mengukur kesenjangan antara garis dasar dan garis tengah [12].

2.2.7 Android

Android awalnya merupakan sistem operasi *open source* untuk *mobile* berbasis linux, android awalnya dibangun oleh Android, inc yang kemudian dibeli oleh google. Untuk sekarang Sistem operasi android sudah berkembang untuk berbagai peralatan elektronik lain selain *handphone* diantaranya Android TV (untuk tv), Android wear (untuk jam tangan), Android Auto (untuk mobil).

Android memiliki beberapa versi yaitu cupcake (API level 3), Donut (API level 4), Eclair (Api level 5), Froyo (Api level 8), Ginger Bread (Api level 9), Honey Comb (API level 11), Ice cream Sandwich (API level 14), jelly bean (Api

level 16), Kitkat (API level 19), Lollipop (API level 21), Marshmellow (API level 23), Nougat (Api level 26), Oreo (Api level 27), Pie (Api level 28) [13].

2.2.8 Citra Digital

Citra digital merupakan suatu representasi dari sebuah objek diantaranya yaitu gambar atau foto dalam bentuk digital, citra digital terdiri dari sebuah matriks kolom (M) dan baris (N), dan perpotongan kolom dan baris (piksel), parameter dari sebuah citra yaitu warna dan koordinat, nilai pada koordinat (x,y) dapat ditulis f(x,y), Gambar 2.12 merupakan Ilustrasi Matriks citra digital.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & \dots & \dots & f(1,M-1) \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$

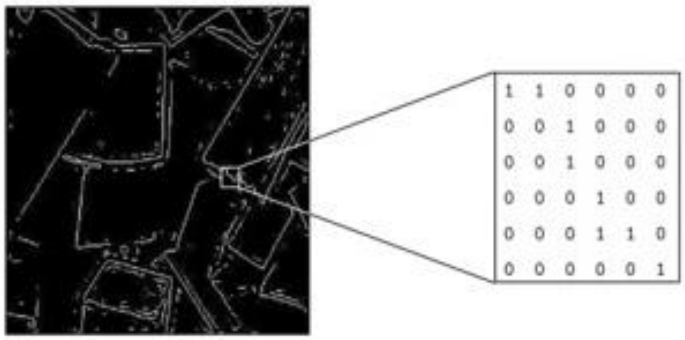
Gambar 2.12 Ilustrasi Matriks citra digital

Berdasarkan matriks tersebut fungsi dari suatu citra dapat ditulis f(x,y), x merupakan baris dan y merupakan kolom. Dan f(x,y) adalah nilai keabuan dari suatu citra tersebut. Macam-macam citra digital adalah sebagai berikut :

1. Citra Biner

Citra Biner merupakan citra yang diproses melalui pemisahan piksel, pemisaahannya dilakukan berdasarkan derajat keabuan objek tersebut. Citra biner hanya memiliki dua warna yaitu warna hitam dan warna putih, untuk menyimpan warna ini dibutuhkan 1 bit *memory*. Pada citra biner ini setiap warna memiliki

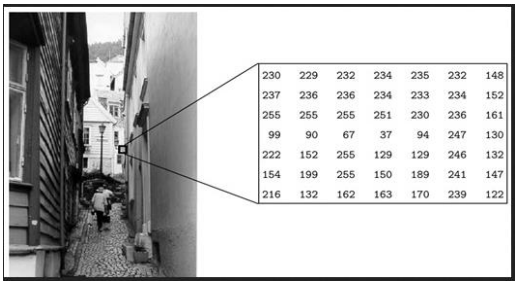
nilai yaitu warna putih bernilai 1 sedangkan warna hitam bernilai 0. Gambar 2.13 merupakan ilustrasi citra *Biner*.



Gambar 2.13 Ilustrasi Citra Biner

2. Citra Grayscale

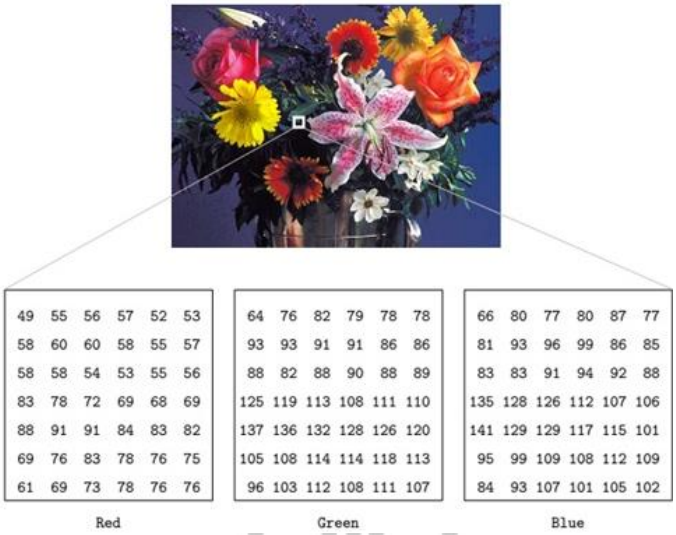
Citra grayscale merupakan citra dengan warna RGB (*Red Green Blue*) dengan intensitas yang sama. Citra grayscale ini berwarna abu-abu, banyaknya suatu warna yang terdapat pada suatu citra yaitu tergantung dari bit yang disediakan memori, memori ini berfungsi sebagai penambpung kebututhan warna. Untuk setiap pixel pada citra disimpan dalam format 8 bit, yang memiliki 256 intensitas. Hal ini dapat mempermudah dalam pemogaman karena untuk mendapatkan citra graysale ini hanya perlu mendapatkan rata-rata dari nilai R, G, dan B. Gambar 2.14 merupakan ilustrasi citra *Grayscale*.



Gambar 2.14 Ilustrasi Citra *Grayscale*

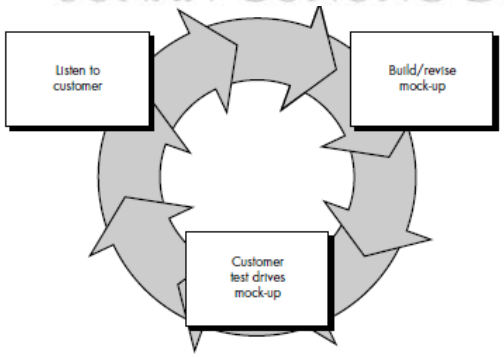
3. Citra Warna

Citra warna warna memiliki 16.777.216 variasi warna, yang dimana setiap pixelnya memiliki 24 bit. Jumlah ini tentunya sudah cukup untuk menampilikan warna yang dapat dilihat oleh manusia. Komposisi dari seluruh warna R,G dan B disimpan kedalam 1 byte data, yang dimana setiap warna merah, hijau dan bitu memiliki 8 bit. Gambar 2.15 merupakan ilustrasi citra warna.



Gambar 2.15 Ilustrasi Citra Warna

2.2.9 Metode *Prototype*



Gambar 2.16 Metode *Prototype*

Metode pengembangan *Prototype* dapat disebut juga dengan desain aplikasi cepat (RAD/*Rapid Aplication Design*). Metode pengembangan *prototype*

bertujuan untuk menganalisis dengan memahami kebutuhan dari user guna menerjemahkannya kedalam bentuk pemodelan sehingga mendapatkan aplikasi yang di inginkan.

Terdapat beberapa fase dalam pengembangan perangkat lunak secara umum yaitu:

1. Fase Pendefinisian
    a. Perencanaan.
    b. Definisi Perangkat Lunak.
    c. Alokasi keperluan sumberdaya.
    d. Penjadwalan.
    e. Perkiraan Biaya.
    f. Mendefinisikan alokasi biaya yang akan dikeluarkan untuk pengembangan sistem yang akan dibangun.
2. Fase Pengembangan
    a. Perancangan.
    b. Mendeskripsikan arsitektur perangkat lunak yang akan dibuat.
3. Fase Verifikasi
    a. Verifikasi untuk semua unit fungsional.
    b. Pengetesan fungsi-fungsi antarmuka.
    c. Perawatan
    d. Mengecek ulang kesalahan-kesalahan perangkat lunak.
    e. Adaptasi terhadap perubahan lingkungan.
    f. Pengembangan kebutuhan mendatang.

McLeod dan Schell berpendapat mengenai kelebihan bagi pengguna dan sistem dalam penggunaan prototype yaitu :

a. Komunikasi yang baik antara pemakai dan sistem.
b. Analis dilakukan guna mencari kebutuhan pengguna sehingga dapat bekerja dengan lebih baik.
c. Pengguna berperan lebih aktif dalam pengembangan sistem;
d. Waktu pengembangan lebih efisien.
e. Mudahnya dalam mengimplementasikan dikarenakan pengguna mengetahui sistem yang direncanakan sesuai dengan kebutuhan yang sudah di analisa. [14]

2.2.10 UML (Unified Modeling Language)

Unified Modeling Language (UML) merupakan suatu metode dalam dokumentasi perancangan sistem berorientasi objek (PBO). Dan menjadi standar dalam penulisan *blue print software* [13]. Diagram UML meliputi :

1. Use Cace Diagram

Use Case Diagram merupakan representasi (gambaran) dari setiap aktor yang berinteraksi dengan sistem yang bersangkutan, Tabel berikut menggambarkan *use case diagram* :

Table 2.2 Simbol-Simbol *Use Case* Diagram

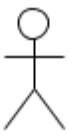
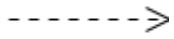
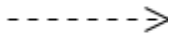
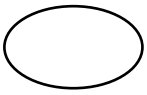
No	Gambar	Nama	Keterangan
1		*Actor*	Actor Merupakan entitas yang langsung berinteraksi dengan sistem

Table 2.2 Simbol-Simbol *Use Case* Diagram (lanjutan)

No	Gambar	Nama	*Keterangan*
2		*include*	*Include* menyatakan sumber *Use case* secara eksplisit.
3		*extends*	*Extends* menggambarkan hubungan yang memperluas suatu prilaku *Use Case*.
4		*Association*	*Association* merupakan gambaran hubungan antar objek.
5		System	*System* Menggambarkan suatu paket sistem yang ditampilkan terbatas.
6		*Use Case*	*Use Case* menggambarkan prilaku yang dapat dilakukan oleh *actor* pada sistem tersebut.

2. *Activity* Diagram

Activity Diagram merupakan gambaran dari setiap alur aktifitas pada suatu rancangan sistem yang di dalam-nya bisa terdapat kondisi-kondisi dari mulai sampai selesai. Tabel 2.3 merupakan komponen *activity diagram* [15].

Tabel 2.3 komponen *Activity Diagram*

No	Gambar	Nama	Keterangan
1		*Star Point*	Merupakan titik awal gambaran dari suatu objek
2		*End Point*	Merupakan akhir dari suatu aktifitas objek.
3		*Activity*	Merupakan gambaran dari suatu interaksi antar *interface* yang terjadi.
4		*Action*	Merupakan gambaran dari aksi yang di eksekusi.
5		*Fork Node*	Merupakan *flow* dalam menambah aliran tertentu dalam suatu proses.
6		*Decision Point*	Menggambarkan proses suatu keputusan berlangsung dalam suatu aliran aktifitas.

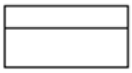
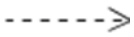
3. *Class* Diagram

class diagram merupakan ngambaran dari objek, package dan kelas yang berhubungan satu sama lain dalam suatu pembuatan aplikasi. Tabel 2.4 Merupakan Gambaran dari komponen *class diagram*.

Tabel 2.4 Komponen *Class Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Generalization*	Menggambarkan suatu hubungan antara *ancestor* (objek induk) dengan *descendent* (objek anak).
2		*N-ary Association*	Dipakai dalam menghindari dua objek dalam satu asosiasi.

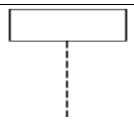
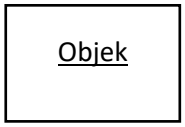
Tabel 2.4 Komponen *Class Diagram* (Lanjutan)

3.		*Class*	Menggambarkan suatu *class* dalam aplikasi yang menghimpun berbagai operasi *method* dan atribut.
4.		*Collaboration*	Menggambarkan suatu hasil bagi *actor* yang merupakan urutan aksi-aksi dalam suatu sistem
5.		*Realization*	Menggambarkan suatu operasi yang dilakukan oleh objek.
6.		*Dependency*	Hubunngan antara elemen tidak mandiri terhadap suatu elemen mandiri (*independent*).
7.		*Association*	Menggambarkan hubungan antara suatu objek dengan objek yang lain.

4. *Sequence* Diagram

Sequence diagram merupakan gambaran dari suatu alur hidup dalam proses dari seluruh interaksi *sequence* diagram menampilkan suatu pesan dalam urutan waktu. Tabel 2.5 Merupakan notasi dari *sequence* Diagram.

Tabel 2.5 Komponen *Sequence* Diagram

No	Gambar	Nama	Keterangan
1		*LifeLine*	Menggambarkan interaksi suatu *interface.*
2		*Message*	Menggambarkan suatu komunikasi yang dilakukan antar objek.
3		*Object*	Menggambarkan suatu instance dari class digambarkan berbentuk kotak.

2.2.11 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan elemen untuk menguji kualitas perangkat lunak baik dari desain, spesifikasi maupun pengkodean. pengujian bertujuan untuk mencari kesalahan perangkat lunak selama definisi fase awal dari fase pembangunan, pengembangan dilakukan untuk membangun perangkat lunak dari konsep yang abstrak sampai implementasi [16].

Pentingnya pengujian perangkat lunak dikaranakan sederetan aktivitas produksi yang memungkinkan peluang terjadinya kesalahan sangat besar dan karena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas [16].

1. Pengujian *Blackbox.*

Pengujian *Blackbox* ini menguji fungsionalitas dari perangkat lunak, dengan demikian pengujian *blackbox* menggunakan inputan yang sesuai dengan fungsionalitas dari sistem itu sendiri. Pengujian *blackbox* bukanlah alternatif dari pengujian *whitebox,* tetapi mungkin pendekatan komplementer yang kemungkinan besar mampu mengungkap jelas kesalahan dari pada metode *whitebox* [17]

Pengujian *blackbox* menguji fungsionalitas dari aplikasi sebagai berikut :

a. Fungsi - fungsi yang salah.
b. Kesalahan dari UI .
c. Kesalahan dalam basis data atau struktur.
d. Inisialisasi dari kesalahan terminasi.
e. Kesalahan kinerja.

Berbeda dengan *whitebox* yang melakukan pengujian diawal, pengujian *blackbox* diaplikasikan diakhir dikarenakan pengujian blackbox memperhatikan struktur dan berfokus pada domain informasi [17].