repository.ub.ac.id

IMPLEMENTASI SUPPORT VECTOR MACHINE BERDASARKAN CIRI HISTOGRAM OF ORIENTED GRADIENTS UNTUK VERIFIKASI CITRA TANDA TANGAN BERBASIS RASPBERRY PI

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
Mohammad Lutfi Zulfikri
NIM: 145150300111065

PROGRAM STUDI TEKNIK KOMPUTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

PENGESAHAN

IMPLEMENTASI SUPPORT VECTOR MACHINE BERDASARKAN CIRI HISTOGRAM OF ORIENTED GRADIENTS UNTUK VERIFIKASI CITRA TANDA TANGAN BERBASIS RASPBERRY PI

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :
Mohammad Lutfi Zulfikri
NIM: 145150300111065

Skripsi ini telah diuji dan dinyatakan lulus pada
27 Desember 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Hurriyatul Fitriyah, S.T, M.Sc	Wijaya Kurniawan, S.T, M.T
NIP. 19851001 201504 2 003	NIP. 19820125 201504 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP. 19710518 200312 1 001

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 16 Desember 2018

Mohammad Lutfi Zulfikri
NIM: 145150300111065

UNIVERSITAS BRAWIJAYA

UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT yang Maha Pengasih dan Maha Penyayang atas limpahan Rahmat dan Ridho-Nya sehingga laporan skripsi yang berjudul "IMPLEMENTASI SUPPORT VECTOR MACHINE BERDASARKAN CIRI HISTOGRAM OF ORIENTED GRADIENTS UNTUK VERIFIKASI CITRA TANDA TANGAN BERBASIS RASPBERRY PI" ini dapat terselesaikan dengan baik.

Dalam proses penyelesaian skripsi ini, penulis telah mendapatkan bantuan dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan ucapan terima kasih kepada:

1. Ibu Hurriyatul Fitriyah, S.T, M.Sc dan Bapak Wijaya Kurniawan, S.T, M.T selaku Dosen Pembimbing Skripsi yang telah meluangkan waktu untuk membimbing dan memberi arahan kepada penulis, sehingga penelitian ini dapat terselesaikan.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D selaku Ketua Jurusan Teknik Informatika.
3. Kedua orang tua penulis, Yayus Susiatno Hadi dan Alimah Alkaf yang telah memberikan dukungan moral berupa semangat dan doa.
4. Seluruh civitas akademik Fakultas Ilmu Komputer Universitas Brawijaya yang telah membantu penulis selama menempuh studi di Universitas Brawijaya terutama dalam penyelesaian skripsi ini.
5. Rekan-rekan penulis Reynald, Wisnu, Ganda, Devi, Fanani, Ikhwan, dan Sandy serta Keluarga Besar Teknik Komputer khususnya angkatan 2014 yang telah membantu memberikan saran maupun dukungan.

Dalam penyusunan skripsi ini penulis menyadari masih terdapat banyak kekurangan dan belum sempurna, oleh karena itu saran dan kritik yang membangun dibutuhkan untuk penyempurnaan penelitian ini. Akhir kata, penulis berharap penelitian ini dapat bermanfaat bagi semua pihak yang menggunakannya.

Malang, 16 Desember 2018

Penulis

mlutfizulfikri@gmail.com

repository.ub.ac.id

ABSTRAK

Tanda tangan telah lama digunakan oleh masyarakat sebagai salah satu alat untuk verifikasi identitas seseorang, umumnya tanda tangan digunakan dalam dokumen resmi. Sayangnya tanda tangan juga dapat disalahgunakan untuk memalsukan legalitas dokumen. Untuk menghindari penyalahgunaan atau pemalsuan tanda tangan tersebut, maka dibuatlah alat untuk dapat melakukan verifikasi tanda tangan. Sistem ini menggunakan kamera sebagai masukkan dengan *push button* sebagai pemicu kamera menangkap citra, Raspberry Pi sebagai unit pemroses citra digital, dan LCD 16x2 sebagai keluaran sistem. Penelitian ini menggunakan metode *Histogram of Oriented Gradients* (HOG) sebagai *feature descriptor* dan *preprocessing* citra seperti konversi *colorspace* citra, *Thresholding*, *Morphological Transformation*, dan deteksi dan koreksi kemiringan. Metode HOG tersebut akan menghasilkan *feature vector* yang merepresentasikan ciri tanda tangan pada citra, *feature vector* ini selanjutnya akan masuk ke proses klasifikasi *Support Vector Machine* (SVM) untuk dilakukan pelatihan data dan prediksi keluaran verifikasi citra. Dalam sisi *software*, terdapat dua bagian utama sistem, yaitu bagian pelatihan data untuk melatih data citra tanda tangan dengan SVM dan bagian verifikasi tanda tangan untuk prediksi keluaran verifikasi, selain itu sistem juga menggunakan bantuan *software* OpenCV sebagai *library* pengolahan citra. Dari pengujian akurasi verifikasi citra tanda tangan didapatkan hasil sebesar 87,33% dari data uji sebanyak 300 citra tanda tangan. Dari pengujian ini data latih sangat mempengaruhi akurasi, data latih yang memiliki pola tanda tangan asli yang cukup bervariasi dapat menurunkan akurasi sistem. Dalam pengujian waktu proses pelatihan data klasifikasi, rata-rata sistem membutuhkan 1,45 detik untuk melatih data, dalam setiap pengujiannya waktu pemrosesan tidak terlalu bervariasi, hal ini dikarenakan komputasi program ditangani oleh proses *scheduling* dari sistem operasi Raspbian.

Kata kunci: verifikasi tanda tangan, pengolahan citra, *histogram of oriented gradients*, *support vector machine*, raspberry pi

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Signatures have long been used by people as a tool to verify a person's identity. Generally, signatures are used in creating official documents. Unfortunately, signatures can also be misused to falsify the legality of documents. To avoid the misuse or the falsification of a signature, a system is made to verify the signature. This system uses a camera which is equipped with a push button to let the camera capture an image as an input, Raspberry Pi as a digital image processing unit, and a 16x2 LCD as a system output. This study uses the Histogram of Oriented Gradients (HOG) method as a feature descriptor and image preprocessing such as image colorspace conversion, Thresholding, Morphological Transformation, and skew detection and correction. The HOG method will produce a feature vector that represents the signature characteristics of the image. This feature vector will be going to the Support Vector Machine (SVM) classification process for data training and image verification output prediction. In the software side, there are two main parts of the system which are training data section to train signature image data with SVM and the signature verification section for verification output predictions. Besides, the system also uses OpenCV software as an image processing library. According to the accuracy of signature image verification test, accuracy results are 87.33% from test data of 300 signature images, according to this test, it is shown that the training data greatly affects accuracy. Training data that have a fairly varied original signature pattern can reduce system accuracy. From testing the classification data, it is shown that training process time system requires 1.45 seconds in average to train the data. In each test, the processing time is not too varied. This is because computational programs are handled by the scheduling process of the Raspbian operating system.

Keywords: signature verification, image processing, histogram of oriented gradients, support vector machine, raspberry pi

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Hingga saat ini tanda tangan masih dipakai sebagai salah satu alat untuk verifikasi identitas dari seseorang. Tanda tangan sering digunakan untuk mengesahkan dokumen penting atau memberi pernyataan bahwa penandatangan menyetujui isi yang tercantum dalam dokumen yang ditandatanganinya. Contoh dokumen penting yang membutuhkan tanda tangan yaitu kartu tanda penduduk, ijazah, kuitansi tanda transaksi, cek bank dan lain sebagainya. Di samping sifat tanda tangan yang dapat memberi legalitas berbagai macam dokumen, tanda tangan cukup mudah diduplikasi atau dipalsukan untuk kepentingan tertentu. Maka dari itu tanda tangan perlu dilakukan verifikasi untuk menghindari penyalahgunaan atau pemalsuan. Proses verifikasi tanda tangan dapat dilakukan dengan mencari perbedaan variasi antara penandatanganan dengan pemalsunya untuk memutuskan tanda tangan tersebut asli atau tidak (Harfiya, et al., 2017). Pada umumnya proses verifikasi sebuah tanda tangan masih menggunakan cara manual yaitu dengan melihat secara langsung, cara ini dianggap tidak efektif jika dokumen yang akan diverifikasi berjumlah banyak.

Verifikasi tanda tangan dapat dicapai dengan menemukan ekstraksi ciri yang cocok untuk citra tanda tangan. Fitur ciri yang baik seharusnya tidak ambigu dan dapat mendeskripsikan perbedaan ciri tanda tangan pada setiap individu (Dasgupta, et al., 2016). Ciri pada tanda tangan umumnya berbentuk gaya tulisan tangan dari nama atau inisial seseorang yang mempunyai pola unik tersendiri. Pola dari sebuah tanda tangan tersusun dari berbagai garis dan lengkungan yang mempunyai orientasi. Ciri orientasi dari citra tanda tangan inilah yang dapat membedakan tanda tangan pada setiap orang (Widodo & Harjoko, 2015). Ciri tersebut dapat diekstraksi dari citra tanda tangan yang nantinya digunakan untuk membantu proses verifikasi tanda tangan. Salah satu metode ekstraksi ciri berbasis pada arah dalam pengolahan citra adalah *Histogram of Oriented Gradient*s (HOG). Metode ini pertama kali dikembangkan oleh Dalal & Triggs pada tahun 2005 untuk deteksi objek manusia, ekstraksi ciri objek tanda tangan pada citra sendiri sama dengan ekstraksi ciri citra manusia, ekstraksi ciri objek tanda tangan maupun objek manusia hanya berdasar dari ciri geometris atau ciri bentuk objek pada citra.

Untuk proses verifikasi citra tanda tangan diperlukan metode klasifikasi untuk membedakan antara tanda tangan asli dan palsu. Pada penelitian yang dilakukan oleh Bailing Zhang pada tahun 2010 tentang verifikasi dan identifikasi tanda tangan secara *off-line* berdasarkan *Pyramid Histogram of Oriented Gradients*, terdapat beberapa metode klasifikasi yang dapat digunakan untuk verifikasi citra tanda tangan berdasarkan ciri HOG. Penelitian tersebut merupakan penelitian analitik yang membandingkan beberapa metode klasifikasi untuk verifikasi tanda tangan seperti *Fisher's Linear Discriminant* (FLD), *Generalized Linear Model*

(GLM), *K-Nearest Neighbor* (K-NN), *Multi-Layer Perceptron* (MLP), dan *Support Vector Machine* (SVM). Penelitian tersebut memverifikasi citra tanda tangan yang didapat dari *scanner* dan menyimpulkan bahwa metode klasifikasi SVM mempunyai tingkat akurasi yang paling tinggi (Zhang, 2010).

Berdasarkan uraian tersebut penelitian ini menggunakan ekstraksi ciri *Histogram of Oriented Gradients* dan metode klasifikasi *Support Vector Machine* dalam mengembangkan sistem verifikasi citra tanda tangan. Diharapkan penelitian ini menghasilkan sistem yang memiliki tingkat akurasi yang tinggi dalam verfikasi tanda tangan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang sudah ada, dirumuskan masalah sebagai berikut.

1. Bagaimana mengimplementasikan metode *Support Vector Machine* berdasarkan ciri *Histogram of Oriented Gradients* untuk verifikasi citra tanda tangan?
2. Bagaimana tingkat akurasi dari sistem verifikasi citra tanda tangan menggunakan metode *Support Vector Machine* berdasarkan ciri *Histogram of Oriented Gradients*?
3. Berapa lama waktu proses yang dibutuhkan sistem verifikasi citra tanda tangan untuk melatih data klasifikasi dan melakukan verifikasi?

1.3 Tujuan Penelitian

Tujuan yang ingin dicapai pada penelitian ini adalah:

1. Menerapkan metode *Support Vector Machine* berdasarkan ciri *Histogram of Oriented Gradients* sebagai verifikasi citra tanda tangan.
2. Menguji tingkat akurasi dari sistem verifikasi citra tanda tangan menggunakan metode *Support Vector Machine* berdasarkan ciri *Histogram of Oriented Gradients*.
3. Mengetahui lama waktu proses sistem verifikasi citra tanda tangan untuk melatih data klasifikasi dan melakukan verifikasi.

1.4 Manfaat Penelitian

Dengan diadakan penelitian ini maka manfaat yang diharapkan dari penelitian ini adalah:

1. Sebagai salah satu solusi dalam mengatasi penyalahgunaan atau pemalsuan tanda tangan.
2. Sebagai pengukur akurasi metode SVM dalam melakukan verifikasi tanda tangan berdasarkan hasil yang diperoleh dari pengujian.
3. Manfaat teoritis sebagai literatur pembanding untuk penelitian pada bidang pengolahan citra digital dan pengenalan pola di masa mendatang.

1.5 Batasan Masalah

Agar pembahasan pada penelitian ini lebih terarah, maka permasalahan akan dibatasi terhadap hal-hal berikut:

1. Penelitian ini menggunakan tanda tangan Indonesia dengan mengambil secara langsung dari partisipan.
2. Penelitian ini hanya melakukan verifikasi untuk tanda tangan palsu atau asli dan tidak melakukan identifikasi pemilik tanda tangan dari berbagai tanda tangan.
3. Parameter yang digunakan untuk mengukur akurasi metode adalah FAR dan FRR.

1.6 Sistematika Pembahasan

Pada sistematika pembahasan, memuat struktur penulisan dan deskripsi singkat dari masing-masing bab, antara lain:

BAB I **Pendahuluan**

Membahas tentang latar belakang, rumusan masalah, tujuan dan manfaat dari penelitian, batasan masalah, serta sistematika pembahasan dari penelitian ini.

BAB II **Landasan Kepustakaan**

Membahas tentang studi literatur mengenai penelitian yang sudah dilakukan dan menguraikan dasar teori dan teori penunjang yang berkaitan dengan pengolahan citra, *Histogram of Oriented Gradients*, metode klasifikasi *Support Vector Machine*, serta perangkat-perangkat yang digunakan dalam sistem.

BAB III **Metodologi**

Membahas tentang metode atau langkah-langkah yang digunakan pada penelitian ini, meliputi studi literatur, analisis kebutuhan, perancangan dan implementasi sistem, hingga proses pengujian dan analisis sistem.

BAB IV **Rekayasa Kebutuhan**

Membahas gambaran umum dari sistem, lingkungan operasional sistem, batasan sistem, kebutuhan fungsional, kebutuhan perangkat keras, dan kebutuhan perangkat lunak.

BAB V **Perancangan dan Implementasi**

Membahas tentang perancangan perangkat keras dan perangkat lunak sistem, serta perancangan metode yang

digunakan sistem. Proses implementasi sistem dilakukan berdasarkan perancangan yang dilakukan hingga sistem.

BAB VI Pengujian

Membahas tentang langkah-langkah pengujian dan hasil pengujian setelah dilakukan. Pada setiap pengujian dilakukan analisis terhadap hasil pengujian.

BAB VII Penutup

Memuat kesimpulan yang didapatkan berdasarkan keseluruhan proses pengerjaan penelitian, serta saran yang berguna bagi pengembangan penelitian ini selanjutnya.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan membahas tentang penelitian-penelitian yang telah dilakukan yang mempunyai kesamaan topik dengan penelitian ini. Bab ini juga memuat dasar teori yang membahas teori, konsep, metode pendukung dari berbagai sumber pustaka yang terkait dengan penelitian ini.

2.1 Tinjauan Pustaka

Penelitian ini mengacu pada penelitian yang dilakukan oleh Bailing Zhang pada tahun 2010 yang berjudul "*Off-Line Signature Verification and Identification by Pyramid Histogram of Oriented Gradients*". Penelitian tersebut menggunakan ekstraksi ciri *Pyramid of Histogram Oriented Gradient* yang sederhana dan efisien untuk ektraksi ciri citra tanda tangan. Penelitian tersebut juga membandingkan akurasi dari beberapa metode klasifikasi yang digunakan dalam proses verifikasi citra tanda tangan, metode klasifikasi yang dibandingkan yaitu *Generalized Linear Model* (GLM), *Fisher's Linear Discriminant* (FLD), *K-Nearest Neighbor* (K-NN), *Multi-Layer Perceptron* (MLP), dan *Support Vector Machine* (SVM). Untuk menghitung akurasi, penelitian tersebut menggunakan parameter FRR (*False Rejection Rate*) dan FAR (*False Acceptance Rate*). Hasil akurasi metode klasifikasi dapat dilihat pada Gambar 2.1

Classifier	GLM (%)	FLD (%)	ANN (%)	MLP (%)	SVM (%)
FRR	29.03	6.06	8.57	7.26	4.0
FAR	17.77	3.90	2.64	4.09	3.25
Accuracy	76.62	95.01	94.39	94.33	96.38

Gambar 2.1 Perbandingan akurasi dari beberapa metode klasifikasi
Sumber: (Zhang, 2010)

Dari Gambar 2.1 menunjukkan bahwa klasifikasi SVM menghasilkan akurasi yang tertinggi dibandingkan metode klasifikasi lainnya sebesar 96.38% dengan FRR 4% dan FAR 3.25% dengan menggunakan dataset GPDS dan DAVAB (Zhang, 2010). Penelitian tersebut merupakan penelitian analitik yang membedakan beberapa metode klasifikasi dalam melakukan verifikasi citra tanda tangan. Hasil dari penelitian tersebut menjadi alasan penulis untuk memilih metode klasifikasi SVM untuk verifikasi citra tanda tangan.

Pada penelitian yang dilakukan oleh Widodo dan Harjoko yang berjudul "Sistem Verifikasi Tanda Tangan *Off-Line* Berdasar Ciri *Histogram of Oriented Gradient* (HOG) dan *Histogram of Curvature* (HoC)", penelitian tersebut membuat sistem verifikasi citra tanda tangan secara *off-line*. Sistem verifikasi secara *off-line* hanya bekerja dengan pemrosesan citra tandatangan yang dapat diperoleh dari suatu mesin pemindai atau kamera digital, salah satu keuntungan verifikasi secara *off-line* yaitu masih terdapatnya sejumlah besar formulir check perbankan, formulir kepemilikian kartu kredit, atau dokumen-dokumen resmi yang masih ditanda tangani secara manual setiap harinya. Pada penelitian

tersebut ciri citra tanda tangan dapat dikatakan serupa dengan ciri karakter tulisan tangan yang keduanya tersusun atas beragam garis dan lengkungan (*curve*) yang memiliki arah atau orientasi. Pola lengkungan dan arah ini menjadi pendukung ciri gradien dan *curvature* yang diimplementasikan dengan metode Histogram of Oriented Gradients (HOG) dan Histogram of Curvature (HoC) untuk verifikasi citra tanda tangan di penelitian tersebut. Penelitian tersebut menyatakan bahwa kombinasi dari ciri HOG dan HoC yang diujikan dapat digunakan untuk proses verifikasi tanda tangan *offline* dengan metode klasifikasi *Support Vector Machine* (SVM). Penelitian tersebut juga menyimpulkan bahwa histogram lokal mampu memberikan nilai performa yang baik dalam hal melakukan verifikasi tanda tangan asli bila dibandingkan histogram global dan cakupan area citra (ukuran sel) di mana HOG dan HoC dihitung sangat menentukan kinerja sistem verifikasi (Widodo & Harjoko, 2015). Perbedaan penelitian tersebut dengan penelitian penulis yaitu pada penelitian tersebut selain menggunakan ciri *Histogram of Oriented Gradients* juga menggunakan ciri *Histogram of Curvature*, selain itu pada penelitian tersebut metode akuisisi citra tanda tangan dilakukan dengan pemindaian dokumen, akuisisi dengan cara memindai dokumen dinilai penulis lebih memakan waktu daripada dengan mengambil citra tanda tangan dengan kamera.

Penelitian ketiga dilakukan oleh (Harfiya, et al., 2017) yang berjudul "Verifikasi Citra Tanda Tangan Berdasarkan Ciri *Pyramid Histogram of Oriented Gradients* (PHOG) Menggunakan Metode Klasifikasi *K-Nearest Neighbor*", penelitian tersebut menyebutkan bahwa *Pyramid Histogram of Oriented Gradients* (PHOG) merupakan ciri yang dapat membuat proses ekstraksi dan seleksi lebih efisien dan meningkatkan tingkat pengenalan. Penelitian tersebut menggunakan metode klasifikasi K-NN dalam melakukan proses verifikasi citra tanda tangan dan akurasinya diukur dengan parameter FRR dan FAR. Hasil dari penelitian tersebut menyatakan bahwa jumlah data latih mempengaruhi nilai akurasi verifikasi, jika semakin banyak jumlah data latih dari penanda tangan maka semakin meningkat kan nilai akurasi. Penelitian ini juga menyatakan jumlah data latih yang banyak justru memperbanyak variasi bentuk tanda tangan pada seorang penanda tangan dan dapat menurunkan akurasi verifikasi tanda tangan palsu. (Harfiya, et al., 2017). Perbedaan penelitian tersebut dengan penelitian penulis yaitu penelitian tersebut menggunakan metode klasifikasi K-NN, penelitian tersebut menyatakan bahwa walaupun algoritma K-NN yang diterapkan telah berhasil melakukan pengenalan dengan baik, namun masih belum cukup baik membedakan antara yang asli dengan yang palsu.

Berdasarkan penelitian-penelitian di atas, dapat diketahui bahwa penelitian mengenai verifikasi citra tanda tangan berdasarkan ciri *Histogram of Oriented Gradients* sudah pernah dilakukan. Ringkasan dari penelitian-penelitian tersebut dan perbedaan dengan penelitian penulis dapat dilihat pada Tabel 2.1.

Tabel 2.1 Perbedaan penelitian ini dengan penelitian sebelumnya

No	Judul	Penelitian Sebelumnya	Perbedaan
1	*Off-Line Signature Verification and Identification by Pyramid Histogram of Oriented Gradients* (Zhang, 2010)	Analisis perbandingan berbagai metode klasifikasi untuk verifikasi tanda tangan berdasarkan *Pyramid Histogram of Oriented Gradients*	Metode Klasifikasi SVM dan feature descriptor HOG diimplementasikan secara langsung pada Raspberry Pi
2	Sistem Verifikasi Tanda Tangan Off-Line Berdasar Ciri *Histogram of Oriented Gradient* (HOG) dan *Histogram of Curvature* (HoC) (Widodo & Harjoko, 2015)	Menggunakan *feature descriptor Histogram of Oriented Gradients* dan *Histogram of Curvature* dengan metode klasifikasi SVM untuk verifikasi citra tanda tangan	Hanya menggunakan *Histogram of Oriented Gradients* sebagai *feature descriptor* dan akuisisi citra tanda tangan dengan cara mengambil langsung menggunakan kamera
3	Verifikasi Citra Tanda Tangan Berdasarkan Ciri Pyramid *Histogram of Oriented Gradient* (PHOG) Menggunakan Metode Klasifikasi *K-Nearest Neighbor* (Harfiya, et al., 2017)	Menggunakan *feature descriptor Histogram of Oriented Gradients* dan metode klasifikasi K-NN untuk verifikasi citra tanda tangan	Menggunakan SVM sebagai metode klasifikasinya

2.2 Dasar Teori

Pada subbab dasar teori ini akan dibahas teori-teori pendukung dan referensi yang berkaitan dan digunakan dalam penelitian ini. Berikut pembahasan dari dasar teori yang digunakan.

2.2.1 Tanda Tangan

Tanda tangan atau dikenal dengan paraf merupakan alat identifikasi personal yang berupa tulisan tangan, tanda tangan umumnya digunakan pada dokumen sebagai suatu bukti dari identitas atau persetujuan seseorang. Seperti pada Gambar 2.2 tanda tangan umumnya berbentuk gaya tulisan tertentu dari nama seseorang atau tanda identifikasi lainya.

Gambar 2.2 Contoh sebuah tanda tangan

Tanda tangan pada suatu dokumen berfungsi identifikasi atau menentukan kebenaran ciri-ciri dari penandatangan, sekaligus penandatangan setuju atau menjamin isi yang tercantum dalam surat tersebut. Menurut (Harahap, 2005) terdapat berbagai bentuk tanda tangan yang dapat diterima oleh hukum sebagai berikut:

1. Penanda tangan menuliskan nama.
2. Penulisan nama kecil untuk tanda tangan diangkap cukup.
3. Penanda tangan menulis sendiri dan tidak menggunakan stempel huruf cetak.
4. Orang yang mendapat kuasa dari pemilik tanda tangan dapat mencantumkan kopi tanda tangan dari pemberi kuasa.
5. Dapat menggunakan karbon untuk mencantumkan tanda tangan.

2.2.2 Pengolahan Citra Digital

Citra digital adalah bentuk representasi dari suatu gambar dengan bantuan alat atau mesin dengan teknik sampling dan kuantisasi. Citra digital terdiri dari titik-titik warna yang disusun secara dua dimensi dalam baris dan kolom. Titik-titik warna tersebut disebut piksel, teknik *sampling* menentukan ukuran piksel dalam citra digital. Setiap piksel pada citra digital mempunyai tingkat kecerahan yang dipengaruhi oleh kuantisasi, tingkat kecerahan ini direpresentasikan dengan derajat keabuan (Basuki & Ramadijanti, 2005). Menurut (Sutoyo & Mulyanto, 2009), cara penyimpanan citra digital pada memori menentukan jenis citra yang terbentuk, jenis citra digital yang umum digunakan antara lain:

1. Citra Biner.

   Citra biner direpresentasikan dengan dua nilai yaitu 0 dan 1, yaitu 0 untuk warna hitam dan 1 untuk warna putih.
2. Citra *Grayscale*

   Citra *grayscale* direpresentaikan dengan gradasi warna hitam dan putih, semakin besar bit yang dimiliki, maka semakin bagus juga gradasi yang ditampilkan
3. Citra Warna.

   Setiap piksel citra terdiri dari kombinasi dari tiga warna dasar yaitu merah, hijau dan biru atau umum disebut RGB. Setiap warnanya mempunyai 255 nilai intensitas.

Pengolahan citra digital adalah proses komputasi piksel pada citra digital yang bertujuan untuk memperbaiki citra atau mengambil informasi yang ada dalam citra. Informasi yang didapat dari citra digital dapat digunakan untuk berbagai bidang *computer vision*, seperti pengenalan ciri biologis yang ada pada manusia, pengenalan pola, deteksi objek pada citra dan lain-lain. (Basuki & Ramadijanti, 2005).

2.2.3 Histogram of Oriented Gradients

Setiap tanda tangan tersusun dari kombinasi garis dan lengkungan yang memiliki arah atau orientasi, ciri inilah yang membedakan tanda tangan satu dengan tanda tangan yang lain. Ciri geometris ini dapat diekstraksi dengan metode *Histogram of Oriented Gradients* (HOG), *Histogram of Oriented Gradients* merupakan sebuah metode fitur deskriptor yang banyak digunakan dalam pengolahan citra untuk bertujuan mendeteksi sebuah objek. Fitur deskriptor merepresentasikan sebuah citra dengan mengekstraksi informasi yang penting dari citra. Fitur deskriptor citra pada HOG direpresentasikan sebagai distribusi (*histogram*) dari arah *gradient* pada citra.

Metode *Histogram of Oriented Gradients* dapat diartikan bahwa ciri visual dan bentuk lokal dari objek didalam citra dapat direpresentasikan menggunakan distribusi gradien pada sebuah histogram (Widodo & Harjoko, 2015). Penerapan dari feature descriptor ini yaitu dengan membagi citra menjadi daerah-daerah kecil berukuran n x n, daerah ini yang disebut sel. Pada setiap sel pada masing-masing blok dihitung *magnitude* dan orientasi gradient-nya (Ilmi, et al., 2015). Untuk dapat menghitung ciri HOG dari sebuah citra maka langkah-langkah yang dilakukan adalah sebagai berikut (Harfiya, et al., 2017):

1. Untuk menghitung deskriptor HOG, proses pertama yaitu menghitung horizontal dan vertikal gradien, proses ini dapat dicapai dengan menggunakan operator sobel. Operator dapat dihitung dengan mengkonvolusi citra dengan kernel G_x untuk perubahan horizontal, dan G_y untuk perubahan vertikal, seperti berikut.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \#(2.1)$$

   Dengan magnitude gradien dapat dihitung dengan menggunakan persamaan 2.2:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \#(2.2)$$

   Dan orientasi gradien dihitung dengan persamaan 2.3:

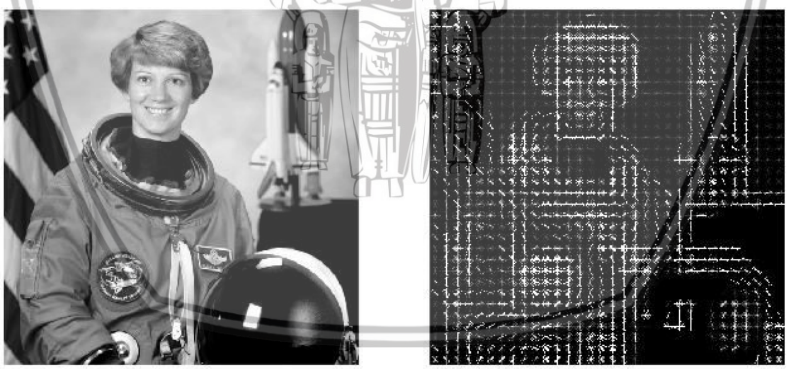
$$\theta = arctan \frac{G_y}{G_x} \#(2.3)$$

2. Menghitung *histogram of oriented gradients* pada setiap sel, langkah kedua perhitungan melibatkan pembuatan sel histogram. Citra dibagi menjadi

beberapa sel. Piksel yang ada dalam sel dihitung nilai magnitude dan orientasinya. Nilai tersebut akan disajikan dalam bentuk histogram dengan rentang orientasi antara 0^o hingga 180^o

3. Menghitung histogram *magnitude* dan orientasi gradien, mengisi histogram pertama dengan magnitude gradien yang memiliki orientasi yang sama dengan nilai bin histogram. Dengan cara yang sama, histogram kedua akan diisi dengan magnitude gradien yang memiliki orientasi yang sama juga.
4. Normalisasi fitur HOG agar independen dari kondisi pencahayaan dan mendapatkan nilai deskriptor HOG akhir

Pembentukan ciri dari metode HOG pada masing masing sel yaitu dengan mengakumulasi piksel yang memiliki nilai orientasi yang sama. Di mana orientasi gradien tersebut akan dikelompokkan menjadi beberapa bagian yang disebut bin, yang kemudian direpresentasikan dengan histogram. Pada penelitian sebelumnya yang dilakukan oleh (Dalal & Triggs, 2005), jumlah bin yang digunakan untuk membagi orientasi minimal berjumlah 8 bin untuk mendapatkan hasil deteksi terbaik. Kemudian pada penelitian (Kobayashi, et al., 2008) menyatakan bahwa jumlah bin orientasi sebesar 9 dapat menghasilkan deteksi yang lebih optimal. Kontribusi piksel terhadap tiap bin bergantung pada nilai gradien magnitude-nya. Nilai-nilai dari seluruh bin dari tiap cell dimasukkan kedalam vektor. Vektor inilah yang menggambarkan fitur HOG dari suatu gambar. Pada Gambar 2.3 di bawah merupakan sebuah ilustrasi hasil fitur HOG dari sebuah citra dengan obyek manusia.



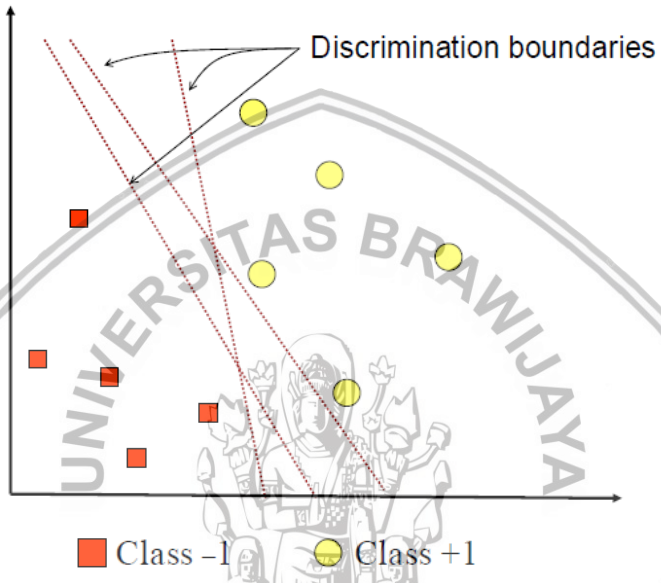
Gambar 2.3 Contoh ilustrasi fitur HOG
Sumber: (Miller, 2017)

2.2.4 Support Vector Machine

Setelah didapatkan fitur HOG, fitur tersebut selanjutnya akan masuk dalam proses pelatihan data dengan klasifikasi, metode klasifikasi yang akan dipakai dalam penelitian ini yaitu *Support Vector Machine* (SVM). Dalam bidang *machine learning*, metode SVM merupakan salah satu metode yang umum digunakan untuk klasifikasi. SVM pertama kali dikembangkan pada tahun 1992 di *Annual Workshop on Computational Learning Theory* oleh Boser, Guyon, dan Vapnik. Sederhananya konsep dari metode SVM merupakan proses mencari *hyperplane*

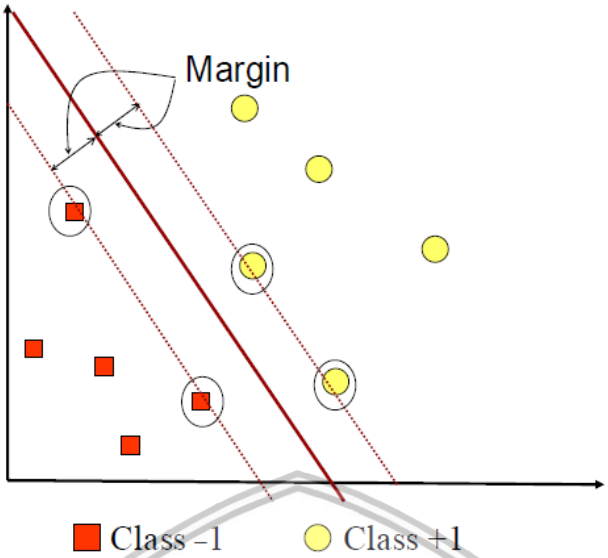
terbaik untuk memisahkan dua buah kelas pada input space (Nugroho, et al., 2003).

Pada Gambar 2.4 di bawah menunjukkan sebaran data dari dua kelas pada suatu feature space, yaitu kelas positif dan kelas negatif. Data dengan lingkaran kuning termasuk pada kelas negatif sedangkan data dengan bentuk kotak merah termasuk pada kelas negatif. Klasifikasi untuk memisahkan dua buah kelas dapat dicapai dengan mencari garis pemisah (*discrimination boundaries*). Contoh berbagai garis pemisah yang memisahkan kelas positif dan negatif ditunjukkan pada Gambar 2.4 di bawah.



Gambar 2.4 Dua buah *class* dengan garis pemisah
Sumber: (Nugroho, et al., 2003)

Hyperplane adalah garis pemisah terbaik yang memisahkan dua buah kelas pada klasifikasi SVM. Hyperplane dapat dicari dengan menghitung jarak maksimal dengan margin. Margin merupakan jarak antara *hyperplane* dengan *support vector. Support vector* disini adalah data yang paling dekat dengan kelas yang berbeda. Pada Gambar 2.5 merupakan *hyperplane* terbaik yang ditunjukkan pada garis tebal. *Hyperplane* ini yang membagi dua kelas tepat ditengah. Untuk data yang dilingkari adalah support vector. Proses untuk mencari *hyperplane* yang memisahkan dua buah kelas ini merupakan inti dari proses klasifikasi SVM (Nugroho, et al., 2003).



Gambar 2.5 Klasifikasi SVM dengan Hyperplane terbaik
Sumber: (Nugroho, et al., 2003)

2.2.5 Image Thresholding

Image Thresholding merupakan salah satu metode untuk segmentasi citra yang sederhana, segmentasi dilakukan untuk bertujuan memisahkan objek dengan background pada sebuah citra digital. Umumnya thresholding diterapkan pada *citra grayscale,* dan akan menghasilkan citra biner. *Thresholding* dilakukan dengan cara memberi nilai 1 pada nilai piksel yang melewati batas threshold, dan nilai 0 untuk nilai piksel di bawah threshold. Citra hasil *thresholding* dapat digunakan untuk pengenalan objek atau ekstraksi fitur. Secara matematis *thresholding* dapat ditulis seperti ditunjukkan pada persamaan 2.4.

$$g(x,y) = \begin{cases} 1 & if\ f(x,y) > T \\ 0 & if\ f(x,y) \leq T \end{cases} \#(2.4)$$

Keterangan:
$g(x,y)$ = Nilai piksel hasil segmentasi
$f(x,y)$ = Nilai piksel citra masukan
T = Nilai *threshold*

Terdapat dua kategori metode *thresholding* dalam menentukan nilai *threshold*, yaitu *global tresholding* dan *local thresholding*. Global Thresholding mengkonversi seluruh piksel pada citra menjadi hitam dan putih dengan satu nilai *threshold* yang telah ditentukan. Sedangkan untuk *local thresholding,* nilai *threshold* didapat dengan membagi citra menjadi blok citra, dan pada blok citra tersebut dihitung dengan nilai threshold yang berbeda (Jalil, 2015), Nilai *local thresholding* dapat dicapai dengan menggunakan *thresholding mean* ataupun *thresholding median* seperti pada persamaan 2.5 dan 2.6 di bawah.

$$T1 = mean\ f(x,y); (x,y) \in W \#(2.5)$$

$$T2 = median\ f(x,y); (x,y) \in W \#(2.6)$$

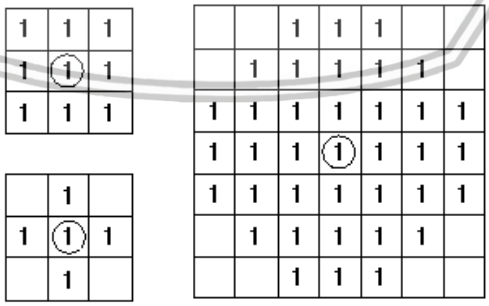
Keteragan:
$T1$ = Nilai local thresholding mean
$T2$ = Nilai local thresholding median
$f(x.y)$ = Nilai piksel masukan
W = Blok citra yang diproses

Dalam penelitian ini segmentasi dengan *thresholding* digunakan dalam bagian *preprocessing* citra tanda tangan yang ditangkap kamera. Sebelum ekstraksi ciri HOG dapat dilakukan, citra terlebih dulu disegmentasi untuk memisahkan objek tanda tangan dengan *background*. Dikarenakan iluminasi cahaya pada background citra tanda tangan berbeda-beda, pada penelitian ini menggunakan *local thresholding* untuk proses segementasinya.

2.2.6 Morphological Transformations

Morphological Transformations merupakan pengoperasian citra secara geometris menggunakan teori himpunan. Metode ini memerlukan dua *input*, *input* pertama yaitu citra yang akan diproses, dan *input* kedua adalah *structure element* atau kernel yang digunakan untuk menentukan bagaimana operasi morfologi dilakukan. Citra yang menjadi masukkan umumnya citra biner yang hanya memiliki nilai 0 dan 1. Sedangkan *Structure element* merupakan sebuah *template* bentuk yang telah didefinisikan sebelumnya.

Structure element ini akan diimplementasikan pada gambar biner dimana nilai 1 pada structure element merepresentasikan bentuk yang akan dioperasikan dan umumnya memiliki ukuran yang lebih kecil dari citra masukkan. Structure element dapat berbagai bentuk atau berbagai ukuran, beberapa contoh bentuk yang umum digunakan yaitu *rectangle*, *elips* atau *disk*, dan *cross-shaped* seperti yang ditunjukkan pada Gambar 2.6. Pada gambar tersebut *structure element* mempunyai titik awal yang berada di tengah *structure element* (Fisher, et al., 2003).



Gambar 2.6 Beberapa contoh *structuring elements*
Sumber: (Fisher, et al., 2003)

Structure element diimplementasikan pada citra masukkan dengan menggunakan operasi himpunan *union* untuk *dilation*, dan *intersection* untuk *erosion*. Dari kombinasi *dilation* dan *erosion* didapatkan operasi lain seperti

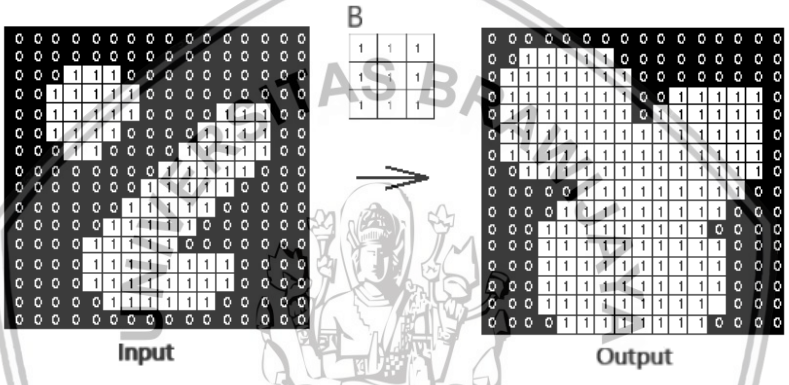
opening dan *closing* (Fisher, et al., 2003). Berikut penjelasan dari setiap operasi morfologi.

1. *Dilation*

Operasi *dilation* akan membuat bentuk geometris citra keluaran menjadi terlihat membesar atau menebal. *Dilation* menggunakan operasi union antara citra *input* A dengan *structure element* B, yakni seperti ditunjukkan pada persamaan (2.7).

$$A \oplus B = \bigcup_{b \in B} A_b \#(2.7)$$

Contoh hasil dari penerapan operasi *dilation* pada input *biner* dengan *structure element* yang berbentuk *rectangle* dengan ukuran 3x3 dapat dilihat pada Gambar 2.7.



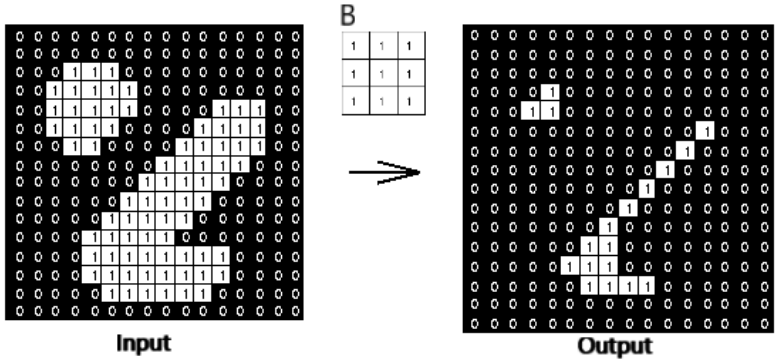
Gambar 2.7 Hasil operasi *dilation*
Sumber: (Fisher, et al., 2003)

2. *Erosion*

Operasi *erosion* akan membuat bentuk geometris citra keluaran menjadi terlihat menipis atau mengecil. *Erosion* menggunakan operasi *intersection* antara citra *input* A dengan *structure element* B, yakni seperti ditunjukkan pada persamaan (2.7).

$$\mathrm{A} \ominus \mathrm{B} = \bigcap_{a \epsilon B} A_{-b} \#(2.8)$$

Contoh hasil dari penerapan operasi *erosion* pada input *biner* dengan *structure element* yang berbentuk *rectangle* dengan ukuran 3x3 dapat dilihat pada Gambar 2.8.



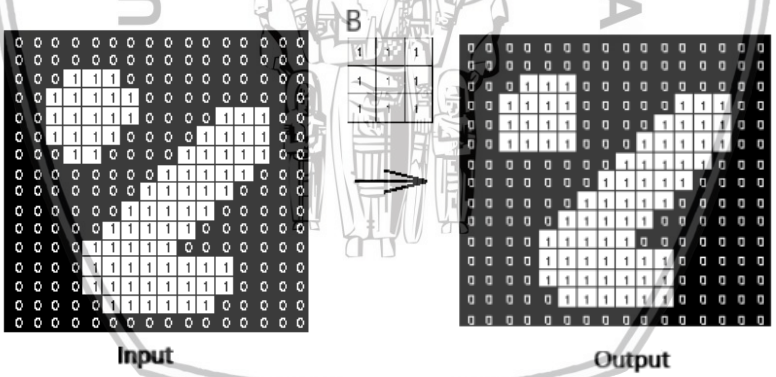
Gambar 2.8 Hasil operasi *erosion*
Sumber: (Fisher, et al., 2003)

3. Opening

Opening adalah operasi *morphology* yakni *dilation* yang diikuti dengan *erosion* Seperti yang ditujukkan pada persamaan 2.9. *Opening* akan membuat sudut objek pada citra terlihat lebih halus.

$$A \circ \mathrm{B} = (A \ominus B) \oplus \mathrm{B} \#(2.9)$$

Contoh hasil dari penerapan operasi *opening* pada input *biner* dengan *structure element* yang berbentuk *rectangle* dengan ukuran 3x3 dapat dilihat pada Gambar 2.9.



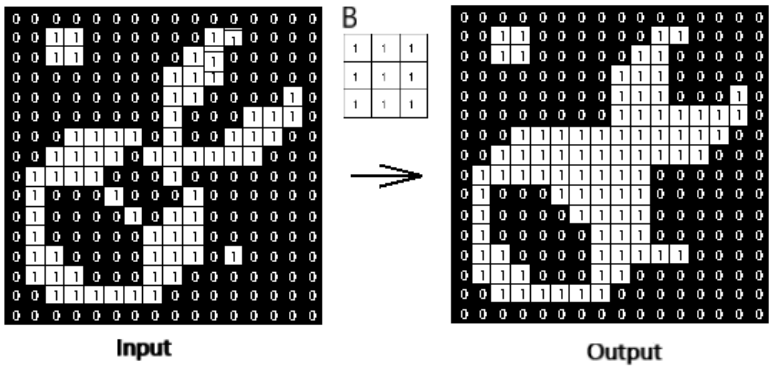
Gambar 2.9 Hasil operasi *opening*
Sumber: (Fisher, et al., 2003)

4. Closing

Closing adalah operasi *morphology* yakni *erosion* yang diikuti dengan *dilation* Seperti yang ditujukkan pada persamaan 2.10. *Closing* akan mengisi atau menghilangkan lubang-lubang hitam pada citra.

$$A \circ \mathrm{B} = (A \oplus B) \ominus \mathrm{B} \#(2.10)$$

Contoh hasil dari penerapan operasi *closing* pada input *biner* dengan *structure element* yang berbentuk *rectangle* dengan ukuran 3x3 dapat dilihat pada Gambar 2.10.



Gambar 2.10 Hasil operasi *closing*
Sumber: (Fisher, et al., 2003)

Morphological Trasnformations pada penelitian ini digunakan untuk menghilangkan *noise* berupa titik-titik hitam pada *background* citra tanda tangan, hal ini dapat dicapai dengan metode morfologi *closing*. Pada sistem, metode morfologi ini akan masuk pada tahap *preprocessing* sebelum dihitung ciri HOG citranya.

2.2.7 Parameter FAR dan FRR

Dalam skenario verifikasi citra tanda tangan yang diverifikasi oleh sistem, akan direspons oleh sistem berupa respons positif atau negatif. Dari sudut pandang klasifikasi pola, tujuan sistem verifikasi tanda tangan adalah untuk membedakan antara dua kelas, yaitu asli dan palsu (Zhang, 2010).

Umumnya akurasi untuk sistem verifikasi diukur berdasarkan parameter FAR dan FRR. Untuk penjelasan lebih lanjut dari FAR dan FRR sebagai berikut (Prakash, 2013).

1. *False Acceptance Rate* (FAR): Didefinisikan sebagai persentase dari kandidat yang diterima akibat kesalahan sistem. Dengan kata lain, FAR adalah tingkat di mana suatu tanda tangan palsu diterima secara salah atau dianggap sistem sebagai tanda tangan asli atau data asli. FAR didefinisikan sebagai berikut:

$$False\ Acceptance\ Rate = \frac{Jumlah\ False\ Acceptance}{Jumlah\ tanda\ tangan\ palsu} \times 100\%\#(2.11)$$

2. *False Rejection Rate* (FRR): Didefinisikan sebagai persentase dari kandidat yang ditolak akibat kesalahan sistem. Dengan kata lain, FRR adalah tingkat di mana suatu tanda tangan asli ditolak secara salah atau dianggap sistem sebagai sebuah tanda tangan palsu data palsu. FRR didefinisikan sebagai berikut:

$$False\ Rejection\ Rate = \frac{Jumlah\ False\ Rejection}{Jumlah\ tanda\ tangan\ asli} \times 100\%\#(2.12)$$

3. Untuk mendapatkan persentase akurasi dari parameter FAR dan FRR diatas dapat dihitung dengan persamaan 2.13 berikut.

$$Akurasi = \left(100\% - \frac{FAR + FRR}{2}\right)\%\#(2.13)$$

2.2.8 Raspberry Pi

Raspberry Pi atau biasa disebut Raspi dengan bentuk seperti Gambar 2.11 adalah komputer papan tunggal (*single-board circuit*) yang dikembangkan di Britaina Raya oleh *Raspberry Pi Foundation* untuk mengenalkan dasar dari ilmu komputer di sekolah dan di negara berkembang. Ukuran Raspberry Pi sebesar kartu kredit dan memiliki *input output* digital port seperti pada sebuah board *microcontroller*. Kelebihan dari Raspberry Pi dibanding dengan *board microcontroller* yaitu Raspberry Pi mempunyai koneksi untuk *display* berupa TV atau Monitor PC serta koneksi USB untuk *Keyboard* serta *Mouse*. Dengan kelebihan tersebut Rasberry Pi sering digunakan untuk menjalankan penelitian untuk pengolahan citra.

Gambar 2.11 Raspberry Pi 3 model B
Sumber: (Raspberrypi.org, 2016)

Raspberry Pi dalam penelitian ini digunakan untuk menerima *input* citra dari kamera, melakukan pemrosesan citra data latih dan data uji tanda tangan, dan mengirimkan keluaran hasil verifikasi pada LCD. Untuk spesifikasi dari Rasberry Pi 3 dapat dilihat pada tabel Tabel 2.2 di bawah.

Tabel 2.2 Spesifikasi Rasberry Pi 3

Arsitektur	ARMv8-A (64/32-bit)
SoC	Broadcom BCM2837
Prosesor	1.2 GHz 64-bit quad-core ARM Cortex-A53
GPU	Broadcom VideoCore IV @ 250 MHz (BCM2837: 3D part of GPU @ 300 MHz, video part of GPU @ 400 MHz)
Memori	1 GB
USB 2.0 ports	4 (via on-board 5-port USB hub)
Video masukan	5-pin MIPI camera interface (CSI) connector
Video keluaran	HDMI (rev 1.3), composite video (3.5 mm TRRS jack),

	MIPI display interface (DSI) for raw LCD panels
On-board storage	Micro SDHC slot, USB Boot Mode
On-board network	10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1
Low-level peripherals	17× GPIO, HAT ID bus
Power ratings	300 mA (1.5 W) saat idle, 1.34 A (6.7 W) maksimum saat bekerja (monitor, keyboard, mouse, WiFi connected)
Sumber daya	5 V via MicroUSB or GPIO header
Dimensi	85.60 mm × 56.5 mm × 17 mm

Sumber: (Raspberrypi.org, 2016)

2.2.9 Raspberry Pi Camera

Modul kamera Raspberry Pi digunakan sebagai *input* sistem untuk mengambil citra tanda tangan dengan bantuan *push button* sebagai pemicunya. Modul kamera ini dihubungkan ke Raspberry Pi melalui port CSI (*Camera Serial Interface*).

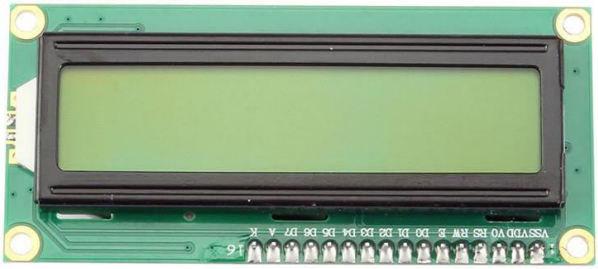
Gambar 2.12 Raspberry Pi Camera Module
Sumber: (Raspberrypi.org, 2016)

Modul kamera pada versi Rev 1.3 yang ditunjukkan pada Gambar 2.12 mempunyai resolusi sebesar 5 *megapixel* dan mampu menangkap gambar dan video sebesar 1080p. Modul Pi Camera mempunyai dimensi sekitar 25 x 20 x 9mm dan mempunyai berat sekitar 3 gram, sehingga cocok untuk aplikasi mobile. Untuk gambar statis, modul kamera ini mampu menangkap gambar beresolusi 2592 x 1944 piksel, dan untuk video, mampu merekam video beresolusi 1080p30, 720p60 dan 640x480p60/90.

2.2.10 *Liquid Crystal Display* 16x2

Liquid Crystal Display (LCD) merupakan perangkat yang umum digunakan sebagai penampil teks pada sistem elektronik. LCD dapat menampilkan teks

dalam bentuk huruf, angka, atau simbol. Bentuk dari LCD dot matrik dengan jumlah karakter 2x16 ditunjukkan pada Gambar 2.13.



Gambar 2.13 LCD 16x2
Sumber: (Makerfabs, 2018)

LCD pada peneltian ini digunakan untuk menampilkan *output* hasil verifikasi dari sistem. Output dari sistem yaitu hasil prediksi dari kelas klasifikasi, hasil tersebut akan ditampilkan pada LCD berupa teks "Asli" atau "Palsu" yang menyatakan bahwa tanda tangan yang ditangkap kamera adalah asli atau palsu. Spesifikasi yang dimiliki dari LCD 16x2 ditunjukkan pada Tabel 2.3 berikut:

Tabel 2.3 Spesifikasi LCD 16x2

Jumlah Karakter	16 karakter x 2 baris
Display Mode	STN, BLUB
Dimensi	80.0 x 36.0 x 1.6 mm
Arah Display	6 O'Clock
Display Font	5 x 8 Dot
Data Masukkan	4-Bit atau 8-Bit
Sumber daya	Single Power Supply (5V±10%)

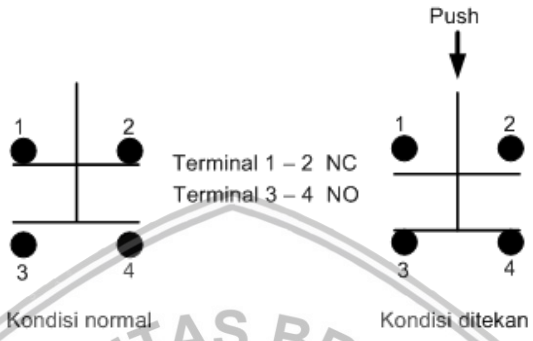
Sumber: (Shenzhen Eone Electronics, 2018)

2.2.11 Push Button

Push button atau saklar tombol tekan seperti pada Gambar 2.14 merupakan perangkat sederhana yang memiliki fungsi untuk memutus atau menghubungkan arus listrik yang sifatnya tidak mengunci. Yang dimaksud tidak mengunci yaitu *push button* hanya bekerja sebagai pemutus atau penghubung aliran arus listrik saat tombol ditekan, dan kembali ke kondisi semula saat tombol dilepas.

Gambar 2.14 *Push Button*
Sumber: (IrishElectronics, 2018)

Berdasarkan fungsinya untuk menghubungkan dan memutuskan aliran arus, *push button* mempunyai 2 jenis kontak seperti yang ditunjukkan pada Gambar 2.15 yaitu NC (*Normally Close*) dan NO (*Normally Open*). Pada *Normally Open* kondisi normal dari kontak terminal tidak dialiri arus listrik, dan saat ditekan kontak akan tertutup dan dialiri arus listrik. Sedangkan pada *Normally Close*, kondisi normal dari kontak terminal dialiri arus listrik, dan saat ditekan kontak akan terbuka dan memutus aliran arus listrik (Suprianto, 2015).



Gambar 2.15 Prinsip kerja *push button*
Sumber: (Suprianto, 2015)

Pada penelitian ini, *push button* digunakan sebagai pemicu kamera menangkap citra tanda tangan. Saat tombol ditekan, kamera akan mengambil *frame* citra dari video yang ditampilkan, *frame* inilah yang nantinya akan dikirim ke Raspberry Pi untuk diproses dan diverifikasi.

2.2.12 Keyboard

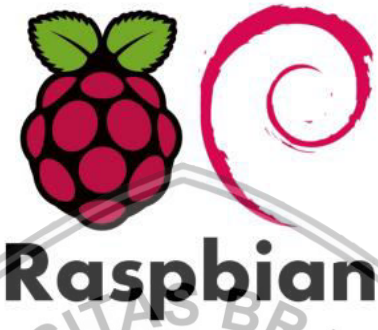
Keyboard seperti pada Gambar 2.16 merupakan perangkat keras yang berfungsi memberikan masukkan berupa karakter huruf, angka atau simbol sesuai dari tombol yang ditekan. *Keyboard* dalam penelitian ini digunakan sebagai perangkat keras sistem untuk memberi masukkan identitas nama penanda tangan. Identitas nama penanda tangan tersebut akan dipakai untuk mengatur data latih dari penanda tangan mana yang akan dilatih dan diverifikasi.

Gambar 2.16 *Keyboard*
Sumber: (Wichary, 2017)

Keyboard untuk penelitian ini dapat dihubungkan dengan Raspberry Pi melalui *port USB* atau dapat menggunakan *keyboard* dari laptop yang terhubung pada Raspberry Pi menggunakan *port ethernet*.

2.2.13 Sistem Operasi Raspbian

Sistem operasi Raspbian yang mempunyai logo pada Gambar 2.17, merupakan sistem operasi dengan basis Linux Debian yang digunakan untuk Raspberry Pi. Raspbian berisi program-program dasar dan utilitas yang dapat membantu Raspberry Pi bekerja. Rasbian mempunyai lebih dari 35 ribu *packages software* yang siap untuk diinstalasi pada Raspberry Pi. Terdapat beberapa versi Raspbian yaitu Wheezy, Jessie, dan Stretch (Raspbian, 2018).



Gambar 2.17 Logo Raspbian
Sumber: (Raspbian, 2018)

Untuk dapat memakai Raspbian, *file image* Raspbian ditulis pada sebuah microSD, kapasitas microSD yang disarankan minimal berukuran 8 GB. Setelah berhasil ditulis pada microSD, Raspbian langsung dapat dipakai pada Raspberry Pi dengan memasukkan microSD yang berisi Raspbian pada slot yang disediakan di Raspberry Pi.

2.2.14 OpenCV

OpenCV (*Open Source Computer Vision Library*) yang mempunyai logo seperti pada Gambar 2.18, merupakan *library* pengolahan citra yang dibuat oleh Intel yang membantu pemrograman untuk *computer vision* dan *machine learning*. OpenCV dapat berjalan pada bahasa pemrograman C++, Python, Java and MATLAB dan mendukung sistem operasi Linux dan Windows (OpenCV, 2018).

Gambar 2.18 Logo OpenCV
Sumber: (OpenCV, 2018)

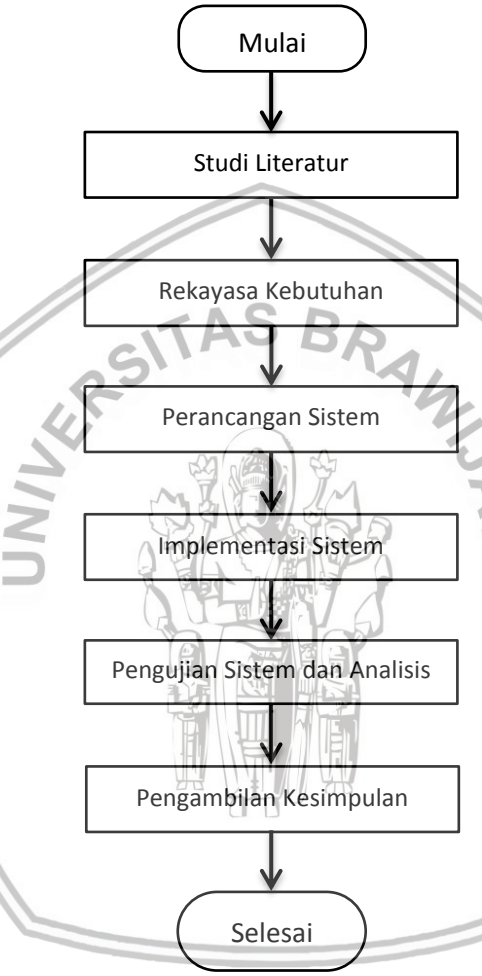
Karena sifatnya yang *open source* OpenCV banyak digunakan dalam penelitian yang mengimplementasikan *Face Recognition*, *Face Detection*, *Object Tracking*, *Road Tracking*, dan lain-lain. Dalam *library* OpenCV menyediakan

berbagai fitur yang dapat dimanfaatkan, antara lain: *Image and video* I/O, *image processing* dan *computer vision*, modul *computer vision high level*, metode untuk AI dan *machine learning,* sampling gambar dan transformasi, dan metode untuk memperhitungkan pemodelan 3D (Sidharta, 2017).

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 3 METODOLOGI

Pada bab ini membahas proses atau tahapan dalam melakukan penelitian ini. Metode penelitian dilakukan agar penelitian dapat berjalan secara terstruktur dan sistematis. Tahapan-tahapan tersebut diilustrasikan dengan diagram alir pada Gambar 3.1.



Gambar 3.1 Diagram alir metodologi penelitian

Dari Gambar 3.1 tahap metodologi yang dilakukan dalam penelitian ini yaitu dimulai dari studi literatur dari berbagai penelitian sebelumnya yang terkait, analisis kebutuhan yang meliputi kebutuhan fungsional, kebutuhan perangkat keras dan kebutuhan perangkat lunak, kemudian dilakukan tahap perancangan sistem, dilanjutkan implementasi sistem berdasarkan perancangan, pengujian sistem dan analisis, serta diakhiri dengan pengambilan kesimpulan dan pemberian saran.

3.1 Studi Literatur

Tahapan pertama yang dilakukan dalam penelitian ini yaitu studi literatur. Studi literatur digunakan untuk mencari referensi teori yang relevan untuk

menunjang penelitian ini. Referensi yang didapat bersumber dari buku, jurnal dan penelitian-penelitian sebelumnya yang berhubungan dengan penelitian ini. Beberapa teori dan bidang ilmu yang dipelajari dan dikaji antara lain:

1. Pengolahan Citra Digital
2. Histogram of Oriented Gradients
3. Metode Klasifikasi SVM (*Support Vector Machine*)
4. Penelitian sebelumnya tentang verifikasi tanda tangan

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan digunakan untuk mengidentifikasi kebutuhan apa saja yang akan diperlukan untuk merancang sistem pada penelitian ini. Hal ini dilakukan dengan mengidentifikasi seluruh kebutuhan sistem, baik pada sisi kebutuhan perangkat keras ataupun perangkat lunak. Dengan dilakukannya identifikasi dan analisis kebutuhan sistem ini diharapkan akan mempermudah dalam proses perancangan sistem di penelitian ini.

3.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan apa saja yang dapat dilakukan sistem untuk memenuhi tujuan dari suatu penelitian. Kebutuhan fungsional sistem dari penelitian ini antara lain:

1. Sistem dapat mengambil citra tanda tangan menggunakan modul kamera Raspberry Pi.
2. Sistem dapat melakukan pelatihan data citra tanda tangan.
3. Sistem dapat melakukan komputasi citra digital.
4. Sistem dapat menampilkan keluaran verifikasi pada LCD.

3.2.2 Kebutuhan Perangkat Keras

Analisis kebutuhan perangkat keras dilakukan untuk mengetahui perangkat keras apa saja yang dibutuhkan dalam pembuatan sistem di penelitian ini. Perangkat keras yang dibutuhkan antara lain:

1. Raspberry Pi 3 (Model B) sebagai perangkat utama pengolahan citra tanda tangan
2. Modul Kamera Raspberry Pi (Rev 1.3) sebagai penangkap citra tanda tangan
3. *Push Button* sebagai pemicu kamera menangkap citra
4. *Liquid Crystal Display* 16x2 sebagai penampil keluaran sistem
5. *Keyboard* sebagai masukkan sistem nama penanda tangan yang akan diverifikasi

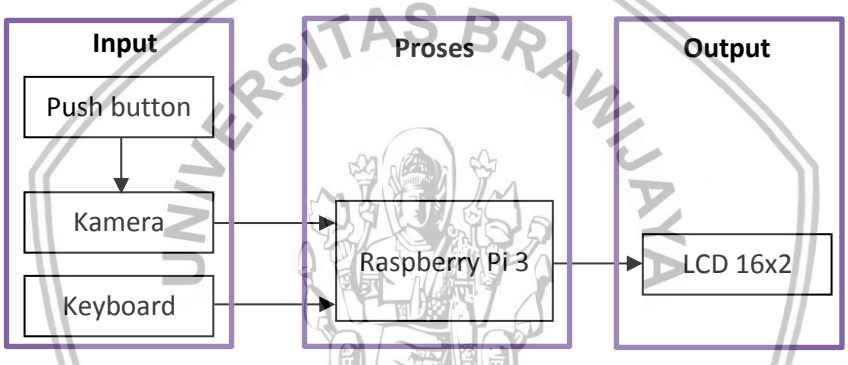
3.2.3 Kebutuhan Perangkat Lunak

Analisis kebutuhan dari perangkat lunak sistem dilakukan untuk mengetahui *software* atau program apa saja yang dibutuhkan dalam pembuatan sistem di penelitian ini. Perangkat lunak yang dibutuhkan antara lain:

1. Sistem Operasi Raspbian Stretch
2. OpenCV 3
3. Real VNC

3.3 Perancangan

Perancangan perlukan sebagai gambaran bagaimana sistem dibangun. Perancangan sistem bertujuan agar implementasi sistem dapat dilakukan secara terstruktur dan sistematis. Berikut adalah diagram blok perancangan sistem yang ditunjukkan pada Gambar 3.2.



Gambar 3.2 Diagram blok perancangan sistem

Berdasarkan diagram blok pada Gambar 3.2, *input* dari sistem menggunakan kamera untuk menangkap citra tanda tangan dengan *push button* sebagai pemicunya, *input* dari *keyboard* juga digunakan untuk memberi identitas nama penanda tangan yang diperlukan untuk memilih tanda tangan mana yang akan diverifikasi. Di dalam Raspberry Pi, data latih yang akan dimuat berdasarkan *input* identitas nama yang diberikan, data tersebut dikalkulasi ciri HOG dan masuk ke *svm classifier* untuk pelatihan data. Dalam Raspberry Pi juga memproses citra tanda tangan yang ditangkap kamera dengan *image preprocessing* sebelum dilakukan perhitungan ciri HOG. Citra tersebut dihitung ciri HOG, dan diklasifikasikan dengan *SVM classifier* apakah citra tanda tangan tersebut termasuk kelas asli atau palsu. Output menggunakan LCD yang akan menampilkan hasil dari klasifikasi citra tanda yang telah dilakukan, keluaran LCD berupa teks “Asli” atau “Palsu”.

3.4 Implementasi

Setelah dilakukan tahap perancangan, selanjutnya melakukan tahap implementasi sistem berdasarkan perancangan yang telah dibuat. Beberapa

langkah implementasi agar sistem dapat terwujud sesuai dengan tahap perancangan sebagai berikut:

1. Implementasi sistem dimulai dengan merangkai perangkat keras sistem, menggabungkan komponen perangkat keras yang dibutuhkan sistem sesuai dengan perancangan.
2. Mengimplementasi metode-metode yang dibutuhkan sistem dengan menggunakan bahasa pemrograman Phyton dan bantuan *library* OpenCV. Metode-metode yang dibutuhkan antara lain metode untuk *image preprocessing* seperti mengubah *colorspace* citra dari RGB ke *grayscale*, *adaptive thresholding*, *closing morphology*, deteksi dan koreksi kemiringan. Selain itu mengimplementasikan *feature descriptor* HOG untuk ekstraksi fitur ciri citra, dan mengimplementasikan *Support Vector Machine* sebagai metode klasifikasi.
3. Mengimplementasi *software* untuk sistem yang berupa *software* untuk pelatihan data dan *software* untuk verifikasi tanda tangan. *Software* pelatihan data digunakan untuk melatih data latih dengan klasifikasi SVM untuk membentuk fungsi pemisah SVM. Sedangkan *software* verifikasi tanda tangan digunakan untuk melakukan verifikasi citra tanda tangan yang ditangkap kamera dan menampilkan *output* hasil verifikasi citra.

3.5 Pengujian dan Analisis

Tahap ini dilakukan pengujian dan analisis dari hasil implementasi sistem yang telah dilakukan. Pengujian dilakukan untuk mengetahui apakah sistem tersebut telah berfungsi dengan baik dan sesuai dengan apa yang diinginkan. Pengujian yang akan dilakukan terdiri dari:

1. Pengujian Akurasi Verifikasi Citra Tanda Tangan

Pada pengujian ini dilakukan pengukuran tingkat akurasi dari metode klasifikasi SVM yang diterapkan dalam melakukan verifikasi citra tanda tangan.

2. Pengujian Waktu Proses Pelatihan Data

Pengujian waktu proses pelatihan data bertujuan untuk mengetahui berapa lama waktu yang dibutuhkan sistem untuk melatih data pada setiap penanda tangan.

3. Pengujian Waktu Proses Verifikasi Tanda Tangan

Pengujian waktu proses verifikasi tanda tangan bertujuan untuk mengetahui berapa lama waktu yang dibutuhkan sistem untuk melakukan verifikasi pada setiap tanda tangan.

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah proses perancangan, implementasi, dan pengujian, tahap terakhir yaitu dilakukan tahap pengambilan kesimpulan, kesimpulan diambil berdasarkan proses penelitian yang yang telah

dilakukan. Pada tahap ini dibahas tentang hasil implementasi metode HOG dan klasifikasi SVM dan tingkat akurasinya untuk verifikasi citra tanda tangan. Hasil kesimpulan pada tahap ini diharapkan menjadi bahan acuan pada penelitian mendatang. Saran di penelitian ini juga dibahas pada tahap ini agar pada penelitian mendatang diharapkan dapat mengembangkan sistem menjadi lebih baik.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 4 REKAYASA KEBUTUHAN

Bab ini membahas secara rinci tentang gambaran umum sistem, rekayasa kebutuhan yang terdiri dari kebutuhan fungsional, kebutuhan perangkat keras, dan kebutuhan perangkat lunak, hal ini diperlukan sebelum melanjutkan ke tahap perancangan dan implementasi.

4.1 Gambaran Umum Sistem

Sistem yang akan dibuat pada penelitian ini yaitu sistem verifikasi citra tanda tangan dengan menggunakan *single-board computers* Raspberry Pi 3 sebagai basis komputasinya. Sistem ini menggunakan metode Histogram of Oriented Gradients (HOG) untuk ekstraksi ciri dari citra tanda tangan dan Support Vector Machine sebagai metode klasifikasinya. Sistem ini memerlukan data latih citra tanda tangan asli dan palsu yang disimpan dalam Raspberry Pi. Input dari sistem ini berupa data uji tanda tangan yang diambil citranya menggunakan modul kamera Raspberry Pi Rev 1.3 dengan penekanan pada *push button* sebagai pemicunya. Raspberry Pi sebagai unit pemroses akan memuat data latih citra tanda tangan sesuai identitas nama yang diberikan menggunakan keyboard, menghitung ciri HOGnya, dan melakukan pelatihan data dengan klasifikasi SVM. Raspberry Pi juga memproses data latih citra tanda tangan yang ditangkap kamera dengan melakukan preprocessing, menghitung ciri HOG citra dan melakukan verifikasi berupa prediksi dari klasifikasi SVM. Hasil dari keluaran verifikasi tersebut akan ditampilkan di LCD 16x2.

4.1.1 Lingkungan Operasional Sistem

Lingkungan operasional sistem yang merupakan faktor pendukung agar sistem verifikasi citra tanda tangan berjalan maksimal yaitu ruangan dengan pencahayaan yang cukup. Tingkat pencahayaan yang dibutuhkan kamera untuk menangkap objek tanda tangan dengan jelas sebesar ±500 lux, sesuai dengan SNI 03-6575-2001 tentang sistem pencahayaan buatan pada bangunan.

4.1.2 Batasan Perancangan dan Implementasi

Dalam pembuatan sistem verifikasi citra tanda tangan terdapat batasan-batasan dalam implementasi sistem, batasan batasan tersebut yaitu:

1. Sistem melakukan verifikasi 1 tanda tangan pada setiap citra yang ditangkap kamera.
2. Sistem melakukan verifikasi tanda tangan dari 1 pendanda tangan pada setiap pengujiannya.
3. Citra tanda tangan mempunyai *background* yang berwarna putih.
4. Tanda tangan yang digunakan menggunakan tinta berwarna hitam
5. Menggunakan data latih citra tanda tangan yang telah dilakukan *preprocessing*.

6. Jarak tanda tangan dengan kamera disesuaikan agar objek tanda tangan pada citra tidak terpotong dan mengisi keseluruhan frame yang ditangkap kamera.
7. Tingkat pencahayaan ruangan minimal 500 lux agar kamera dapat menangkap citra tanda tangan dengan jelas.

4.2 Rekayasa Kebutuhan

Rekayasa kebutuhan membahas mengenai kebutuhan dari sistem secara keseluruhan dan terperinci agar sistem dapat beroperasi dengan baik, hal ini diperlukan untuk mempermudah dalam perancangan sistem. Subbab ini terdiri dari kebutuhan fungsional, kebutuhan perangkat keras, dan kebutuhan perangkat lunak.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan sistem yang wajib dipenuhi agar mencapai tujuan penelitian. Berikut kebutuhan fungsional dari sistem:

1. Sistem dapat mengambil citra tanda tangan menggunakan modul kamera Raspberry Pi

Citra tanda tangan yang diperoleh dari modul kamera digunakan sebagai sumber data input sistem, citra tersebut akan dikirim ke Raspbery Pi untuk diproses dan diverifikasi.

2. Sistem dapat melakukan pelatihan data citra tanda tangan

Data latih citra tanda tangan asli dan palsu yang ada dalam Raspberry Pi akan dihitung ciri HOG citranya, fitur ciri tersebut digunakan untuk melatih data pada SVM *classifier*. Pelatihan data ini digunakan sebagai proses pembelajaran untuk membentuk model klasifikasi dari metode SVM. Model klasifikasi tersebut merupakan representasi pengetahuan yang akan digunakan untuk membuat fungsi pemisah kelas data baru yang belum pernah ada. Proses data latih sangat penting karena nantinya sangat mempengaruhi *output* verifikasi citra tanda tangan.

3. Sistem dapat melakukan komputasi citra digital

Pada kebutuhan sistem ini, Raspberry Pi memproses citra tanda tangan masukan yang ditangkap kamera. Sebelum dilakukan ekstraksi ciri, citra tersebut terlebih dahulu melalui *preprocessing*, *preprocessing* yang dilakukan meliputi konversi *colorspace* citra dari RGB ke *grayscale*, *adaptive thresholding*, morfologi, deteksi dan koreksi kemiringan. Setelah dilakukan *preprocessing* citra, Raspberry Pi melakukan ekstraksi ciri citra tanda tangan dengan metode Histogram of Oriented Gradients. Ciri tersebut akan diklasifikasikan berdasarkan model klasifikasi yang sudah dibentuk dalam proses pelatihan data. Hasil prediksi klasifikasi yang berupa tanda tangan asli atau palsu akan dijadikan output sistem.

4. Sistem dapat menampilkan keluaran verifikasi pada LCD

Kebutuhan sistem ini digunakan untuk menampilkan output sistem hasil verifikasi. Output dalam bentuk teks ditampilkan pada LCD yang berukuran 16x2, isi teks yang ditampilkan berupa “Asli” atau “Palsu” sesuai dengan hasil dari prediksi klasifikasi.

4.2.2 Kebutuhan Perangkat Keras

Bagian ini membahas terkait dengan kebutuhan perangkat keras yang akan digunakan pada sistem ini. Perangkat keras yang dimaksud meliputi komponen fisik elektronika yang nantinya dapat memenuhi kebutuhan fungsi dan membentuk sistem ini. Kebutuhan perangkat keras yang digunakan pada proses implementasi sistem yakni sebagai berikut:

1. Raspberry Pi 3 (Model B)

Raspberry Pi 3 merupakan *single-board computer* yang memiliki prosessor berkecepatan 1.2, dan 1 GB RAM. Raspberry Pi 3 umum digunakan dalam bidang *computer vision* karena dilengkapi konektor 15 pin *Camera Serial Interface* (CSI) yang dapat digunakan dengan modul kamera Raspberry Pi. Raspberry Pi akan berfungsi sebagai unit untuk memproses citra digital.

2. Modul Kamera Raspberry Pi (Rev 1.3)

Modul Kamera Raspberry Pi Rev 1.3 memiliki resolusi gambar sebesar 5 megapiksel dan dapat merekam video hingga 1080p dengan 30 FPS. Modul kamera ini kompatibel dengan Raspberry Pi dan dapat dihubungkan melalui 15 pin *Camera Serial Interface* yang terdapat pada Raspberry Pi. Modul Kamera ini akan berfungsi untuk menangkap citra dari tanda tangan dan mengirimkan citra ke Raspberry Pi untuk diproses.

3. Push Button

Sistem ini menggunakan *push button* sebagai pemicu modul kamera untuk menangkap gambar.

4. Liquid Crystal Display 16x2

Keluaran hasil verifikasi citra tanda tangan akan ditampilkan pada LCD yang berukuran 16x2. Modul I2C (*Inter Integrated Circuit*) digunakan pada LCD untuk menghemat pemakaian pin pada Raspberry Pi.

5. Keyboard

Keyboard digunakan untuk memasukkan identitas nama dari tanda tangan mana yang akan diverifikasi.

4.2.3 Kebutuhan Perangkat Lunak

Bagian ini membahas kebutuhan perangkat lunak yang akan digunakan pada sistem ini, perangkat yang dibutuhkan sistem antara lain:

1. Sistem Operasi Raspbian Stretch

Raspbian merupakan sistem operasi berbasis Debian untuk Raspberry Pi, Raspbian digunakan agar Raspberry Pi dapat berfungsi dengan baik dan dapat menjalankan program serta utilitas. Sistem operasi Raspbian diinstall pada microSD yang ada pada Rasberry Pi.

2. OpenCV 3

OpenCV (*Open Source Computer Vision*) merupakan *library* perangkat lunak yang utamanya digunakan dalam bidang pengolahan citra secara *real-time*. OpenCV umum digunakan untuk pengolahan citra karena sifatnya yang *open source* dan memiliki banyak *library* untuk analisis gambar dan video, selain itu pemrogramannya dapat menggunakan bahasa C++, dan Python. Sistem ini membutuhkan beberapa *library* dari OpenCV seperti metode *Adaptive Thresholding*, Morfologi, HOG, SVM dan lain-lain.

3. Real VNC

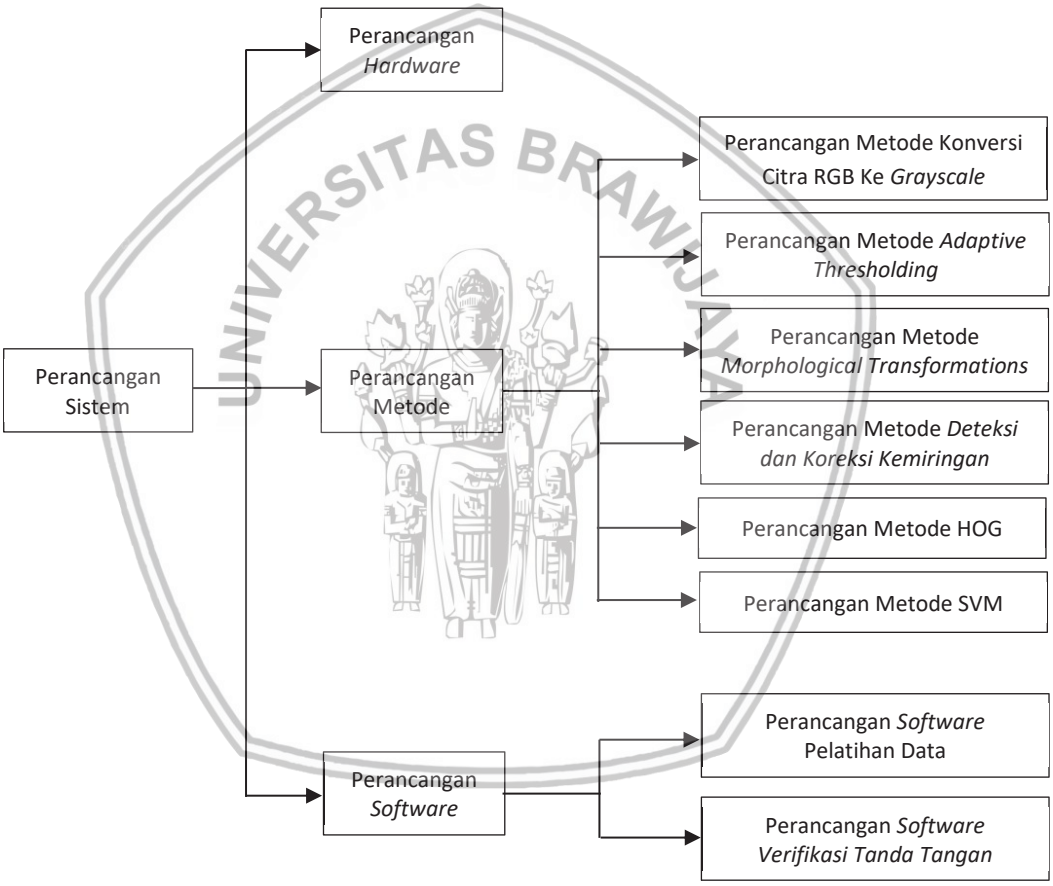
VNC merupakan *software* untuk melakukan akses *remote* pada perangkat desktop, *software* ini dapat digunakan untuk mengakses Raspberry Pi melalui laptop.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Pada bab perancangan dan implementasi sistem dibahas secara rinci tentang bagaimana sistem dirancang dari sisi *hardware*, metode, dan *software* serta bagaimana mengimplementasikannya.

5.1 Perancangan Sistem

Pada perancangan sistem terdapat tiga bagian pembahasan, yaitu dimulai dari perancangan *hardware*, perancangan metode dan perancangan *software* yang akan digunakan pada sistem, tahapan perancangan sistem dapat dilihat pada Gambar 5.1.

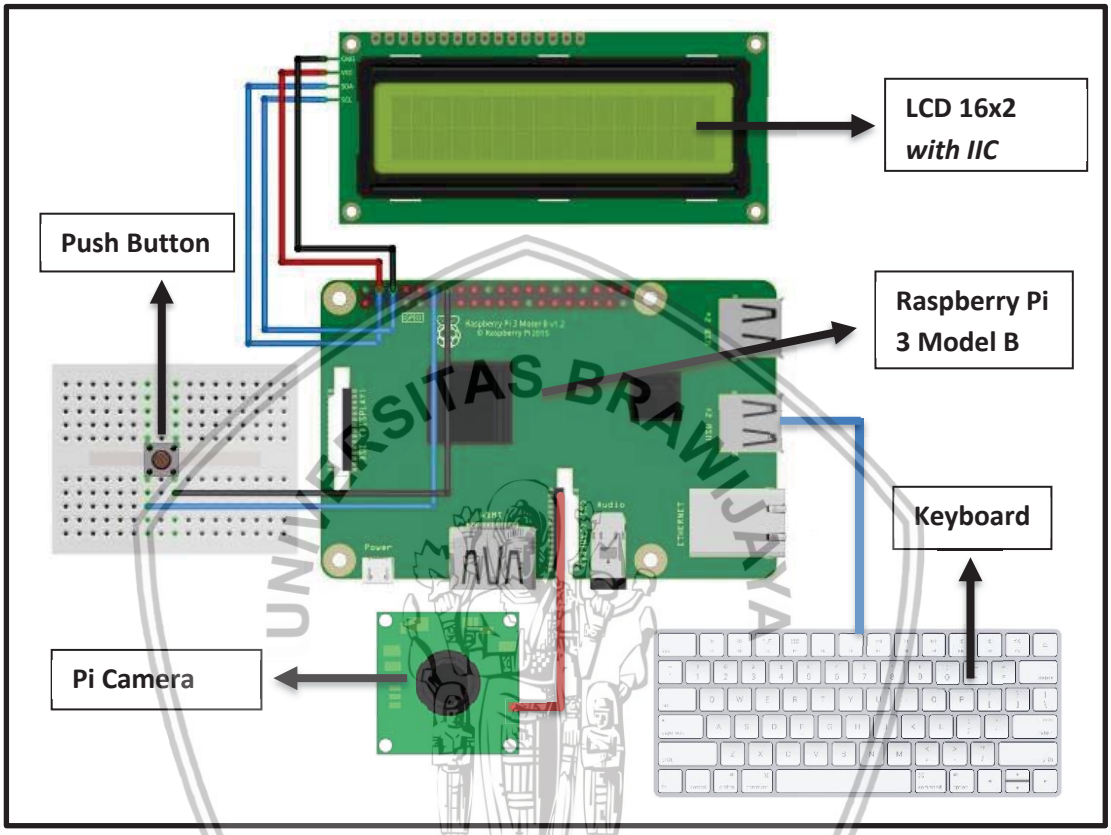


Gambar 5.1 Skema perancangan sistem

5.1.1 Perancangan *Hardware*

Penelitian ini menggunakan perangkat keras yang nantinya akan dibagi menjadi tiga bagian, yaitu bagian *input*, pemroses dan *output*. Perancangan bagian *input* meliputi penggunaan modul Pi Camera v1.3 sebagai *input* citra digital tanda tangan yang akan diproses dengan *push button* sebagai pemicu modul kamera menangkap citra dan *keyboard* sebagai *input* identitas nama untuk folder data latih mana yang digunakan. Bagian pemroses menggunakan Raspberry Pi 3 Model B sebagai unit pengolahan citra, pemilihan *mini computer* Raspberry Pi

dikarenakan pengolahan citra membutuhkan kemampuan komputasi yang besar dan tidak cukup jika menggunakan mikrokontroler. Bagian *output* menggunakan LCD dengan ukuran 16x2 sebagai tampilan keluaran hasil pengolahan citra, LCD yang dipakai memiliki IIC (*Inter Integrated Circuit*) *backpack* untuk menyederhanakan dan menghemat pemakaian pin. Rancangan *hardware* sistem dapat dilihat pada Gambar 5.2.



Gambar 5.2 Skema perancangan *hardware*

Keterangan koneksi pin untuk perancangan *hardware* pada penelitian ini dapat dilihat pada Tabel 5.1.

Tabel 5.1 Keterangan pin perancangan *hardware*

Pin dan Port Raspberry Pi	Push Button	Pi Camera	Keyboard	LCD
Pin 3 (SDA, I2C)				SDA
Pin 4 (VCC)				VCC
Pin 5 (SCL, I2C)				SCL
Pin 6 (Ground)				GND
Pin 12 (GPIO18)	+			
Pin 14 (Ground)	-			
CSI Port		✓		
USB Port			✓	

Dari Gambar 5.2 dan Tabel 5.1 dapat dilihat bahwa dengan memakai I2C, LCD hanya membutuhkan 4 pin yaitu pin SDA (*Serial Data*), SCL (*Serial Clock*), VCC, dan

ground. Untuk *push button* dihubungkan pada pin 12 (GPIO18) sebagai pin *input* dan *ground* pada pin 14. Pi Camera dihubungkan pada CSI (*Camera Serial Interface*) port dan *keyboard* dihubungkan pada USB port.

5.1.2 Perancangan Metode

Subbab ini menjabarkan tentang perancangan metode yang dipakai dalam penelitian ini. Sebelum ekstraksi ciri citra dapat dilakukan, diperlukan *preprocessing* citra terlebih dahulu, metode yang digunakan dalam *preprocessing* dalam penelitian ini meliputi metode konversi citra RGB ke *Grayscale*, metode *Adaptive Thresholding*, metode *Morphological Transformations* dan metode deteksi dan koreksi kemiringan citra. Setelah melalui *preprocessing* citra, dilakukan ekstraksi ciri citra menggunakan metode *Histogram of Oriented Gradients*, dan untuk klasifikasi menggunakan metode *Support Vector Machine*.

5.1.2.1 Metode Konversi Citra RGB ke *Grayscale*

Pada umumnya citra digital yang dihasilkan oleh kamera disimpan dalam format warna atau *color-space* RGB (*Red, Green, Blue*) yang memiliki 3 nilai dalam setiap pixelnya. Format warna *grayscale* hanya memiliki 1 nilai yang direpresentasikan dalam derajat keabuan. Pada format citra *grayscale* hitam merupakan representasi warna dengan intensitas terendah yang bernilai 0, dan putih merupakan representasi warna tertinggi yang bernilai 1. Perubahan format warna RGB ke *grayscale* dilakukan agar pencarian ROI (*Region of Interest)* dari citra mudah dilakukan. Untuk perhitungan konversi format warna RGB ke *grayscale* dapat dicapai dengan metode *Luma* (Y) pada persamaan di bawah.

$$Y = 0.299R + 0.587G + 0.144B \tag{5.1}$$

Persamaan 5.1 di atas didapat dari fungsi merubah *colorspace* RGB ke *grayscale* pada *library* OpenCV, di persamaan tersebut nilai derajat keabuan dari nilai *Luma* dapat diketahui dengan memasukkan nilai piksel *Red, Green,* dan *Blue* pada persamaan tersebut. Sebagai contoh citra digital mempunyai nilai piksel pada suatu koordinat bernilai Red=130, Green=60, Blue=90, maka hasil konversi nilai Luma adalah:

$$Y = (0.299 \times 130) + (0.587 \times 60) + (0.144 \times 90) = 87$$

Citra digital *grayscale* tersebut mengesampingkan ciri warna pada citra namun tetap menjaga ciri dari bentuk objek, hal ini dapat meringankan komputasi pada citra digital karena pada format warna *grayscale* sistem hanya menghitung satu nilai setiap pikselnya dibandingkan tiga nilai pada format RGB.

5.1.2.2 Metode *Adaptive Thresholding*

Untuk membedakan antara objek tanda tangan dan *background* pada citra digital diperlukan suatu metode untuk mendapatkan *Region of Interest* (ROI) atau objek yang diinginkan pada citra. *Thresholding* merupakan salah satu metode segmentasi untuk mendapatkan ROI pada suatu citra digital yang mudah diimplementasikan, metode ini mensegmentasi citra dengan mengatur nilai intensitas piksel yang diatas batas (*threshold*) sebagai *foreground* yang diberi nilai

1, dan nilai piksel di bawah *threshold* sebagai *background* yang diberi nilai 0. *Thresholding* memerlukan input berupa citra *grayscale* dan hasil akhirnya berupa citra *binary*. Formula *thresholding* dapat dilihat pada persamaan 5.2 di bawah.

$$dst(x,y) = \begin{cases} 1, & src(x,y) > T \\ 0, & else \end{cases} \quad (5.2)$$

Keterangan:

$dst(x,y)$ = Nilai destinasi piksel (x,y) pada citra.
$src(x,y)$ = Nilai piksel (x,y) pada citra *input*.
T = Nilai *threshold*.

Saat menggunakan metode *thresholding* untuk segmentasi citra, *Fixed Thresholding* tidak cocok jika *background* memiliki nilai piksel yang bervariasi (Chan, et al., 1998), maka dari itu digunakan metode *Adaptive Thresholoding*. *Adaptive Thresholoding* mempunyai nilai threshold yang berbeda pada setiap pixel citra, metode ini mengkalkulasi nilai *threshold* pada area-area kecil dalam citra, sehingga area yang berbeda pada citra yang sama memiliki nilai threshold yang juga berbeda. Metode ini memberikan hasil yang baik untuk citra dengan tingkat pencahayaan yang berbeda.

Pada *adaptive threshold* untuk mendapatkan nilai *threshold* suatu piksel dapat dicapai dengan mencari *mean* pada area kecil citra dari piksel ketetanggan yang telah didefinisiakan sebelumnya. Untuk rumus mencari nilai threshold dapat dilihat pada persamaan 5.3 berikut:

$$T = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \quad (5.3)$$

Keterangan:
T : Nilai threshold
n : Jumlah data

Sebagai contoh pada suatu koordinat piksel (x,y) dari citra *grayscale* bernilai 182 dan memiliki nilai ketetanggaan seperti pada Gambar 5.3, akan dihitung nilai *threshold* piksel tersebut dengan parameter blok ketetanggan 3×3 piksel.

157	153	174
155	182	163
180	180	50

Gambar 5.3 Contoh blok 3x3 piksel pada citra grayscale

Dengan menggunakan persamaan 5.3 nilai threshold pada piksel tengah yang bernilai 182 tersebut yaitu:

$$Threshold = \frac{157 + 153 + 174 + 155 + 182 + 163 + 180 + 180 + 50}{9}$$

$$= \frac{1394}{9}$$

$$= 154{,}89$$

Nilai threshold tersebut akan menjadi refrensi untuk memutuskan apakah piksel merupakan objek atau background, dengan memasukkan pada persamaan 5.2.

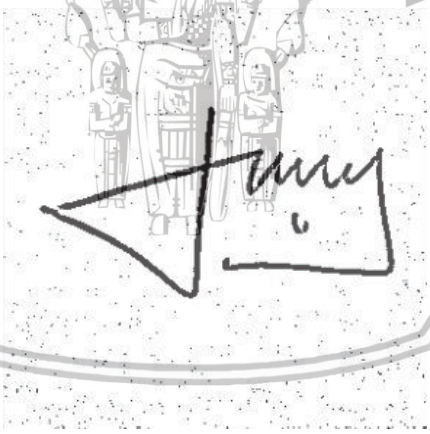
$$dst(x, y) = \begin{cases} 1, & 182 > 154{,}89 \\ 0, & else \end{cases}$$

$$= 1$$

Karena nilai piksel melebihi *threshold*, maka *output* pada piksel tujuan bernilai 1 atau dikategorikan sebagai objek.

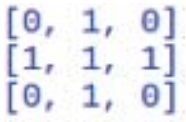
5.1.2.3 Metode *Morphological Transformations*

Dari citra hasil segmentasi *thresholding* untuk mendapatkan objek tanda tangan masih terdapat *noise*, *noise* tersebut berupa titik--titik kecil pada *background* citra, seperti ditunjukkan pada Gambar 5.4 berikut.



Gambar 5.4 Contoh noise pada citra tanda tangan

Salah satu metode untuk dapat mengurangi atau menghilangkan *noise* titik-titik hitam tersebut dapat menggunakan metode transformasi *closing morphology*. *Morphology* merupakan teknik komputasi untuk merubah struktur geometris atau bentuk dalam citra digital. Metode ini umumnya diaplikasikan pada gambar *binary* dan memerlukan *structuring element* untuk menentukan bagaimana operasi morfologi dilakukan. *Structure element* yang dipakai dalam penelitian ini berbentuk elips dengan ukuran panjang 3 dan lebar 3 seperti yang ditunjukkan pada Gambar 5.5.



Gambar 5.5 Structure element elips dengan panjang 3 dan lebar 3

Structure element ini akan diaplikasikan pada citra tanda tangan dengan operasi *closing*. Operasi *closing* pada citra biner $\boldsymbol{A}$ dengan sebuah struktur elemen $\boldsymbol{B}$ yaitu didapat dari operasi dilasi yang kemudian diikuti oleh erosi, seperti ditunjukkan pada persamaan 5.4 berikut.

$$A \circ \mathrm{B} = (A \oplus B) \ominus \mathrm{B} \tag{5.4}$$

Dimana dilasi dapat dicari dengan menerapkan operasi union atau gabungan seperti pada persamaan 5.5 berikut:

$$A \oplus B = \bigcup_{b \in B} A_b \tag{5.5}$$

Dan erosi dapat dicari dengan menerapkan operasi *intersection* atau irisan seperti pada persamaan 5.6 berikut:

$$\mathrm{A} \ominus \mathrm{B} = \bigcap_{a \epsilon B} A_{-b} \tag{5.6}$$

Sebagai contoh citra biner mempunyai nilai seperti terlihat pada Gambar 5.6 akan diterapkan morfologi closing seperti pada persamaan 5.4 dengan structure element dengan bentuk elips dari Gambar 5.5.

1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	0	0
1	1	1	0	0	0	0	1	1	1
1	1	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	1
1	0	0	1	1	1	1	0	0	1
0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	0	1	0	0	0

Gambar 5.6 Contoh citra biner

Operasi morfologi closing diawali dengan dengan melakukan dilasi terlebih dahulu, yaitu menerapkan operasi *union* dari *structure element* dengan citra biner Gambar 5.6. Hasil operasi dilasi terlihat pada Gambar 5.7, operasi ini membuat area hitam terlihat lebih mengecil.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	1	1	1
1	1	0	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0	0

Gambar 5.7 Citra biner hasil operasi dilasi

Hasil dari operasi dilasi pada Gambar 5.7 kemudian dilakukan erosi dengan menerapkan operasi *intersection* dengan *structure element.* Hasil akhir dari erosi tersebut merupakan *output* dari morfologi *closing* yang ditunjukkan pada Gambar 5.8. Dari hasil tersebut terlihat bahwa titik-titik hitam pada citra biner *input* menghilang.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	1	1	1
1	1	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	1
1	0	0	1	1	1	1	0	0	1
0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	1	0	0	0

Gambar 5.8 Hasil akhir morfologi closing

5.1.2.4 Metode Deteksi dan Koreksi Kemiringan

Dalam deteksi objek pada citra khususnya metode deteksi objek berdasarkan ciri geometrisnya, tingkat kemiringan objek tanda tangan pada citra dapat mempengaruhi akurasi dalam proses pengolahan citra. Kemiringan ini umumnya disebabkan dari gaya tulisan dari masing-masing individu atau kesalahan saat menangkap citra dari objek, oleh karena itu diperlukan metode untuk deteksi dan koreksi kemiringan objek pada citra.

Koreksi kemiringan pada citra biner dapat dicapai dengan bantuan *image moments* yang ada pada library OpenCV. *Image moment* biasanya digunakan untuk mengkarakteristik bentuk dari sebuah objek pada citra, image moment ini meng*capture* properti objek pada citra seperti *centeroid* (koordinat titik tengah dari objek), area, orientasi dan properti lain (Rosebrock, 2016). Pada library OpenCV *image momets* memiliki dua jenis, *spatial moments* (m_{ji}) dan *central moments* (mu_{ji}). Untuk *spatial moments* dari sebuah citra (x, y) dapat dihitung dengan formula sebagai berikut:

$$m_{ji} = \sum_{x,y} \left(array(x,y) \cdot x^j \cdot y^i \right) \tag{5.7}$$

Sedangkan untuk formula dari *central moments* sebagai berikut:

$$mu_{ji} = \sum_{x,y} \left(array(x,y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i \right) \tag{5.8}$$

Dimana $(\bar{x}, \bar{y})$ adalah *centeroid* yang dapat dihitung dengan:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}} \tag{5.9}$$

Dengan menggunakan properti dari *image moment* tersebut dapat dicari *skewness* atau derajat ketidaksimetrisan objek citra berdasarkan rasio dua *central moments* (Malik, 2017), yang dapat dicari dengan rumus sebagai berikut:

$$skewness = \frac{mu_{11}}{mu_{02}} \tag{5.10}$$

Nilai skewness tersebut digunakan dalam matrik untuk melakukan transformasi *affine* pada citra yang akan dikoreksi kemiringannya. Pada OpenCV affine transform direpresentasikan dengan menggunakan matrik 2×3 sebagai berikut:

$$M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix}$$

Dengan diberikan koordinat (x, y) pada matriks affine transform diatas, maka akan berpindah pada koordinat (x_t, y_t) menggunakan persamaan di bawah.

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.11}$$

Untuk mengkoreksi kemiringan objek tanda tangan pada citra, nilai *skewness* yang didapat dimasukkan pada matrik *affine transform* sebagai berikut:

$$M = \begin{bmatrix} 1 & skew & -0.5 \times size \times skew \\ 0 & 1 & 0 \end{bmatrix} \tag{5.12}$$

Keterangan:

- $skew$: Nilai derajat ketidaksimetrisan objek citra yang didapat pada persamaan 5.10.
- $size$: Ukuran dimensi lebar atau tinggi pada citra $(n \times n)$.

Matriks *affine transform* pada persamaan 5.12 diaplikasikan pada data input citra tanda tangan yang akan dikoreksi kemiringannya dengan menggunakan perhitungan seperti pada persamaan 5.11.

5.1.2.5 Metode Histogram of Oriented Gradients

Sebelum menghitung Histogram of Oriented Gradients pada citra tanda tangan, terlebih dahulu citra melewati tahap *prepocessing*. *Preprocessing* yang diperlukan untuk citra tanda tangan seperti merubah colorspace RGB ke *grayscale*, merubah ukuran resolusi citra, segmentasi, morfological filter, dan deteksi dan koreksi kemiringan. Langkah-langkah dalam menghitung fitur deskriptor HOG sebagai berikut:

1. Menghitung nilai gradient citra

Untuk menghitung deskriptor HOG, proses pertama yaitu menghitung horizontal dan vertikal gradien, proses ini dapat dicapai dengan menggunakan operator sobel. Operator dapat dihitung dengan mengkonvolusi citra $\boldsymbol{I}$ dengan kernel $\boldsymbol{G_x}$ untuk perubahan horizontal, dan $\boldsymbol{G_y}$ untuk perubahan vertikal, seperti berikut.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

Dan untuk G_y sebagai berikut:

$$G_y = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

Selanjutnya dengan menggunakan kernel operator sobel, magnitude gradien pada setiap titik citra dapat dihitung dengan menggunakan rumus:

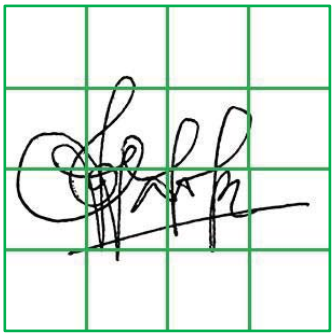
$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Untuk arah gradiennya dihitung dengan menggunakan:

$$\theta = arctan \frac{G_y}{G_x}$$

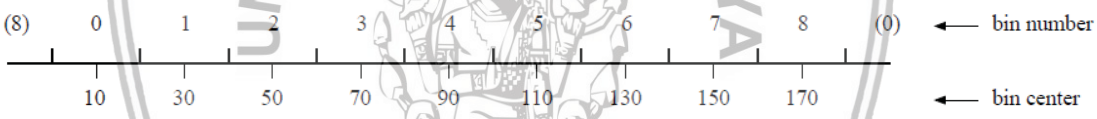
2. Menghitung *histogram of oriented gradients* pada setiap sel

Langkah selanjutnya yaitu membagi citra menjadi sel dengan ukuran $n \times n$ piksel, ukuran sel ini hendaknya tidak terlalu besar maupun kecil sehingga dapat merepresentasikan setiap ciri dari objek dalam citra. Pada penelitian ini, sistem menggunakan sel dengan ukuran 75x75 piksel pada citra tanda tangan yang berdimensi 300x300 piksel. Ilustrasi sel untuk perhitungan HOG dapat dilihat pada Gambar 5.9.



Gambar 5.9 4x4 sel pada citra tanda tangan

Kemudian masing-masing sel dalam citra dihitung nilai magnitude dan orientasi gradiennya, nilai gradient tersebut direpresentasikan dengan histogram. Dasar pembentukan ciri HOG pada masing-masing sel adalah proses akumulasi gradien dengan orientasi yang sama, orientasi gradien tersebut dikelompokkan menjadi beberapa bagian bin histogram (Harfiya, et al., 2017). Pada penelitian dengan judul "Selection of Histograms of Oriented Gradients Features for Pedestrian Detection" yang dilakukan Kobayashi, et al. (2008) jumlah bin 9 pada histogram dapat memberikan hasil yang optimal, oleh karena itu sistem ini juga menggunakan jumlah bin sebanyak 9 dengan rentang orientasi gradient dari 0° hingga 180°. Ilustrasi histogram dengan 9 bin dapat dilihat pada Gambar 5.10.



Gambar 5.10 Histogram dengan 9 bin dan *bin center*nya
Sumber: (Tomasi, 2012)

Dengan histogram pada Gambar 5.10, nilai magnitude gradien setiap piksel dalam sel akan didistribusikan berdasarkan nilai orientasi gradiennya. Untuk menghindari *error* pada distribusi, setiap piksel pada sel mendistribusikan magnitude gradiennya pada dua bin yang bin centernya berdekatan dengan nilai orientasi gradien piksel tersebut (Tomasi, 2012).

Secara teknis bin diberi nomor dari 0 hingga $B-1$ dan mempunyai *range* $w = \frac{180}{B}$. Bin i mempunyai batas $[wi, w(i+1)]$ dan titik tengah $c_i = w(i+\frac{1}{2})$. Sebuah piksel dengan magnitude gradien μ dan orientasi θ memberikan vote nilainya pada nomor bin ***j*** dengan rumus sebagai berikut:

$$j = \left\lfloor \frac{\theta}{w} - \frac{1}{2} \right\rfloor mod\ B \text{ dengan magnitude sebesar } v_j = \mu \frac{c_{j+1} - \theta}{w}$$

Dan juga memberikan vote pada:

$$(j+1) mod\ B \text{ dengan magnitude sebesar } v_{j+1} = \mu \frac{\theta - c_j}{w}$$

Sebagai contoh jika sebuah piksel mempunyai gradien dengan orientasi $\theta = 77^o$ dan magnitude 20 akan memberikan vote nilainya pada bin:

$$j = \left\lfloor \frac{77}{20} - \frac{1}{2} \right\rfloor mod\ 9$$

$$j = \left\lfloor \frac{67}{20} \right\rfloor mod\ 9$$

$$j = 3\ mod\ 9 = 3$$

Dengan magnitude sebesar:

$$v_j = 20\frac{c_{3+1} - 77}{20}$$

$$v_j = 20\frac{90 - 77}{20}$$

$$v_j = 20\frac{13}{20} = 13$$

Dan juga memberikan vote pada bin 4 sebesar:

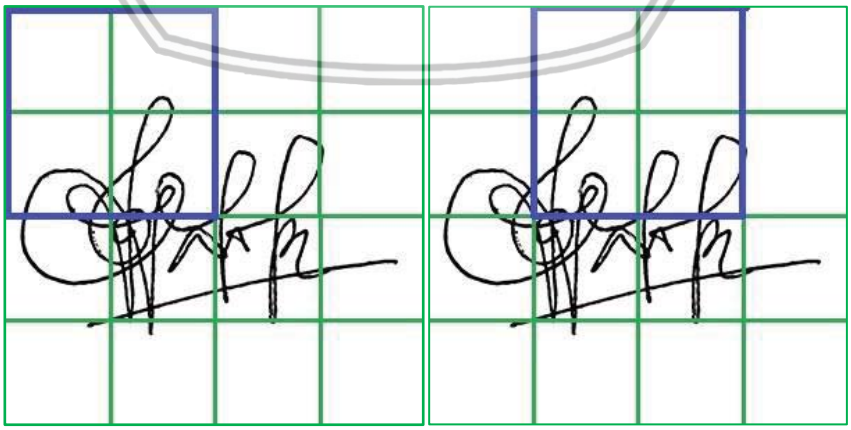
$$v_{j+1} = 20\frac{77 - 70}{20}$$

$$v_{j+1} = 20\frac{7}{20} = 7$$

Dengan perhitungan tersebut nilai magnitude gradien akan didistribusikan pada bin 3 sebesar 13 dan pada bin 4 sebesar 7, jumlah dari dua distribusi selalu μ.

3. Normalisasi Blok

Agar nilai HOG independen dari kondisi pencahayaan diperlukan normalisasi, normalisasi dilakukan dengan cara menggabungkan sel-sel kedalam sebuah blok terdiri dari 2×2 sel, jadi setiap blok berukuran $2C \times 2C$ piksel. Blok akan bergerak secara horizontal dan vertikal dengan jarak sebesar C piksel. Untuk ilustrasi dari blok dapat dilihat pada area kotak biru dalam Gambar 5.11.



(a) Blok pertama **(b) Blok kedua**

Gambar 5.11 Blok HOG pada citra tanda tangan

Dengan proses perhitungan blok yang tumpang tindih tersebut maka akan ada beberapa sel yang muncul pada blok yang berbeda. Perhitungan normalisasi dilakukan dengan menggabungkan histogram empat sel dalam blok menjadi suatu vector $\boldsymbol{v}$ dan dinormalisasikan dengan *Euclidean norm* untuk mendapatkan *block feature* $\boldsymbol{f}$ sebagai berikut:

$$f = \frac{v}{\sqrt{\|v\|^2 + \epsilon}} \tag{5.13}$$

Pada rumus tersebut ϵ adalah konstanta positif dengan nilai yang kecil untuk bertujuan mencegah pembagian dengan nol, jika nilai gradien pada citra adalah nol.

4. HOG *feature vector*

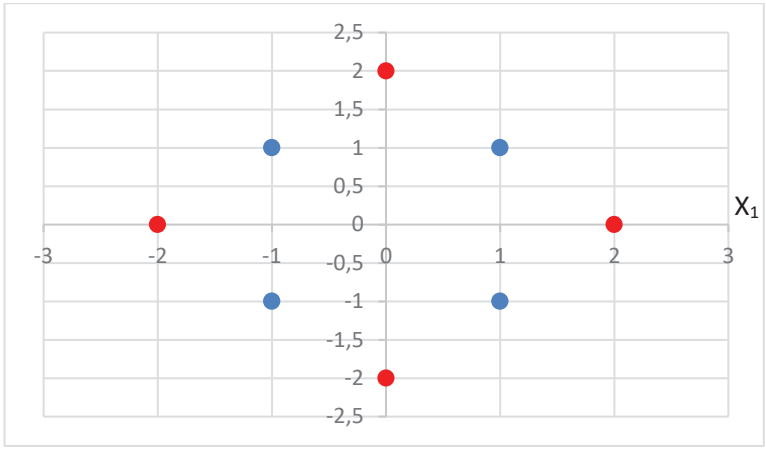
Untuk mendapatkan fitur vector final pada seluruh area citra, setiap *blok feature* yang telah dinormalisasi digabungkan menjadi satu vector besar, vector inilah yang akan menjadi ciri HOG untuk dimasukkan pada proses klasifikasi. Pada sistem ini menggunakan citra tanda tangan dengan ukuran dimensi 300x300 piksel yang dibagi menjadi beberapa sel dengan masing-masing sel berukuran 75x75 piksel, dan blok berukuran 150x150 piksel. Dengan parameter tersebut, akan didapat 9 posisi blok yang berbeda, yang masing-masing blok terdiri dari 4 histogram sel, dan setiap histogram terdapat 9 nilai. Jika digabungkan maka satu citra tanda tangan akan terdiri dari vector yang berukuran $9 \times 4 \times 9 = 324$ nilai.

5.1.2.6 Metode *Support Vector Machine*

Setelah didapatkan fitur HOG, fitur tersebut selanjutnya akan masuk dalam proses pelatihan data dengan klasifikasi, metode klasifikasi yang akan dipakai dalam penelitian ini yaitu Support Vector Machine (SVM). Konsep dasar SVM yaitu usaha mencari *hyperplane* atau garis pembatas yang paling optimal (terbaik) yang digunakan sebagai pembeda dua buah *class* pada input space. Fitur HOG dari citra tanda tangan yang didapat tidak memungkinkan jika diklasifikasikan menggunakan linier SVM, hal ini dikarenakan pada setiap citra tanda tangan dihasilkan 324 nilai fitur.

Salah satu metode untuk mengklasifikasikan data yang tidak dapat dipisahkan secara linier adalah dengan mentransformasikan data ke dalam dimensi ruang fitur (*feature space*). Data dipetakan menggunakan fungsi pemetaan (transformasi) $x_k \rightarrow \Phi(x_k)$ sehingga terdapat bidang pemisah yang dapat memisahkan data sesuai kelasnya (Sembiring, 2007). Sebagai contoh pada Gambar 5.12 terdapat data set yang datanya memiliki dua atribut dan dua kelas yaitu kelas positif $(+1)$ dan kelas negatif (-1). Data yang memiliki kelas positif (berwarna merah) adalah $\{(2,0), (0,2), (-2,0), (0,-2)\}$, dan data yang memiliki kelas negatif (berwarna biru) adalah $\{(1,1), (1,-1), (-1,1), (-1,-1)\}$.

X_2



Gambar 5.12 Contoh data yang tidak dapat dipisahkan secara linier

Kelas Positif: $\binom{2}{0}, \binom{0}{2}, \binom{-2}{0}, \binom{0}{-2}$

Kelas Negatif: $\binom{1}{1}, \binom{1}{-1}, \binom{-1}{1}, \binom{-1}{-1}$

Pada kasus ini diperlukan fungsi pemetaan non-linier (Φ) agar data dapat ditransformasikan ke *feature space* dimana *hyperplane* atau garis pemisah dapat ditemukan. Untuk fungsi pemetaan dapat dilihat pada persamaan 5.14 berikut:

$$\Phi\binom{x_1}{x_2} = \begin{cases} \binom{6 - x_1 + (x_1 - x_2)^2}{6 - x_2 + (x_1 - x_2)^2} & if \sqrt{x_1^2 + x_2^2} \geq 2 \\ \binom{x_1}{x_2} & otherwise \end{cases} \quad (5.14)$$

Selanjutnya mentransformasi setiap data di kelas positif dan negatif dengan fungsi pemetaan non-linier pada persamaan 5.14.

- Untuk kelas positif: $\binom{2}{0}, \binom{0}{2}, \binom{-2}{0}, \binom{0}{-2}$

$$\Phi\binom{x_1}{x_2} = \Phi\binom{2}{0} = \binom{6 - 2 + (2 - 0)^2}{6 - 0 + (2 - 0)^2} = \binom{8}{10}$$

$$\Phi\binom{x_1}{x_2} = \Phi\binom{0}{2} = \binom{6 - 0 + (0 - 2)^2}{6 - 2 + (0 - 2)^2} = \binom{10}{8}$$

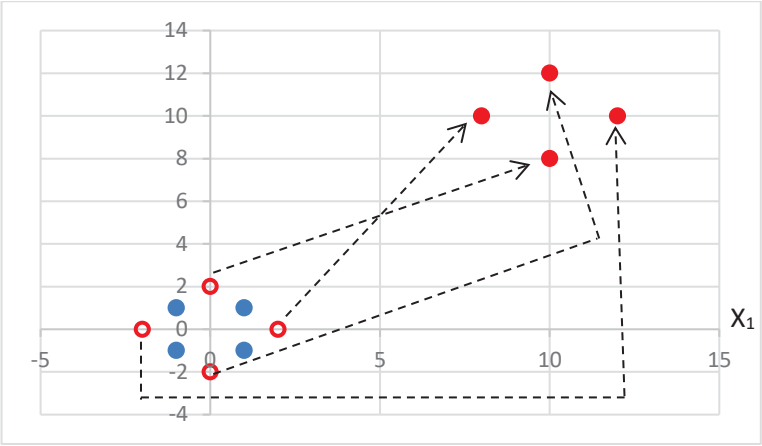
$$\Phi\binom{x_1}{x_2} = \Phi\binom{-2}{0} = \binom{6 + 2 + (-2 - 0)^2}{6 - 0 + (-2 - 0)^2} = \binom{12}{10}$$

$$\Phi\binom{x_1}{x_2} = \Phi\binom{0}{-2} = \binom{6 - 0 + (0 + 2)^2}{6 + 2 + (0 + 2)^2} = \binom{10}{12}$$

- Untuk kelas negatif: $\binom{1}{1}, \binom{1}{-1}, \binom{-1}{1}, \binom{-1}{-1}$ tidak ada perubahan karena $\sqrt{x_1^2 + x_2^2 < 2}$ untuk semua nilai vector.

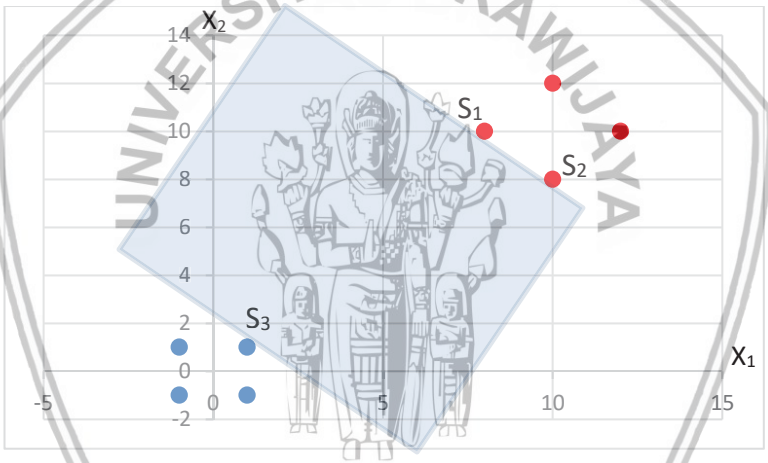
Hasil transformasi *feature space* dari data kelas positif dan negatif yang telah ditransformasikan dapat dilihat pada Gambar 5.13 berikut:

X2



Gambar 5.13 Hasil trasnformasi feature space

Dari gambar Gambar 5.13, data kelas positif dan negatif hasil transformasi sudah dapat dipisahkan secara linier. Selanjutnya dipilih 3 support vector untuk dapat menemukan *hyperplane* mengklasifikasikan dari dua kelas.



Gambar 5.14 Pemilihan Support Vector untuk penentuan Hyperplane

Dari Gambar 5.14 dipilih 3 *support vector* S_1, S_2, dan S_3, *support vector* ini dipilih berdasarkan jarak data terdekat dari antar kelas. Koordinat support vector yang didapat yaitu:

$$S_1 = \begin{pmatrix} 8 \\ 10 \end{pmatrix} \quad S_2 = \begin{pmatrix} 10 \\ 8 \end{pmatrix} \quad S_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Selanjutnya ketiga support vector tersebut ditambahkan nilai 1 (*bias vector*).

$$S_1 = \begin{pmatrix} 8 \\ 10 \end{pmatrix} \qquad \widetilde{S_1} = \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 10 \\ 8 \end{pmatrix} \qquad \widetilde{S_2} = \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix}$$

$$S_3 = \begin{pmatrix}1\\1\end{pmatrix} \qquad \widetilde{S_3} = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

Tahap selanjutnya perlu menemukan nilai dari parameter α_1, α_2 dan α_3 menggunakan persamaan berikut:

$$\alpha_1\widetilde{S_1}.\widetilde{S_1} + \alpha_2\widetilde{S_2}.\widetilde{S_1} + \alpha_3\widetilde{S_3}.\widetilde{S_1} = +1\ (Kelas\ Positif)$$
$$\alpha_1\widetilde{S_1}.\widetilde{S_2} + \alpha_2\widetilde{S_2}.\widetilde{S_2} + \alpha_3\widetilde{S_3}.\widetilde{S_2} = +1\ (Kelas\ Positif)$$
$$\alpha_1\widetilde{S_1}.\widetilde{S_3} + \alpha_2\widetilde{S_2}.\widetilde{S_3} + \alpha_3\widetilde{S_3}.\widetilde{S_3} = -1\ (Kelas\ Negatif)$$

Kemudian mensubstitusikan nilai $\widetilde{S_1}$, $\widetilde{S_2}$, dan $\widetilde{S_3}$ pada persamaan diatas.

$$\widetilde{S_1} = \begin{pmatrix}8\\10\\1\end{pmatrix} \quad \widetilde{S_2} = \begin{pmatrix}10\\8\\1\end{pmatrix} \quad \widetilde{S_3} = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

$$\alpha_1\begin{pmatrix}8\\10\\1\end{pmatrix}.\begin{pmatrix}8\\10\\1\end{pmatrix} + \alpha_2\begin{pmatrix}10\\8\\1\end{pmatrix}.\begin{pmatrix}8\\10\\1\end{pmatrix} + \alpha_3\begin{pmatrix}1\\1\\1\end{pmatrix}.\begin{pmatrix}8\\10\\1\end{pmatrix} = +1$$

$$\alpha_1\begin{pmatrix}8\\10\\1\end{pmatrix}.\begin{pmatrix}10\\8\\1\end{pmatrix} + \alpha_2\begin{pmatrix}10\\8\\1\end{pmatrix}.\begin{pmatrix}10\\8\\1\end{pmatrix} + \alpha_3\begin{pmatrix}1\\1\\1\end{pmatrix}.\begin{pmatrix}10\\8\\1\end{pmatrix} = +1$$

$$\alpha_1\begin{pmatrix}8\\10\\1\end{pmatrix}.\begin{pmatrix}1\\1\\1\end{pmatrix} + \alpha_2\begin{pmatrix}10\\8\\1\end{pmatrix}.\begin{pmatrix}1\\1\\1\end{pmatrix} + \alpha_3\begin{pmatrix}1\\1\\1\end{pmatrix}.\begin{pmatrix}1\\1\\1\end{pmatrix} = -1$$

Dari perkalian diatas didapatkan hasil sebagai berikut:

$$165\alpha_1 + 161\alpha_2 + 19\alpha_3 = +1$$
$$161\alpha_1 + 165\alpha_2 + 19\alpha_3 = +1$$
$$19\alpha_1 + 19\alpha_2 + 3\alpha_3 = -1$$

Dengan mengguanakan proses eliminasi atau substitusi didapatkan nilai α_1, α_2, dan α_3 sebagai berikut:

$$\alpha_1 = 0{,}0859 \qquad \alpha_2 = 0{,}0859 \qquad \alpha_3 = -1{,}4219$$

Nilai α_1, α_2, dan α_3 digunakan untuk mencari hyperplane yang memisahkan kelas positif dan negatif dengan persamaan berikut:

$$\widetilde{w} = \sum_i \alpha_i \tilde{S}_i \qquad (5.15)$$

Dengan mensubstitusikan nilai α dan nilai $\tilde{S}$ akan didapat:

$$\widetilde{w} = \alpha_1\begin{pmatrix}8\\10\\1\end{pmatrix} + \alpha_2\begin{pmatrix}10\\8\\1\end{pmatrix} + \alpha_3\begin{pmatrix}1\\1\\1\end{pmatrix}$$

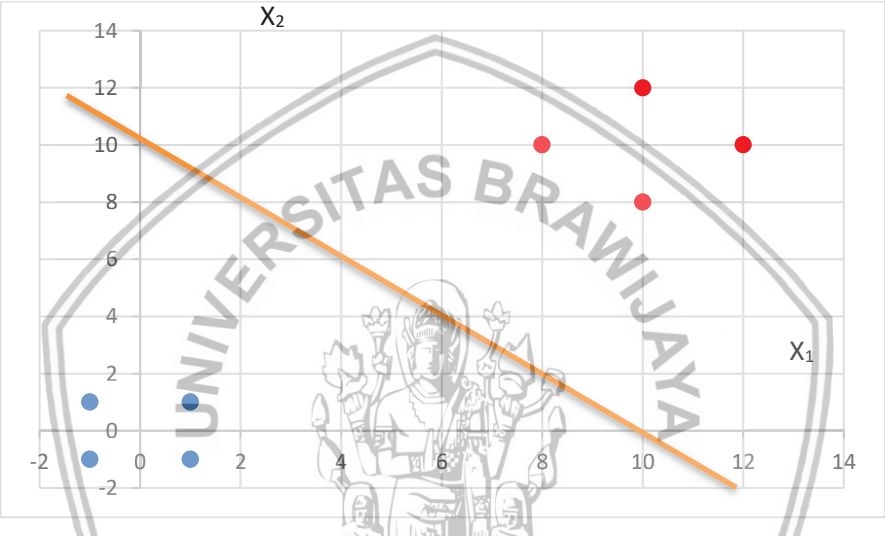
$$\widetilde{w} = 0{,}0859.\begin{pmatrix}8\\10\\1\end{pmatrix} + 0{,}0859.\begin{pmatrix}10\\8\\1\end{pmatrix} + (-1{,}4219).\begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}0{,}1243\\0{,}1243\\-1{,}2501\end{pmatrix}$$

Dari vektor $\tilde{S}$ yang ditambahkan nilai bias ($\boldsymbol{b}$) didapatkan nilai $\tilde{w}$ yang dapat digunakan sebagai hyperplane dengan *offset* $\boldsymbol{b}$, nilai tersebut dimasukkan kedalam fungsi pemisah hyperplane berikut:

$y = -(wx + b)$ dengan nilai $\boldsymbol{w}$ yang dinormalisasi $w = \begin{pmatrix} 0{,}1243/0{,}1243 \\ 0{,}1243/0{,}1243 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ sebagai gradien dan *offset* $b = \frac{-1{,}2501}{0{,}1243} = -10{,}057$ sebagai bias.

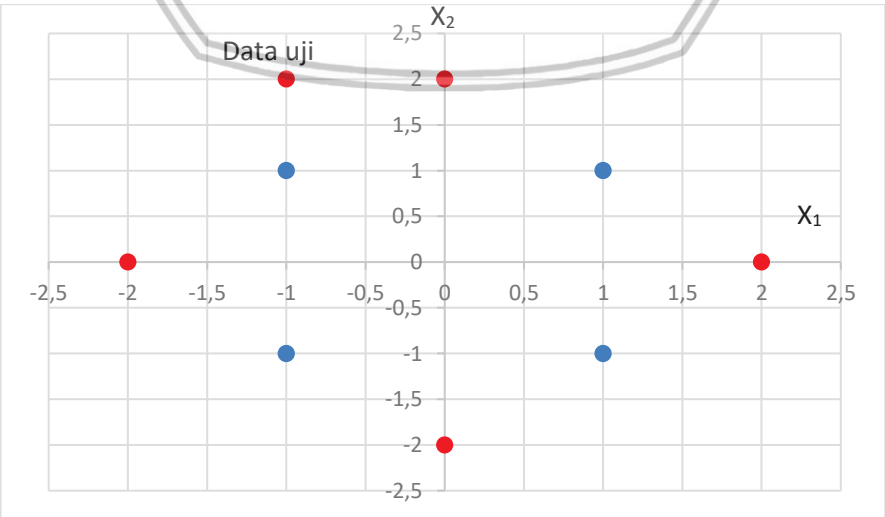
Maka didapat: $y = \frac{-1}{1}x + 10{,}057$

Dengan nilai fungsi tersebut didapatkan garis hyperplane yang memisahkan antara dua kelas seperti yang ditunjukkan pada Gambar 5.15.



Gambar 5.15 Hyperplane yang memisahkan dua kelas

Jika hyperplane tersebut diberikan data uji dengan nilai $(x_1, x_2) = (-1,2)$ seperti pada gambar Gambar 5.16, maka dapat diklasifikasikan dengan menggunakan persamaan 5.6.



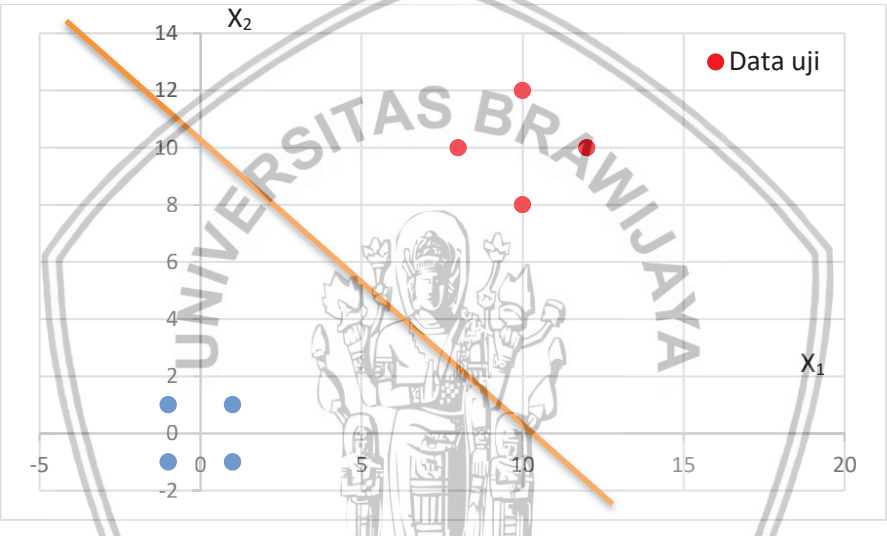
Gambar 5.16 Contoh klasifikasi pada data uji

Untuk $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ diperoleh:

$$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \Phi\begin{pmatrix} -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 6+1+(-1-2)^2 \\ 6-2+(-1-2)^2 \end{pmatrix} = \begin{pmatrix} 16 \\ 13 \end{pmatrix}$$

$$w.\Phi(x) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 16 \\ 13 \end{pmatrix} = 29 > 10{,}057$$

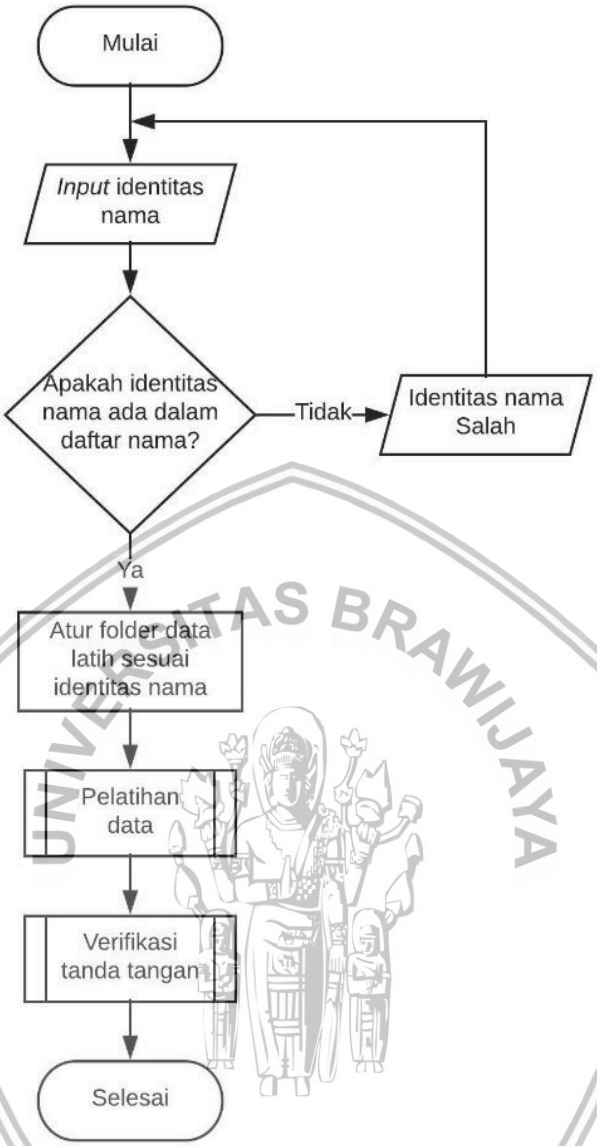
Dapat dilihat pada Gambar 5.17 data uji tersebut jika ditransformasikan dalam *feature space* mempunyai koordinat baru menjadi $(16{,}13)$. Titik koordinat tersebut jika dikalikan dengan $\boldsymbol{w}$ mempunai nilai 29 yang mana lebih dari nilai $b = 10{,}057$ sehingga diklasifikasikan menjadi kelas positif atau kelas yang berwarna merah.



Gambar 5.17 Hasil transformasi dan klasifikasi pada data uji

5.1.3 Perancangan *Software*

Pada subbab ini menjelaskan perancangan *software* yang merupakan alur kerja sistem pada sisi *software*, alur kerja sistem tersebut digambarkan dalam bentuk *flowchart.* Terdapat dua bagian utama program pada sistem ini, yaitu *software* untuk pelatihan data dan *software* untuk verifikasi tanda tangan. Program pelatihan data digunakan untuk melatih klasifikasi SVM berdasarkan ciri HOG pada data latih citra tanda tangan sedangkan program verifikasi tanda tangan digunakan untuk memberikan *output* prediksi pada citra tanda tangan yang akan diverifikasi. *Flowchart* alur kerja dari sistem dapat dilihat pada Gambar 5.18.



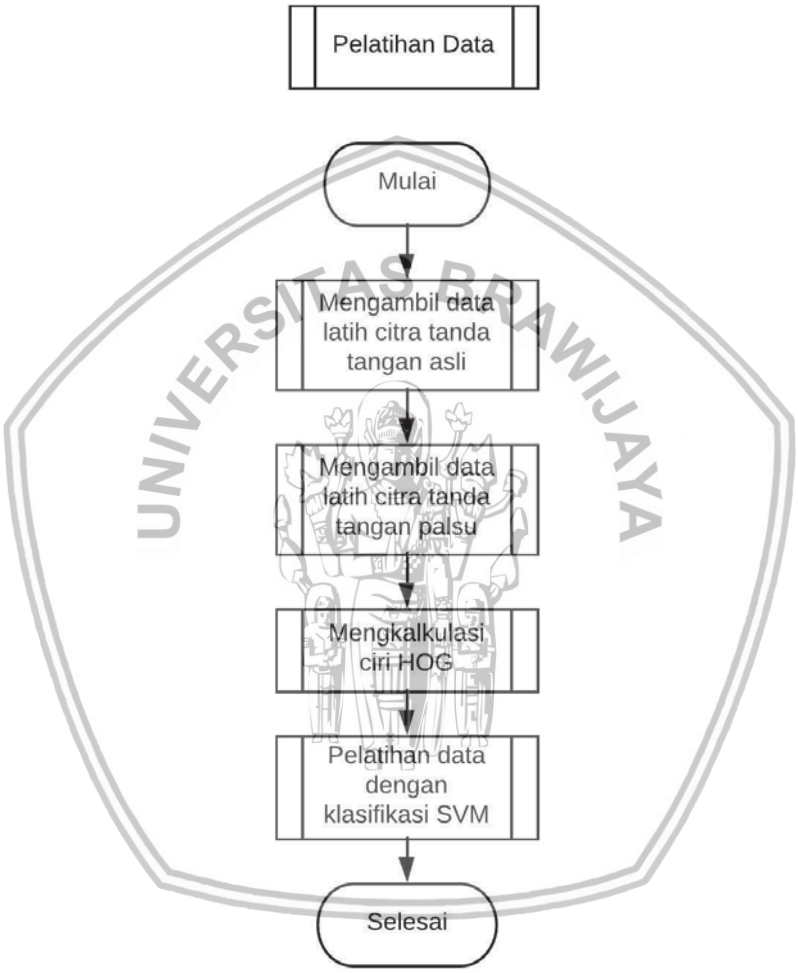
Gambar 5.18 *Flowchart* perancangan *software* sistem

Berdasarkan diagram alir pada Gambar 5.18, saat sistem dimulai sistem akan meminta *input* identitas nama penandatangan, identitas nama ini digunakan untuk menentukan citra tanda tangan siapa yang nantinya akan diverifikasi. Jika identitas nama yang dimasukkan salah, sistem akan meminta identitas nama lagi hingga benar. Setelah didapat identitas nama yang benar, sistem akan mengatur folder data latih sesuai dengan identitas nama yang diberikan dan setelah itu masuk ke proses pelatihan data dan proses verifikasi tanda tangan.

5.1.3.1 Perancangan *Software* Pelatihan Data

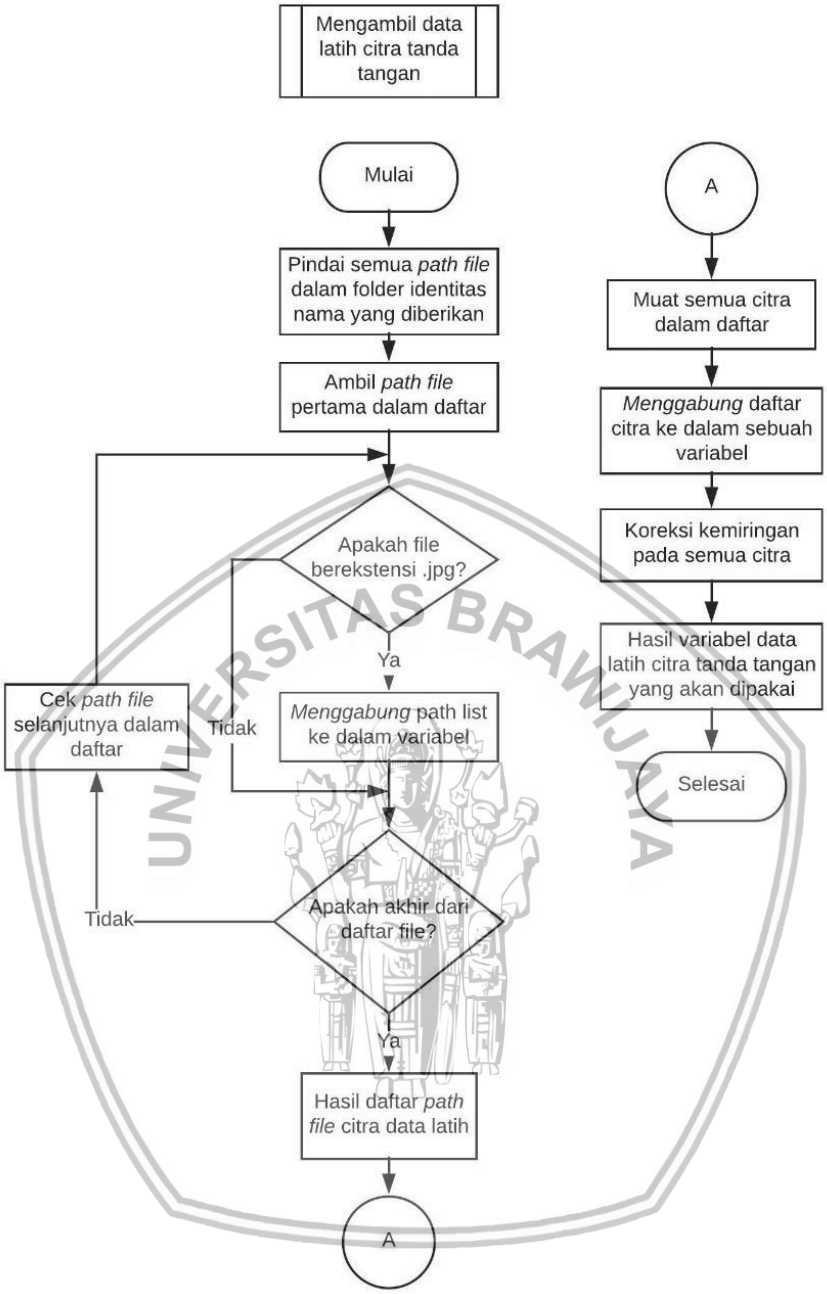
Perancangan *software* pelatihan data berfungsi untuk melatih program klasifikasi SVM berdasarkan ciri HOG dari citra tanda tangan. Untuk data latih digunakan 30 orang penanda tangan dengan masing-masing penanda tangan terdapat 10 citra tanda tangan asli dan 10 citra tanda tangan palsu dengan total

citra data latih sebanyak 600 citra, jenis ekstensi file citra yaitu .jpg dengan ukuran citra 300x300 piksel. Data latih citra tanda tangan disimpan didalam Raspberry Pi dan dibedakan berdasarkan nama penanda tangan, nama penandatangan digunakan sebagai nama folder sekaligus sebagai identitas nama untuk mengidentifikasi data latih mana yang nantinya akan digunakan. Citra tanda tangan asli dan palsu juga dibedakan dalam folder "asli" dan folder "palsu" untuk mempermudah sistem melakukan pelatihan data. Secara garis besar alur kerja dari *software* pelatihan data dapat dilihat pada *flowchart* di Gambar 5.19.



Gambar 5.19 *Flowchart* perancangan *software pelatihan data*

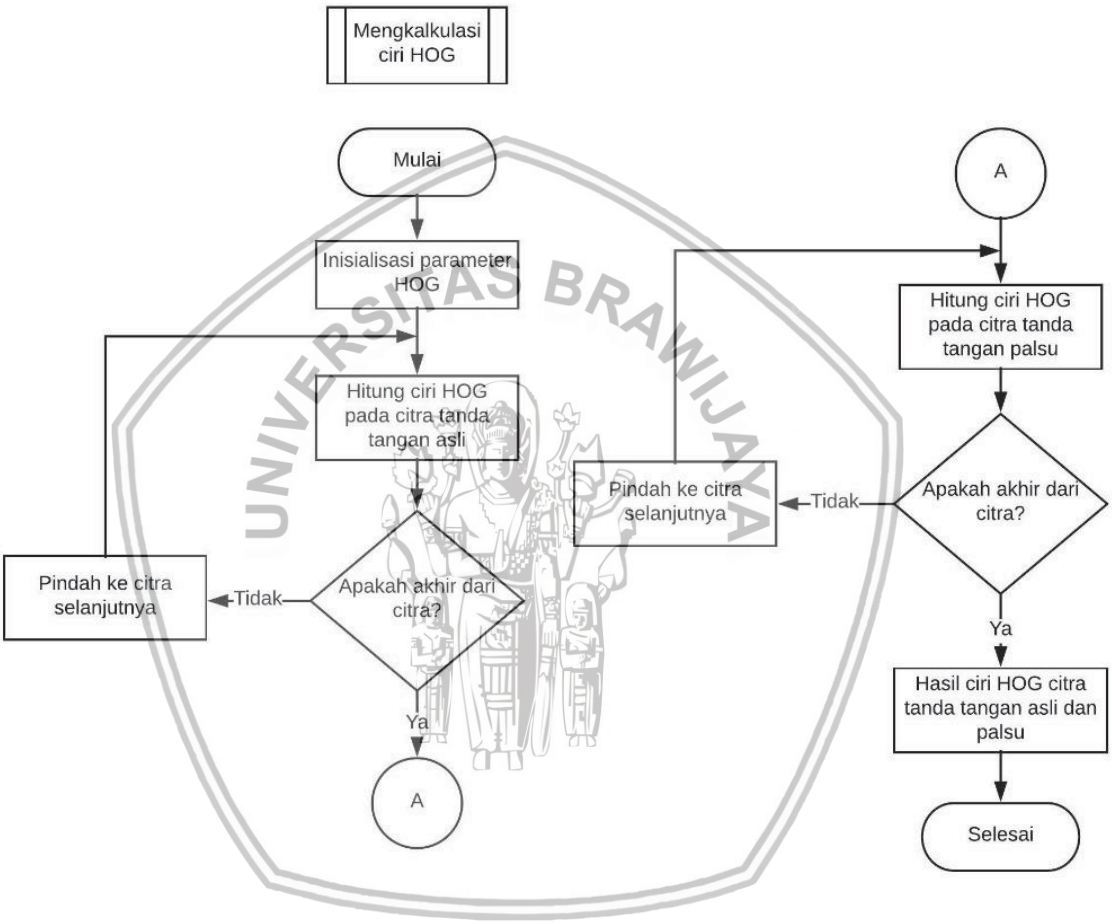
Pada Gambar 5.19 dalam proses pelatihan data, sistem akan mengambil data latih citra tanda tangan asli dan palsu berdasarkan identitas nama yang telah dimasukkan, kemudian citra tersebut dikalkulasi ciri HOGnya dan diklasifikasikan dengan SVM. Untuk Penjelasan lebih rinci untuk proses pengambilan data latih dapat dilihat pada Gambar 5.20.



Gambar 5.20 *Flowchart* proses mengambil data latih citra tanda tangan

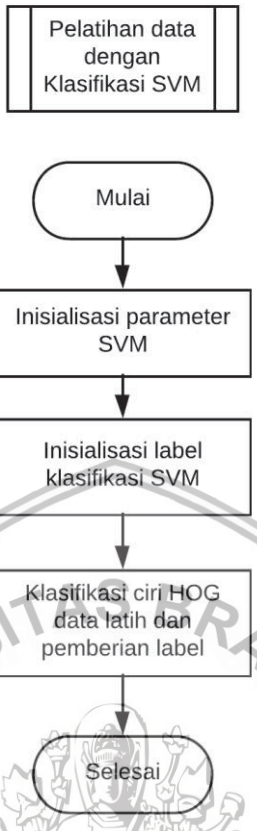
Pada Gambar 5.20 ada beberapa tahapan dalam proses pengambilan data latih citra tanda tangan, proses ini sama untuk pengambilan data latih citra tanda tangan asli maupun citra tanda tangan palsu. Dalam mengambil data latih, pertama sistem akan memindai seluruh *path file* yang ada pada folder identitas nama yang dimasukkan, kemudian file yang ada dalam daftar *path file* tersebut akan dilakukan pengecekan ekstensi filenya satu persatu apakah berformat .jpg, jika iya *path file* tersebut akan disimpan dalam sebuah variabel dan jika tidak sistem akan mengecek path file selanjutnya hingga akhir dari daftar *path file*. Pengecekan file ini bertujuan untuk menghindari *error* saat memuat data dengan

ekstensi file yang berbeda. Setelah dihasilkan daftar *path file* yang valid sistem akan memuat semua citra yang ada dalam daftar *path file* tersebut dan disimpan pada sebuah variabel *array* tiga dimensi. Daftar citra tersebut selanjutnya dikoreksi tingkat kemiringannya dengan fungsi *deskewing*, hal ini dilakukan karena tingkat kemiringan garis pada citra berpengaruh terhadap ciri HOG. Setelah dilakukan koreksi kemiringan, hasil data latih tersebut selanjutnya akan masuk ke proses kalkulasi ciri HOG, untuk detail proses kalkulasi ciri HOG dapat dilihat di flowchart pada Gambar 5.21 di bawah.



Gambar 5.21 *Flowchart* proses mengkalkulasi ciri HOG

Pada Gambar 5.21 proses kalkulasi ciri HOG pada data latih diawali dengan inisialisasi parameter-parameter yang dibutuhkan untuk kalkulasi HOG. Kemudian dilakukan perhitungan ciri HOG pada citra tanda tangan asli hingga selesai, begitu pula pada citra tanda tangan palsu. Setelah proses kalkulasi ciri HOG selesai didapat hasil ciri HOG citra data latih untuk tanda tangan asli dan palsu, hasil ini kemudian digunakan untuk proses pelatihan pada klasifikasi SVM. Untuk detail proses pelatihan klasifikasi SVM dapat dilihat pada Gambar 5.22.

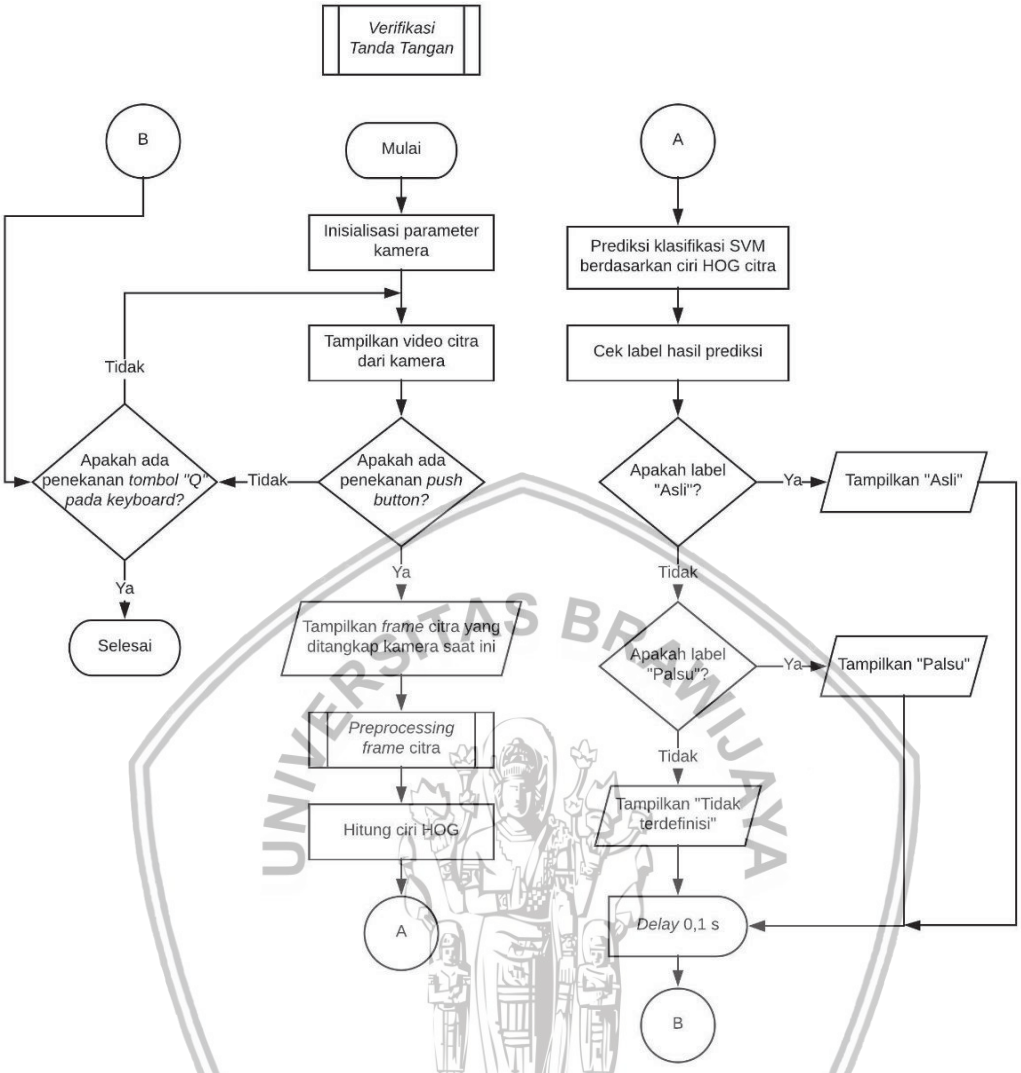


Gambar 5.22 *Flowchart* proses pelatihan data dengan klasifikasi SVM

Pada Gambar 5.22 proses pelatihan klasifikasi data latih dimulai dari insialisasi parameter yang diperlukan untuk klasifikasi SVM, dan juga diperlukan insialisasi label untuk tanda tangan asli dan dan tanda tangan palsu. Setelah dilakukan insialisasi parameter dan label selanjutnya ciri HOG dari tanda tangan asli dan palsu diklasifikasikan dengan SVM, pemberian label asli dan palsu juga dilakukan pada proses ini. Sampai tahap ini proses dari pelatihan data telah selesai dan selanjutnya akan masuk ke proses verifikasi tanda tangan.

5.1.3.2 Perancangan *Software* Verifikasi Tanda Tangan

Perancangan *software* verifikasi tanda tangan berfungsi untuk memberikan *output* hasil verifikasi dari citra tanda tangan yang diuji, hasil verifikasi tersebut didapat dari proses prediksi klasifikasi SVM. Verifikasi tanda tangan dilakukan dengan menangkap citra tanda tangan dengan kamera, pemicu kamera untuk menangkap citra tanda tangan yaitu dengan penekanan *push button*, dan *output* verifikasi apakah tanda tangan tersebut asli atau palsu ditampilkan pada LCD. Untuk detail alur kerja dari software verifikasi tanda tangan digambarkan dengan *flowchart* pada Gambar 5.23.



Gambar 5.23 *Flowchart* perancangan *software* verifikasi tanda tangan

Pada Gambar 5.23 proses verifikasi tanda tangan diawali dengan inisialisasi parameter yang diperlukan kamera untuk menangkap citra. Setelah itu sistem akan menampilkan video atau *stream frame* yang ditangkap kamera. Saat ada penekanan *push button*, *frame* citra tanda tangan yang ditangkap kamera saat itu akan ditampilkan dan diproses untuk dilakukan verifikasi. Sebelum dilakukan perhitungan ciri HOG, citra tersebut akan melewati tahap *preprocessing* terlebih dahulu, untuk proses detail *preprocessing* dapat dilihat pada Gambar 5.24. Setelah dilakukan *preprocessing*, citra akan dihitung ciri HOGnya, ciri tersebut digunakan untuk memprediksi kelas klasifikasi. Kemudian sistem akan mengecek label kelas hasil prediksi, apakah termasuk kelas tanda tangan asli atau tanda tangan palsu. Jika label yang dicek termasuk kelas asli, maka *output* pada LCD akan menampilkan "asli" dan sebaliknya jika label yang dicek termasuk kelas palsu, maka *output* pada LCD akan menampilkan "palsu". Setelah output ditampilkan, dilakukan *delay* program selama 0.1 detik, hal ini bertujuan untuk menghindari penekanan ganda yang tidak disengaja pada *push button*. Program ini akan terus berjalan hingga ada penekanan tombol 'Q' pada *keyboard*.

Gambar 5.24 *Flowchart* proses *preprocessing* citra

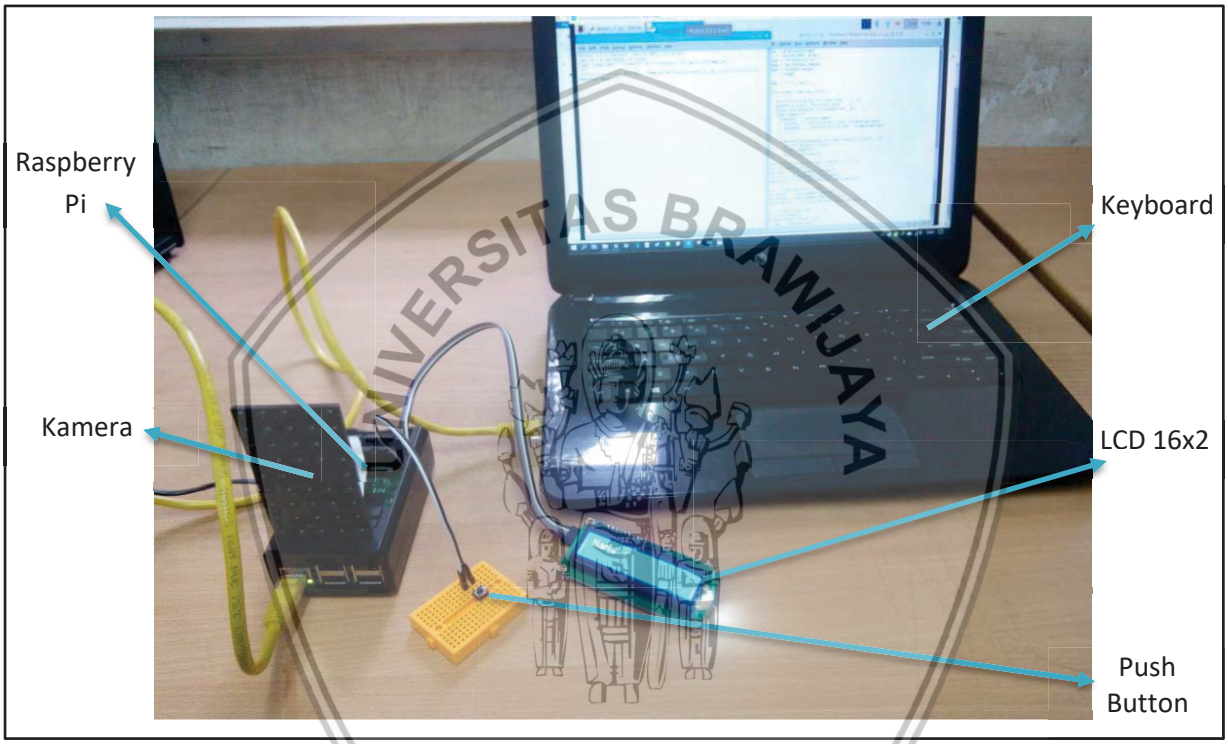
Pada Gambar 5.24 ada beberapa tahap dalam *preprocessing* citra, pertama yaitu merubah *colorspace* citra yang ditangkap kamera, yaitu dari *colorspace* RGB menjadi *grayscale*. Kemudian dimensi dari citra disamakan dengan dimensi citra data latih dengan merubahnya menjadi 300x300 piksel. Untuk mendapatkan *Region of Interest (ROI)* objek tanda tangan dan memisahkan dari background dilakukan segmentasi citra menggunakan adaptive threshold. Noise yang masih ada dari hasil segmentasi dikurangi atau dihilangkan dengan metode *closing morphology*. Terakhir dilakukan *deskewing* atau koreksi kemiringan pada citra tanda tangan dan proses *preprocessing* citra selesai.

5.2 Implementasi Sistem

Pada subbab ini membahas implementasi sistem berdasarkan perancangan yang telah dibuat pada subbab sebelumnya. Implementasi sistem meliputi implementasi hardware, implementasi metode, dan implementasi *software*.

5.2.1 Implementasi *Hardware*

Implementasi *hardware* sistem dibuat berdasarkan pada subbab perancangan. Sistem ini memiliki tiga perangkat input yaitu kamera, *push button,* dan *keyboard*. *Push button* digunakan sebagai pemicu kamera untuk mengambil citra tanda tangan dan *keyboard* digunakan untuk memberi masukkan identitas nama yang dipakai untuk pelatihan data. Rasberry pi sebagai unit pemroses masukan data citra digital dan hasil keluarannya dikirimkan ke LCD untuk ditampilkan. Tampilan implementasi *hardware* dari sistem secara keseluruhan dapat dilihat pada Gambar 5.25 berikut ini.



Gambar 5.25 Implementasi Hardware

Pada Gambar 5.25, *push button* dan LCD dihubungkan dengan GPIO (*General Purpose Input Output*) sedangkan Pi Camera dihubungkan dengan CSI port pada Raspberry Pi, sesuai dengan subbab perancangan. Penambahan *case* pada sistem direkomendasikan agar Raspberry Pi terlindungi dan sekaligus sebagai tempat peletakan kamera.

5.2.2 Implementasi Metode

Subbab ini membahas implementasi dari perancangan metode yang dipakai dalam penelitian ini, implementasi berupa kode program berbahasa Python yang dijalankan pada Raspberry Pi. Untuk dapat menjalankan program dari metode-metode yang dipakai, perlu adanya OpenCV sebagai penyedia *library* pada Raspberry Pi.

5.2.2.1 Metode Konversi Citra RGB ke *Grayscale*

Pada Tabel 5.2 program metode konversi citra *RGB ke Grayscale* diimplementasikan dalam bentuk fungsi Python, saat diperlukan nantinya fungsi ini dapat dipanggil dengan menggunakan *syntax* `grayscale(img)`. Fungsi ini merubah *colorspace* citra RGB ke *grayscale*.

Tabel 5.2 Kode program konversi citra RGB ke *Grayscale*

No	Kode Program
1	`def grayscale(img):`
2	`  gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
3	`  return gray_image`

Pada Tabel 5.2, pendefinisian nama fungsi untuk metode konversi citra RGB ke *grayscale* ditulis pada baris 1, fungsi ini memiliki satu parameter `(img)` sebagai input citra digital yang akan di konversi. Pada baris ke 2, untuk merubah *colorspace* citra digunakan fungsi `cv2.cvtColor()`, fungsi ini memiliki dua paramater, parameter pertama `img` sebagai *input* citra, dan parameter kedua `cv2.COLOR_BGR2GRAY` sebagai fungsi dari OpenCV untuk merubah colorspace citra RGB ke *grayscale*. Citra *grayscale* disimpan dalam variabel `gray_image` dan nilai variabel ini dikembalikan ke program utama pada baris ke 3.

5.2.2.2 Metode *Adaptive Thresholding*

Implementasi metode *Adaptive Thresholding* ke dalam kode program juga ditulis dalam bentuk fungsi Python dengan nama `threshold(img)`. Fungsi ini melakukan segmentasi *Adaptive Thresholding* pada *input* citra *grayscale*.

Tabel 5.3 Kode program *Adaptive Thresholding*

No	Kode Program
1	`def threshold(img):`
2	`  thres=cv2.adaptiveThreshold(img,255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)`
3	`  return thres`

Pada Tabel 5.3, definisi nama fungsi dari metode ditulis pada baris 1, fungsi ini memiliki satu parameter `(img)` sebagai parameter input citra *grayscale*. Baris ke 2 mendeklarasikan variabel `thres` untuk menyimpan hasil *thresholding* dari fungsi `cv2.adaptiveThrehold()`. Fungsi `cv2.adaptiveThreshold()` memiliki enam parameter yaitu:

- `img` sebagai parameter input citra digital.
- `255` sebagai parameter nilai yang akan diberikan pada piksel yang nilainya melebihi *threshold*.
- `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` sebagai parameter metode threshold yang dipakai.
- `cv2.THRESH_BINARY` sebagai parameter tipe thresholding yang dipakai.
- `11` sebagai parameter ukuran ketetanggaan piksel yang digunakan untuk menghitung nilai threshold.

- 2 sebagai parameter nilai konstanta pengurang dari *mean* atau *weighted mean*.

Baris ke 3 digunakan untuk mengembalikan nilai variabel `thres` ke program utama.

5.2.2.3 Metode *Morphological Transformations*

Implementasi dari metode *Morphological Transformations* juga ditulis dalam bentuk fungsi Python dengan nama `morfologi(img)`. Fungsi ini menerapkan operasi morfologi closing pada citra yang telah disegmentasi.

Tabel 5.4 Kode program *Morphological Transformations*

No	Kode Program
1	`def morfologi(img):`
2	`    closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3)))`
3	`    return closing`

Pada Tabel 5.4 baris 1 mendefinisikan nama fungsi dari metode, fungsi ini ini memiliki satu parameter `(img)` untuk input citra yang telah disegmentasi. Baris 2 mendeklarasikan variabel `closing` untuk menyimpan hasil morfologi. Morfologi dilakukan dengan menggunakan fungsi `cv2.morphologyEx()`, fungsi ini memiliki tiga parameter yaitu:

- `img` sebagai parameter input citra digital.
- `cv2.MORPH_CLOSE` sebagai parameter tipe operasi morfologi yang digunakan adalah operasi closing.
- `cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))` sebagai parameter bentuk struktur elemen yang dipakai adalah elips dengan ukuran 3x3 untuk operasi morfologi.

Baris ke 3 digunakan untuk mengembalikan nilai variabel `closing` ke program utama.

5.2.2.4 Metode Deteksi dan Koreksi Kemiringan

Metode *deskewing* diimplementasikan dalam bentuk fungsi Python dengan nama `deskew(img)`, fungsi ini mengkoreksi kemiringan citra tanda tangan pada citra yang telah melalui proses morfologi.

Tabel 5.5 Kode program *deskewing*

No	Kode Program
1	`def deskew(img):`
2	`    invert = cv2.bitwise_not(img)`
3	`    m = cv2.moments(invert)`
4	`    if abs(m['mu02']) < 1e-2:`
5	`        return img.copy()`
6	`    skew = m['mu11']/m['mu02']`
7	`    M = np.float32([[1, skew, -0.5*img_size*skew], [0, 1, 0]])`
8	`    img = cv2.warpAffine(img, M, (img_size, img_size), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)`

9	`    img = cv2.bitwise_not(img)`
10	`    return img`

Pada Tabel 5.5, nama fungsi dari metode didefinisikan pada baris 1, fungsi ini memiliki parameter input (`img`) untuk masukan citra. Selanjutnya baris ke 2 digunakan untuk menginvert setiap bit pada citra, hal ini dibutuhkan agar obyek tanda tangan yang berwarna hitam tidak bernilai 0 dan dapat dihitung image momentnya. Pada baris ke 3 digunakan untuk menghitung image moment pada input citra. Baris 4 dan 5 digunakan jika input citra tidak memerlukan deskewing maka akan mengembalikan citra inputan. Kemudian pada baris ke 6 digunakan untuk menghitung kemiringan berdasarkan *central moments*. Setelah itu nilai kemiringan yang didapat digunakan pada baris 7 dalam menghitung *affine transform* untuk mengkoreksi *skewness* atau kemiringan. Pada baris 8 nilai hasil kalkulasi *affine transform* diterapkan pada citra. Terakhir pada baris 9 hasil koreksi kemiringan citra diinvert kembali untuk menghitamkan warna obyek tanda tangan dan nilainya dikembalikan ke program utama pada baris 10.

5.2.2.5 Metode Histogram of Oriented Gradients

Dalam imlementasi metode *Histogram of Oriented Gradients* (HOG) terdapat dua bagian dalam program utama yang ditulis sebagai fungsi Python, yaitu fungsi insialisasi parameter HOG dengan nama `init_hog()` pada Tabel 5.6 dan fungsi kalkulasi HOG dengan nama `cal_hog(deskew)` pada Tabel 5.7.

Tabel 5.6 Kode program parameter HOG

No	Kode Program
1	`def init_hog():`
2	`    winSize = (300,300)`
3	`    blockSize = (150,150)`
4	`    blockStride = (75,75)`
5	`    cellSize = (75,75)`
6	`    nbins = 9`
7	`    derivAperture = 1`
8	`    winSigma = -1.`
9	`    histogramNormType = 0`
10	`    L2HysThreshold = 0.2`
11	`    gammaCorrection = 1`
12	`    nlevels = 64`
13	`    signedGradients = True`
14	`    hog=cv2.HOGDescriptor(winSize,blockSize,blockStride, cellSize,nbins,derivAperture,winSigma,histogramNormType, L2HysThreshold,gammaCorrection,nlevels,signedGradients)`
15	`    return hog`

Pada Tabel 5.6, parameter-parameter yang diperlukan untuk kalkulasi HOG didefinisikan dalam fungsi ini. Pada baris 1 nama fungsi dari metode didefiniskan, fungsi ini tidak memilik parameter. Baris 2, parameter `winSize` digunakan untuk menentukan besar resolusi piksel citra yang akan dihitung nilai HOGnya. Pada baris ke 3 parameter `blockSize` digunakan untuk menentukan ukuran blok dalam piksel. Pada baris ke 4 parameter `blockStride` digunakan untuk menentukan seberapa besar *overlap* piksel antara blok ketetanggaan dan mengontrol derajat

normalisasi kontras citra. Parameter `cellSize` pada baris 5 digunakan untuk menentukan seberapa besar ukuran sel HOG dihitung. Baris 6 parameter nbins digunakan untuk menentukan jumlah bin histogram gradient. Parameter `derivAperture` pada baris ke 7 untuk menghitung *image derivative*, namun parameter ini sudah tidak dipakai dalam fungsi `HOGDescriptor()`. Baris 8 `winSigma` sebagai parameter *Gaussian Smoothing*. Parameter `histogramNormType` pada baris 9 untuk menentukan tipe normalisasi histogram, parameter ini juga sudah tidak dipakai dalam fungsi `HOGDescriptor()` namun tetap harus didefinisikan agar menghindari error dalam penulisan program. Baris ke 10 `L2HysThreshold` sebagai parameter metode normalisasi L2-Hys. Parameter `gammaCorrection` pada baris 11 sebagai penanda apakah proses *preprocessing* koreksi *gamma* diperlukan atau tidak. Baris 12 `nlevels` sebagai parameter jumlah maksimum jendela deteksi. Parameter `signedGradients` pada baris 13 menunjukkan *signed gradient* digunakan atau tidak. Terakhir pada baris 14 parameter-parameter yang telah didefinisikan dimasukkan dalam fungsi `cv2.HOGDescriptor()` dan variabelnya dikembalikan ke program utama pada baris ke 15.

Tabel 5.7 Kode program komputasi HOG

No	Kode Program
1	`def cal_hog(deskew):`
2	`    for img in deskew:`
3	`        hog_descriptor.append(hog.compute(img))`
4	`    return hog_descriptor`

Pada Tabel 5.7 adalah fungsi untuk mengkalkulasi HOG, di baris 1 didefinisikan nama fungsi dan memiliki 1 parameter `deskew` untuk *input* citra yang telah dilakukan *deskewing*. Pada baris 2 dan 3 dihitung nilai HOG setiap gambar pada parameter *input* `deskew` dengan menggunakan fungsi `hog.compute`. Nilai-nilai HOG dari beberapa gambar digabung menjadi satu variabel dalam bentuk matriks dan dikembalikan ke program utama pada baris ke 4.

5.2.2.6 Metode *Support Vector Machine*

Implementasi metode *Support Vector Machine* (SVM) diterapkan dalam bentuk fungsi Python. Terdapat tiga bagian fungsi, yaitu bagian inisialisasi parameter untuk klasifikasi SVM yang ditunjukkan pada Tabel 5.8, bagian pelatihan data citra tanda tangan yang ditunjukkan pada Tabel 5.9, dan terakhir bagian prediksi data *input* citra tanda tangan yang ditunjukkan pada Tabel 5.10.

Tabel 5.8 Kode program inisialisasi parameter SVM

No	Kode Program
1	`def svmInit(C=100, gamma=1.0):`
2	`    model = cv2.ml.SVM_create()`
3	`    model.setGamma(gamma)`
4	`    model.setC(C)`
5	`    model.setKernel(cv2.ml.SVM_RBF)`
6	`    model.setType(cv2.ml.SVM_C_SVC)`
7	`    return model`

Pada Tabel 5.8, parameter-parameter yang diperlukan untuk klasifikasi SVM didefinisikan di fungsi ini. Pada baris 1 nama fungsi didefinisikan, fungsi ini jugamemberi nilai untuk parameter `C` dan `gamma`. Baris 2 dideklarasikan fungsi klasifikasi SVM yang ada pada openCV. Pada baris ke 3 dan 4 nilai parameter `gamma` dan `C` dimasukkan ke fungsi klasifikasi SVM. Selanjutnya pada baris 5 digunakan untuk mengatur tipe kernel SVM yang digunakan yaitu *Radial Basis Function* (RBF). Pada baris 6 digunakan untuk mengatur tipe SVM yang akan digunakan, disini menggunakan C_SVC (*C-Support Vector Classification*), selain umum digunakan tipe ini dapat digunakan untuk klasifikasi sebanyak *n-class* (n ≥ 2) dan dapat digunakan saat data latih tidak dapat dipisahkan secara linier. Terakhir pada baris 7 parameter-parameter yang telah dideklarasikan dikembalikan ke program utama sebagai model parameter klasikasi SVM yang akan digunakan.

Tabel 5.9 Kode program pelatihan data SVM

No	Kode Program
1	`def svmTrain(model, samples, labels):`
2	`    model.train(samples, cv2.ml.ROW_SAMPLE, labels)`
3	`    return model`

Pada Tabel 5.9 adalah fungsi untuk melakukan pelatihan data menggunakan klasifikasi SVM. Pada baris 1 didefinisikan nama fungsi dan parameternya, fungsi ini memiliki tiga parameter yaitu `model` digunakan untuk input model parameter SVM, parameter `samples` digunakan untuk input data train citra, dan parameter `labels` digunakan untuk input label dari data train. Baris ke 2 digunakan untuk memulai melatih data citra berdasarkan parameter model yang telah didefisikan sebelumnya. Kemudian pada baris 3 hasil pelatihan data dengan SVM dikembalikan ke program utama.

Tabel 5.10 Kode program prediksi SVM

No	Kode Program
1	`def svmPredict(model, data_test):`
2	`    return model.predict(data_test)[1]`

Pada Tabel 5.10 adalah fungsi untuk memprediksi data citra tanda tangan termasuk kelas yang mana. Baris ke 1 didefinisikan nama fungsi dan parameternya, terdapat dua parameter yaitu parameter `model` yang berfungsi sebagai input model pemisah pelatihan data dari data latih dan parameter `data_test` sebagai input data citra yang akan di prediksi. Selanjutnya pada baris ke 2 hasil prediksi SVM dikembalikan ke program utama.

5.2.3 Implementasi *Software*

Dalam subbab ini dijelaskan tentang implementasi *software* berdasarkan *flowchart-flowchart* yang telah disusun pada subbab perancangan *software* sebelumnya. Implementai dilakukan dengan kode program berbahasa Python yang nantinya dijalankan pada sistem operasi Rasbian yang ada dalam Raspberry Pi. Dalam implementasinya sistem memerlukan OpenCV sebagai penyedia library

komputasi citra, dan juga memerlukan *driver* I2C LCD untuk menampilkan *output* pada LCD. Berikut penjelasan mengenai masing-masing bagian program sistem.

5.2.3.1 Software Pelatihan Data

Pada bagian *software* pelatihan data sistem akan memuat data latih citra tanda tangan yang telah ada dalam Raspberry Pi, data latih tersebut kemudian akan dikalkulasi ciri HOG dan digunakan untuk pembentukan *hyperplane model* pada klasifikasi SVM. Dalam pembuatan program sistem ini, diperlukan *library-library* untuk mendukung beberapa fungsi yang digunakan dalam sistem. Untuk library yang dibutuhkan sistem dapat dilihat pada Tabel 5.11.

Tabel 5.11 Library yang dibutuhkan sistem

No	Kode Program
1	`import cv2`
2	`import os, os.path`
3	`import numpy as np`
4	`import time`
5	`import RPi.GPIO as GPIO`
6	`import I2C_LCD_driver`
7	`from picamera.array import PiRGBArray`
8	`from picamera import PiCamera`

Pada Tabel 5.11 library OpenCV dipanggil kedalam program yang ditunjukkan pada baris 1, library ini banyak digunakan dalam beberapa fungsi program untuk pengolahan citra. Dalam memuat file data latih diperlukan library yang dapat memanipulasi file, library ini ditunjukkan pada baris 2. Pada baris 3 library numpy digunakan untuk komputasi numerik pada pemograman Python, library ini umum digunakan untuk membangun *array* multidimensi pada komputasi citra. Pada baris 4 library *time* digunakan untuk modul fungsi yang berhubungan dengan waktu, seperti memberi delay pada program. Sistem ini memerlukan akses pin GPIO (*General-Purpose Input/Output*) untuk perangkat keras *push button*, oleh karena itu pemanggilan library untuk mengontrol GPIO dideklarasikan seperti pada baris 5, sedangkan untuk kontrol I2C pada LCD membutuhkan *library driver* pada baris 6 . Untuk mengakses modul kamera Raspberry Pi juga diperlukan library picamera yang ditunjukkan pada baris 7 dan 8.

Tabel 5.12 Insialisasi variabel yang dibutuhkan sistem

No	Kode Program
1	`label_asli = np.array([])`
2	`label_palsu = np.array([])`
3	`img_size = 300`
4	`GPIO.setmode(GPIO.BCM)`
5	`GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)`
6	`mylcd = I2C_LCD_driver.lcd()`

Pada Tabel 5.12 merupakan bagian program untuk insialisasi variabel-variabel yang akan dibutuhkan dalam sistem, variabel ini diinisalisasi untuk dapat diakses secara global dalam keseluruhan program. Untuk keperluan data latih, insialisasi label asli dan palsu ditunjukkan pada baris 1 dan 2, dan inisialisasi ukuran citra

yang akan dipakai pada baris ke 3 sebesar 300 piksel. Untuk keperluan perangkat keras, pada baris ke 4 dan 5 digunakan untuk inisialisasi pin GPIO 18 sebagai input untuk *push button*, sedangkan pada baris 6 digunakan untuk inisialisasi driver LCD.

Tabel 5.13 Implementasi fungsi mengambil daftar nama

No	Kode Program
1	`def person_list():`
2	`    directory_list = list()`
3	`    for dirs in os.listdir("Data"):`
4	`        directory_list.append(dirs)`
5	`    return directory_list`

Dalam menentukan data latih dari penanda tangan mana yang akan diverifikasi, sistem memerlukan identitas nama yang didapat dari daftar nama folder pada data latih, Tabel 5.13 merupakan fungsi program untuk mengambil daftar nama folder. Pada baris 1 merupakan pendefinisian dari nama fungsi, fungsi ini tidak memiliki parameter sebagai input, selanjutnya pada baris 2 dideklarasikan sebuah variabel bertipe *list* sebagai tempat untuk identitas nama nantinya. Pada baris 3 dan 4, program akan mengambil seluruh nama folder pada folder data latih dan disimpan dalam variabel yang telah disediakan sebelumnya. Terakhir variabel yang berisi daftar nama folder dikirim ke program utama sebagai daftar identitas nama.

Tabel 5.14 Implementasi fungsi mengambil *path file* data latih

No	Kode Program
1	`def list_gambar(namefolder):`
2	`image_path_list = []`
3	`    valid_image_extensions = [".jpg"]`
4	`    valid_image_extensions = [item.lower() for item in valid_image_extensions]`
5	`    for file in os.listdir(namefolder):`
6	`        extension = os.path.splitext(file)[1]`
7	`        if extension.lower() not in valid_image_extensions:`
8	`            continue`
9	`        image_path_list.append(os.path.join(namefolder, file))`
10	`    return image_path_list`

Pada Tabel 5.14 merupakan sebuah fungsi program yang digunakan untuk mengambil daftar alamat *path file* data latihTabel 5.15 yang disimpan dalam Raspberry Pi, hasil dari fungsi ini nantinya akan digunakan sistem untuk memuat file data latih. Baris 1 digunakan sebagai definisi nama fungsi dan parameternya, parameter ini dapat diisi dengan *path folder* data latih citra tanda tangan asli maupun tanda tangan palsu, variabel untuk menyimpan daftar path file juga dideklarasikan pada baris 2. Pada baris 3 dan 4 berfungsi sebagai inisialisasi ekstensi file apa saja yang valid untuk digunakan sebagai data latih, untuk penelitian ini hanya menggunakan ekstensi file .jpg. Selanjutnya program akan memindai setiap file pada folder input dan mengecek apakah ekstensi dari file tersebut valid atau tidak yang ditunjukkan pada baris 5 hingga 8. Pada baris 9 dan

10, hasil dari daftar path file yang valid disimpan dalam variabel `image_path_list` dan dikembalikan pada program utama.

Tabel 5.15 Implementasi fungsi memuat citra data latih

No	Kode Program
1	`def load_gambar(path):`
2	`    daftar_gambar = np.array([], dtype=np.uint8)`
3	`    for imagePath in path:`
4	`        gambar = cv2.imread(imagePath, 0)`
5	`        if gambar is not None:`
6	`            daftar_gambar = np.append(daftar_gambar, gambar)`
7	`        elif gambar is None:`
8	`            print ('Error loading: ' + imagePath)`
9	`            continue`
10	`    daftar_gambar = daftar_gambar.reshape(-1, img_size, img_size)`
11	`    return daftar_gambar`

Bagian program pada Tabel 5.15 merupakan fungsi untuk memuat daftar citra data latih dari *path file* yang telah valid sebelumnya, hasil dari fungsi program ini berupa daftar citra data latih yang nantinya akan diekstraksi ciri HOG citranya. Pada baris 1 didefinisikan nama fungsi dengan parameternya, parameter ini digunakan sebagai input daftar *path file* dari fungsi sebelumnya. Selanjutnya dideklarasikan variabel untuk tempat program memuat citra yang ditunjukkan pada baris 2. Pada baris program 3 hingga 10, program akan memuat setiap *path file* citra dan disimpan ke dalam variabel yang telah disediakan sebelumnya, dalam pemuatan citra data latih, program langsung mengkonversi colorspace citra dari RGB ke *grayscale* yang ditunjukkan pada baris 4, hal ini agar citra langsung dapat dihitung ciri HOGnya. Terakhir variabel hasil citra dikembalikan ke program utama yang ditunjukkan pada baris ke 11.

Tabel 5.16 Implementasi pengaturan folder data latih

No	Kode Program
1	`person_name = person_list()`
2	`while True:`
3	`    mylcd.lcd_display_string("Nama : ", 1)`
4	`    person = input('Masukkan nama : ')`
5	`    mylcd.lcd_display_string(person, 2)`
6	`    time.sleep(1)`
7	`    if person in person_name:`
8	`        asliDir = 'Data/{}/asli/300'.format(person)`
9	`        palsuDir = 'Data/{}/palsu/300'.format(person)`
10	`        break`
11	`    else:`
12	`        mylcd.lcd_display_string("Nama salah", 1)`
13	`        print('Nama salah')`
14	`        time.sleep(1)`
15	`        mylcd.lcd_clear()`

Sebelum dilakukan data latih, sistem perlu mengatur folder penanda tangan mana yang akan digunakan sebagai data latih, bagian program pada Tabel 5.16 merupakan proses untuk mengatur folder data latih sesuai identitas nama yang

dimasukkan *user* kedalam sistem. Pertama-tama pada baris 1 sistem akan memanggil fungsi `person_list()` yang digunakan untuk mengambil daftar identitas nama. Selanjutnya pada baris 3 hingga 5 sistem akan meminta identitas nama dan menampilkannya pada LCD, *delay* pada baris 6 digunakan untuk memberi waktu LCD untuk menampilkan identitas nama yang dimasukkan. Jika identitas nama yang dimasukkan cocok dengan salah satu daftar nama folder, maka program akan mengatur folder data latih sesuai dengan identitas nama tersebut dan melanjutkan proses ke program selanjutnya yang ditunjukkan pada program baris 7 hingga 10. Sedangkan jika identitas nama tidak ada yang sesuai, sistem akan menampilkan "Nama salah" dan meminta user memasukkan identitas nama lagi yang ditunjukkan pada baris 12 hingga 15.

Tabel 5.17 Implementasi memuat data latih

No	Kode Program
1	`mylcd.lcd_display_string("Loading ...", 1)`
2	`print ('Loading Data Asli ...')`
3	`path_asli = list_gambar(asliDir)`
4	`data_asli = load_gambar(path_asli)`
5	`deskew_asli = list(map(deskew, data_asli))`
6	`print ('Loading Data Palsu ...')`
7	`path_palsu = list_gambar(palsuDir)`
8	`data_palsu = load_gambar(path_palsu)`
9	`deskew_palsu = list(map(deskew, data_palsu))`

Setelah proses pengaturan folder sesuai identitas nama, selanjutnya yaitu proses memuat data latih, pada Tabel 5.17 adalah bagian program untuk memuat data latih citra tanda tangan asli dan tanda tangan palsu yang digunakan untuk proses klasifikasi. Baris 1, 2 dan 6 menampilkan "*loading*" sebagai penanda bahwa sistem memasuki proses memuat data. Selanjutnya untuk memuat data latih citra tanda tangan asli, pada baris 3 program memanggil fungsi untuk mengambil daftar *path file* dengan parameter folder data citra tanda tangan asli dan data tersebut dimuat dengan fungsi pada baris 4, kemudian semua data latih tersebut dikoreksi kemiringan tulisan dengan fungsi *deskewing* pada baris 5. Dalam memuat data latih citra tanda tangan palsu yang ditunjukkan pada program baris 7 hingga 9, prosesnya sama dengan memuat data latih citra tanda tangan asli, perbedaannya dengan merubah parameter folder ke folder data latih citra tanda tangan palsu dan juga diterapkan koreksi kemiringan dengan fungsi *deskewing*.

Tabel 5.18 Implementasi kalkulasi HOG

No	Kode Program
1	`print('Menghitung HoG ... ')`
2	`hog = init_hog();`
3	`hog_descriptor_train = []`
4	`for img in deskew_asli:`
5	`    hog_descriptor_train.append(hog.compute(img))`
6	`for img in deskew_palsu:`
7	`    hog_descriptor_train.append(hog.compute(img))`
8	`hog_descriptor_train = np.squeeze(hog_descriptor_train)`

Data latih citra tanda tangan asli dan palsu yang telah dimuat selanjutnya akan dihitung ciri HOG citranya, Tabel 5.18 merupakan bagian program untuk proses kalkulasi ciri HOG. Pertama pada baris 1 program akan menampilkan "Menghitung HOG" sebagai penanda bahwa sistem masuk dalam proses kalkulasi HOG. Selanjutnya sistem melakukan insialisasi parameter-parameter yang diperlukan untuk metode HOG dengan fungsi `init_hog()` yang ditunjukkan pada program baris 2, variabel sebagai tempat menampung ciri HOG dari data latih juga dideklarasikan yang ditunjukkan pada program baris 3. Pada baris 4 dan 5 mulai dilakukan perhitungan ciri HOG citra tanda tangan asli, perhitungan dilakukan dengan fungsi `hog.compute()` pada setiap citra di variabel `deskew_asli` yang merupakan data latih citra tanda tangan asli, baris 6 dan 7 juga menghitung ciri HOG pada setiap citra tanda tangan palsu yang ada pada variabel `deskew_palsu`. Terakhir variabel yang berisi ciri HOG tersebut dikurangi dimensi arraynya agar dapat diklasifikasikan dengan fungsi klasifikasi SVM.

Tabel 5.19 Implementasi pelatihan data dengan klasifikasi SVM

No	Kode Program
1	`print ('Training SVM model ...')`
2	`model = svmInit()`
3	`label_asli = np.repeat(1,len(path_asli))`
4	`label_palsu = np.repeat(2,len(path_palsu))`
5	`label_train = np.append(label_asli, label_palsu)`
6	`svmTrain(model, hog_descriptor_train, label_train)`
7	`mylcd.lcd_display_string("Loaded ", 1)`

Setelah ciri HOG citra data latih didapat, selanjutnya masuk ke proses klasifikasi, Tabel 5.19 merupakan bagian program dari proses pelatihan data dengan klasifikasi SVM. Proses ini diawali dengan menampilkan "Training SVM model" dengan fungsi `print` pada baris 1 sebagai penanda bahwa sistem masuk proses klasifikasi. Sebelum dilakukan pelatihan data klasifikasi pada baris 2 diperlukan inisialisasi parameter-parameter yang diperlukan untuk klasifikasi SVM, pada baris 3 hingga 5 label untuk keperluan kelas klasifikasi juga diinisialisasi dengan memakai label *integer* '1' untuk data asli dan label *integer* '2' untuk data palsu. Selanjutanya pada baris 6 dilakukan pelatihan data dengan fungsi `SVMTrain()`, fungsi ini memiliki tiga parameter yaitu `model` yang berisi parameter yang diperlukan dalam klasifikasi SVM, `hog_descriptor_train` yang berisi ciri HOG dari citra data latih, dan `label_train` yang berisi label untuk data latih. Terakhir program baris 7 menampilkan "*Loaded*" pada LCD menandakan bahwa proses pelatihan data telah selesai dan masuk ke proses verifikasi tanda tangan.

5.2.3.2 Software Verifikasi Tanda Tangan

Saat sistem mulai memasuki proses verifikasi tanda tangan sistem akan mulai mengaktifkan kamera dan menampilkan video pada monitor, dengan *push button* sebagai pemicunya kamera akan menangkap data uji citra tanda tangan yang nantinya diproses dan diklasifikasikan. Sebelum kamera dapat digunakan, parameter kamera perlu diinisialisasi terlebih dahulu, Tabel 5.20 merupakan fungsi program untuk inisialisasi parameter kamera.

Tabel 5.20 Implementasi fungsi insialisasi parameter kamera

No	Kode Program
1	`def videoInit():`
2	`    camera = PiCamera()`
3	`    camera.resolution = (480, 480)`
4	`    camera.framerate = 40`
5	`    rawCapture = PiRGBArray(camera, size=(480, 480))`
6	`    time.sleep(0.1)`
7	`    video = camera.capture_continuous(rawCapture, format="bgr", use_video_port=True)`
8	`    return video, rawCapture`

Pada Tabel 5.20 parameter yang diperlukan kamera ditulis dalam bentuk fungsi program, pertama baris 1 pada program digunakan sebagai nama fungsi dan saat diperlukan dapat memanggil fungsi ini. Fungsi dari *library* `PiCamera()` dimasukkan ke variabel kamera pada baris 2 agar lebih mudah untuk mengatur parameter kamera. Pada baris 3 dan 4 resolusi dan framerate kamera diinisalisasi, resolusi menggunakan 480x480 piksel dengan framerate 40 fps agar tidak terlalu membebani sistem. Selanjutanya baris 5 digunakan untuk mengakses nilai tiga dimensi RGB *numpy array* dari citra yang ditangkap kamera, delay pada baris 6 diperlukan untuk mempersiapkan kamera sebelum diaktifkan. Inisialisasi variabel `video` pada baris 7 dilakukan dengan fungsi `camera.capture_continuous` untuk kamera menangkap citra secara terus menerus. Terakhir nilai variabel `video` dan `rawCapture` yang berisi parameter kamera dikembalikan ke program utama.

Tabel 5.21 Implementasi fungsi merubah ukuran dimensi citra

No	Kode Program
1	`def resize(inputSize, img):`
2	`    r = float(inputSize) / img.shape[1]`
3	`    dim = (inputSize, int(img.shape[0] * r))`
4	`    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)`
5	`    return resized`

Dalam pengujian data, dimensi citra data uji yang ditangkap kamera terlebih dahulu disamakan dengan dimensi citra data latih, pada Tabel 5.21 merupakan fungsi untuk merubah ukuran dimensi citra. Nama fungsi didefinisikan pada baris 1 dengan memiliki dua parameter yaitu `inputSize` untuk *input* nilai ukuran piksel yang diinginkan dan `img` untuk *input* citra yang akan dirubah. Baris 2 menghitung *image ratio* atau aspek perbandingan ukuran dimensi citra yang lama dengan dimensi yang baru, dan baris 3 membuat ukuran dimensi citra yang baru dari hasil ukuran piksel *input* dan perkalian *ratio* sebelumnya, hal ini dimaksudkan untuk menjaga proporsional dari dimensi citra. Selanjutnya pada baris 4 ukuran dimensi yang didapat diaplikasikan pada *input* citra dan nilainya dikembalikan ke program utama pada baris 5.

Tabel 5.22 Implementasi fungsi *preprocessing*

No	Kode Program
1	`def preprocessing(img):`
2	`    gray = grayscale(img)`
3	`    rez = resize(300, gray)`
4	`    image = threshold(rez)`
5	`    image = morfologi(image)`
6	`    image = deskew(image)`
7	`    return image`

Sebelum ciri HOG citra tanda tangan yang ditangkap kamera dapat diambil, terlebih dahulu dilakukan *preprocessing* terhadap citra tersebut, pada Tabel 5.22 merupakan fungsi program untuk *preprocessing* citra sebelum dilakukan ekstraksi ciri HOG. Pada baris 1 didefinisikan nama fungsi dan parameter untuk *input* citra. Proses pertama dalam *preprocessing* citra yaitu merubah *colorspace* citra dari RGB menjadi *grayscale* dengan memanggil fungsi `grayscale()` dalam program pada baris 2. Dimensi citra tanda tangan yang ditangkap kamera juga disamakan dengan dimensi data latih menggunakan fungsi `resize()` pada baris 3. Fungsi segmentasi `threshold()` yang telah didefinisikan sebelumnya dipanggil pada baris 3 untuk memisahkan objek tanda tangan dengan *background* pada citra, selanjutnya *noise* hasil dari *thresholding* yang masih ada dihilangkan dengan fungsi `morfologi()` pada baris 5. Terakhir citra tanda tangan dikoreksi kemiringan dengan fungsi `deskew()` pada baris 6 dan nilainya dikembalikan ke program utama pada baris 7.

Tabel 5.23 Implementasi proses verifikasi tanda tangan

No	Kode Program
1	`video, rawCapture = videoInit()`
2	`count = 0`
3	`for frame in video:`
4	`    input_state = GPIO.input(18)`
5	`    live = frame.array`
6	`    cv2.imshow("Kamera", live)`
7	`    key = cv2.waitKey(1) & 0xFF`
8	`    if input_state == False:`
9	`        count = count + 1`
10	`        mylcd.lcd_clear()`
11	`        image = frame.array`
12	`        cv2.imshow("Image", resize(300, image))`
13	`        pre_img = preprocessing(image)`
14	`        hog_d = hog.compute(pre_img)`
15	`        hog_d = np.transpose(hog_d)`
16	`        predict = svmPredict(model, hog_d)`
17	`        predict = int(predict)`
18	`        mylcd.lcd_display_string("Output ke :{}".format(count), 1)`
19	`        if predict == 1:`
20	`            print ('Asli')`
21	`            mylcd.lcd_display_string("Asli", 2)`
22	`        elif predict == 2:`
23	`            print ('Palsu')`
24	`            mylcd.lcd_display_string("Palsu", 2)`
25	`        else:`
26	`            print ('Tidak Terdefinisi')`

```
27          time.sleep(0.1)
28          cv2.imwrite(person + str(count) + ".jpg", image)
29      if key == ord("q"):
30          break
31      rawCapture.truncate(0)
32  cv2.destroyAllWindows()
```

Bagian program pada Tabel 5.23 merupakan program utama untuk proses verifikasi tanda tangan yang mana pengambilan citra tanda tangannya diambil langsung dengan kamera sesuai penekanan *push button* dan hasil verifikasinya ditampilkan pad LCD, program ini akan terus berjalan dan akan berhenti jika ada penekanan tombol "Q" pada keyboard. Pertama parameter kamera diinisialisasi dengan memanggil fungsi `videoInit()` pada baris 1, variabel count di baris 2 digunakan sebagai penampung jumlah output dari sistem. Setiap looping program yang ada pada baris 3, program akan mengecek penekanan *push button* yang dihubungkan pada pin GPIO 18 dan mengecek penekanan keyboard komputer yang ditunjukkan pada baris 4 dan 7. Pada baris 5 dan 6 setiap frame yang ditangkap kamera ditampilkan pada layar secara terus menerus.

Saat ada penekanan *push button* yang ditunjukkan pada baris 8, program akan melakukan increment jumlah output program di baris 9 dan membersihkan tulisan pada LCD untuk persiapan output sistem di baris 10. Program juga mengambil nilai array dari frame citra tanda tangan yang ditangkap kamera, citra tersebut ditampilkan ke monitor dan dilakukan preprocessing citra yang ditunjukkan pada baris 11 hingga 13. Pada baris 14 dan 15 setelah melalui preprocessing, citra akan dihitung ciri HOG dengan memanggil fungsi `hog.compute()`. Berdasarkan ciri HOG citra tersebut diklasifikasikan dengan fungsi `svmPredict()` apakah tanda tangan tersebut termasuk kelas tanda tangan asli atau palsu yang ditunjukkan pada baris 16 dan 17. Pada baris 18 *output* ditampilkan ke LCD beserta jumlah *output* yang telah ditampilkan, jika label prediksi bernilai 1 *output* akan menampilkan asli, namun jika prediksi bernilai 2 *output* akan menampilkan palsu seperti yang ditunjukkan pada program baris 19 hingga 26. Delay pada baris 27 berfungsi untuk menghindari penekanan ganda pada *push button*, pada baris 28 citra yang ditangkap kamera disimpan dalam disk jika diperlukan untuk dokumentasi. Jika ada penekanan tombol "Q" pada keyboard program akan keluar dari looping dan mengakhiri program seperti yang ditunjukkan pada baris 29 dan 30. Program pada baris 31 dan 32 digunakan untuk menghapus jendela OpenCV saat program memuat frame baru yang ditangkap kamera atau saat program berhenti.

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini membahas tentang pengujian, hasil pengujian, dan analisis hasil pengujian dari sistem. Pengujian dan analisis dilakukan untuk mengetahui apakah sistem telah sesuai dengan tujuan penelitian. Pengujian pada penelitian ini meliputi pengujian tingakat keberhasilan sistem yang berupa seberapa akurat sistem melakukan verifikasi citra tanda tangan, pengujian waktu proses dibutuhkan sistem dalam melatih data, dan pengujian waktu proses yang dibutuhkan sistem untuk verifikasi citra tanda tangan.

6.1 Pengujian Akurasi Verifikasi Citra Tanda Tangan

6.1.1 Tujuan

Tujuan dari pengujian akurasi verifikasi citra tanda tangan yaitu untuk mengetahui akurasi dari metode klasifikasi SVM dalam memverifikasi citra tanda tangan. Citra tanda tangan yang diverifikasi merupakan citra data uji yang ditangkap kamera dan bukan termasuk citra data latih. Output verifikasi berupa prediksi klasifikasi SVM yang menyatakan tanda tangan yang diuji adalah tanda tangan asli atau palsu, hasil dari pengujian berupa persentase dari akurasi sistem.

6.1.2 Alat

Alat yang digunakan dalam pengujian akurasi verifikasi citra tanda tangan antara lain sebagai berikut:

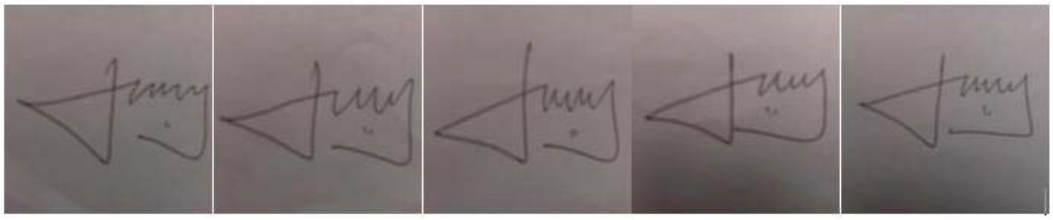
1. Raspberry Pi 3 Model B
2. Pi Camera
3. LCD 16x2 dengan I2C
4. Push Button
5. Laptop
6. Kabel ethernet
7. Data latih dan data uji tanda tangan

6.1.3 Prosedur Pengujian

Agar berjalan dengan semestinya, dalam pengujian akurasi verifikasi citra tanda tangan dilakukan metode sebagai berikut:

1. Melakukan pemasangan Pi Camera dan *push button* pada Raspberry Pi.
2. Hubungkan Raspberry Pi dengan laptop menggunakan kabel ethernet.
3. Hubungkan Raspberry Pi dengan sumber daya.
4. Buka aplikasi VNC pada laptop untuk mengakses Raspberry Pi dan menjalankan program.
5. Memberikan identitas nama penanda tangan pada sistem dan input data uji tanda tangan pada kamera disertai penekanan push button sebagai pemicu kamera menangkap citra tanda tangan.

6. Lakukan pengujian pada data uji sebanyak 30 penanda tangan dengan masing-masing penanda tangan terdapat 5 tanda tangan asli dan 5 tanda tangan palsu, seperti contoh data latih pada Gambar 6.1



Gambar 6.1 Contoh data uji

7. Catat dan amati data hasil keluaran dari sistem.
8. Menghitung akurasi verifikasi dari sistem dengan persamaan:

$$Akurasi = \left(100\% - \frac{FAR + FRR}{2}\right)\%$$

Dimana:

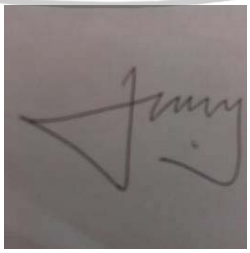
$$False\ Rejection\ Rate = \frac{Jumlah\ False\ Rejection}{Jumlah\ tanda\ tangan\ asli} \times 100\%$$

$$False\ Acceptance\ Rate = \frac{Jumlah\ False\ Acceptance}{Jumlah\ tanda\ tangan\ palsu} \times 100\%$$

9. Menganalisa hasil dan mengambil kesimpulan.

6.1.4 Hasil

Pengujian dilakukan dengan memasukkan identitas atau inisial nama penanda tangan yang akan diverifikasi, inisial penanda tangan pertama yang akan diuji yaitu "ABD". Setelah itu mengambil data uji citra tanda tangan menggunakan kamera yang dipicu oleh *push button* seperti pada Gambar 6.2.



Gambar 6.2 Citra tanda tangan yang ditangkap kamera

Hasil dari verifikasi citra tanda tangan tersebut dapat dilihat dari keluaran sistem pada LCD seperti yang ditunjukkan pada Gambar 6.3, dari Gambar 6.3 hasil verifikasi dari tanda tangan tersebut adalah "Asli".

Gambar 6.3 Hasil verifikasi citra tanda tangan

Hasil keseluruhan dari pengujian verifikasi untuk citra tanda tangan asli ditunjukkan pada Tabel 6.1, pada pengujian ini dihitung *False Rejection* pada setiap penanda tangan, *False Rejection* terjadi saat sistem memberikan *output* "Palsu" pada tanda tangan asli. Hasil *output* dari pengujian sistem dinilai benar apabila sistem memberikan *output* "Asli" pada tanda tangan asli.

Tabel 6.1 Hasil pengujian verifikasi citra tanda tangan asli

No.	Inisial Nama	Hasil Verifikasi					*False Rejection*
		Citra ke-1	Citra ke-2	Citra ke-3	Citra ke-4	Citra ke-5	
1	ABD	Asli	Asli	Asli	Asli	Asli	0
2	ADN	Asli	Asli	Asli	Asli	Asli	0
3	AGN	Asli	Asli	Asli	Asli	Asli	0
4	ALN	Asli	Asli	Asli	Asli	Asli	0
5	AND	Asli	Asli	Asli	Asli	Asli	0
6	ANG	Asli	Asli	Asli	Asli	Asli	0
7	ANS	Asli	Palsu	Asli	Asli	Palsu	2
8	ARD	Asli	Asli	Palsu	Palsu	Asli	2
9	AYX	Asli	Asli	Asli	Asli	Asli	0
10	DYH	Asli	Asli	Asli	Asli	Asli	0
11	EDX	Palsu	Palsu	Palsu	Asli	Palsu	4
12	ELS	Asli	Asli	Palsu	Palsu	Palsu	3
13	ENX	Asli	Asli	Asli	Asli	Asli	0
14	FRD	Asli	Asli	Asli	Palsu	Asli	1
15	HDR	Asli	Asli	Palsu	Palsu	Palsu	3
16	INT	Asli	Asli	Asli	Asli	Asli	0
17	IQB	Asli	Asli	Asli	Asli	Asli	0
18	KHR	Asli	Asli	Asli	Asli	Asli	0
19	MSB	Asli	Asli	Asli	Asli	Asli	0
20	OGY	Asli	Asli	Asli	Asli	Asli	0
21	PSP	Asli	Asli	Asli	Asli	Asli	0
22	PTR	Asli	Asli	Asli	Asli	Asli	0
23	RYN	Asli	Asli	Asli	Asli	Asli	0
24	RFD	Asli	Asli	Asli	Asli	Asli	0
25	RSM	Asli	Asli	Asli	Asli	Asli	0
26	RZT	Asli	Asli	Asli	Asli	Asli	0
27	SPT	Asli	Asli	Asli	Asli	Palsu	1

28	UMR	Asli	Asli	Asli	Asli	Palsu	1
29	YNS	Palsu	Asli	Palsu	Palsu	Asli	3
30	ZKY	Asli	Palsu	Asli	Palsu	Palsu	3
Total False Rejection							**23**

Dari hasil pengujian citra tanda tangan asli pada Tabel 6.1 didapat nilai *False Rejection Rate* sebagai berikut:

$$FRR = \frac{23}{150} \times 100\% = 15.33\%$$

Untuk hasil pengujian verifikasi untuk citra tanda tangan palsu dapat dilihat pada Tabel 6.2, pada pengujian ini dihitung *False Acceptance* pada setiap penanda tangan, *False Acceptance* terjadi saat sistem memberikan *output* "Asli" pada tanda tangan palsu. Hasil *output* dari pengujian sistem dinilai benar apabila sistem memberikan *output* "Palsu" pada tanda tangan palsu.

Tabel 6.2 Hasil pengujian verifikasi citra tanda tangan palsu

No.	Inisial Nama	Hasil Verifikasi					*False Acceptance*
		Citra ke-1	Citra ke-2	Citra ke-3	Citra ke-4	Citra ke-5	
1	ABD	Palsu	Palsu	Palsu	Palsu	Palsu	0
2	ADN	Palsu	Palsu	Palsu	Palsu	Palsu	0
3	AGN	Palsu	Palsu	Palsu	Palsu	Palsu	0
4	ALN	Palsu	Palsu	Palsu	Palsu	Palsu	0
5	AND	Palsu	Palsu	Palsu	Palsu	Palsu	0
6	ANG	Asli	Asli	Asli	Asli	Asli	5
7	ANS	Palsu	Palsu	Asli	Asli	Palsu	2
8	ARD	Palsu	Palsu	Palsu	Palsu	Palsu	0
9	AYX	Palsu	Palsu	Palsu	Palsu	Palsu	0
10	DYH	Palsu	Palsu	Palsu	Palsu	Palsu	0
11	EDX	Palsu	Palsu	Palsu	Palsu	Palsu	0
12	ELS	Palsu	Palsu	Asli	Palsu	Palsu	1
13	ENX	Palsu	Palsu	Asli	Palsu	Palsu	1
14	FRD	Palsu	Palsu	Palsu	Palsu	Palsu	0
15	HDR	Palsu	Asli	Palsu	Palsu	Palsu	1
16	INT	Palsu	Palsu	Palsu	Palsu	Palsu	0
17	IQB	Palsu	Palsu	Palsu	Asli	Palsu	1
18	KHR	Asli	Asli	Palsu	Asli	Palsu	3
19	MSB	Palsu	Palsu	Palsu	Palsu	Palsu	0
20	OGY	Palsu	Palsu	Palsu	Palsu	Palsu	0
21	PSP	Palsu	Palsu	Palsu	Palsu	Palsu	0
22	PTR	Palsu	Palsu	Palsu	Palsu	Palsu	0
23	RYN	Palsu	Palsu	Palsu	Palsu	Palsu	0
24	RFD	Palsu	Palsu	Palsu	Palsu	Palsu	0
25	RSM	Palsu	Palsu	Asli	Palsu	Palsu	1

26	RZT	Palsu	Palsu	Palsu	Palsu	Palsu	0
27	SPT	Palsu	Palsu	Palsu	Palsu	Palsu	0
28	UMR	Palsu	Palsu	Palsu	Palsu	Palsu	0
29	YNS	Palsu	Palsu	Palsu	Palsu	Palsu	0
30	ZKY	Palsu	Palsu	Palsu	Palsu	Palsu	0
Total False Acceptance							**15**

Dari hasil pengujian citra tanda tangan asli pada Tabel 6.2 didapat nilai *False Acceptance Rate* sebagai berikut:

$$FAR = \frac{15}{150} \times 100\% = 10\%$$

Dari nilai FRR dan FAR didapat nilai akurasi verifikasi sebagai berikut:

$$Akurasi = 100\% - \left(\frac{15{,}33\% + 10\%}{2}\right)$$

$$= 100\% - 12.67\% = 87.33\%$$

6.1.5 Analisis

Pengujian dilakukan pada 30 penanda tangan yang berbeda dengan memakai data latih sebanyak 300 citra tanda tangan asli dan 300 citra tanda tangan palsu, dan memakai data uji sebanyak 150 citra tanda tangan asli dan 150 citra tanda tangan palsu. Berdasarkan hasil pengujian pada Tabel 6.1 dan Tabel 6.2, sistem dapat melakukan verifikasi citra tanda tangan dengan tingkat akurasi sebesar 87.33%. Dari hasil observasi, data latih sangat menentukan akurasi sistem, *False Rejection* banyak ditemukan pada data latih dari penanda tangan yang mempunyai pola tanda tangan yang bervariasi pada setiap tanda tangannya. Hal ini menyebabkan sistem memberikan output "palsu" pada tanda tangan asli sehingga tingkat akurasi sistem menurun. Selain itu sistem juga memberikan output yang salah pada beberapa citra tanda tangan palsu yang memilki tingkat kemiripan yang tinggi dengan tanda tangan asli.

6.2 Pengujian Waktu Proses Pelatihan Data

6.2.1 Tujuan

Sebelum sistem dapat melakukan verifikasi, sistem membutuhkan waktu untuk melakukan proses pelatihan data yang digunakan untuk klasifikasi. Tujuan dari pengujian ini untuk mengetahui berapa lama waktu yang dibutuhkan sistem untuk menyelesaikan proses pelatihan data pada setiap penanda tangan.

6.2.2 Alat

Alat yang digunakan dalam pengujian waktu proses pelatihan data untuk klasifikasi antara lain sebagai berikut:

1. Raspberry Pi 3 Model B

2. Pi Camera
3. LCD 16x2 dengan I2C
4. Push Button
5. Laptop
6. Kabel ethernet
7. Data latih tanda tangan

6.2.3 Prosedur Pengujian

Agar berjalan dengan semestinya, dalam pengujian waktu proses pelatihan data untuk klasifikasi dilakukan metode sebagai berikut:

1. Melakukan pemasangan Pi Camera dan *push button* pada Raspberry Pi.
2. Hubungkan Raspberry Pi dengan laptop menggunakan kabel ethernet.
3. Hubungkan Raspberry Pi dengan sumber daya.
4. Buka aplikasi VNC pada laptop untuk mengakses Raspberry Pi dan menjalankan program.
5. Memberikan *input* identitas nama pada sistem.
6. Hitung waktu pemrosesan sistem dari proses sistem memuat data hingga proses klasifikasi data.
7. Lakukan pengujian pada data latih tanda tangan asli dan palsu dari 30 penanda tangan.
8. Catat dan amati data hasil waktu pemrosesan dari sistem.

6.2.4 Hasil

Hasil pengujian waktu proses pelatihan data untuk klasifikasi ditunjukkan pada Tabel 6.3 sebagai berikut.

Tabel 6.3 Hasil pengujian waktu proses pelatihan data

No.	Inisial Nama	Waktu Proses (s)
1	ABD	1,44
2	ADN	1,40
3	AGN	1,47
4	ALN	1,46
5	AND	1,49
6	ANG	1,47
7	ANS	1,48
8	ARD	1,46
9	AYX	1,44
10	DYH	1,44
11	EDX	1,47
12	ELS	1,46
13	ENX	1,43
14	FRD	1,44
15	HDR	1,45

16	INT	1,46
17	IQB	1,45
18	KHR	1,45
19	MSB	1,44
20	OGY	1,48
21	PSP	1,47
22	PTR	1,44
23	RYN	1,45
24	RFD	1,48
25	RSM	1,45
26	RZT	1,45
27	SPT	1,45
28	UMR	1,46
29	YNS	1,45
30	ZKY	1,45
Rata-rata waktu proses		**1,45**

Berdasarkan hasil pengujian pada Tabel 6.3, dari 30 kali pengujian didapat nilai rata-rata waktu proses pelatihan data klasifikasi sebesar 1,45 detik.

6.2.5 Analisis

Berdasarkan hasil pengujian pada Tabel 6.3, untuk setiap penanda tangan sistem memerlukan rata rata 1,45 detik untuk memproses data latih citra tanda tangan, mulai dari proses mengambil citra data latih, menghitung ciri HOG citra, dan pelatihan data klasifikasi. Waktu pemrosesan data latih citra tanda tangan pada masing-masing penanda tangan tidak terlalu berbeda jauh. Waktu pemrosesan tercepat sistem untuk melatih data yaitu 1,40 detik dan waktu terlama sebesar 1,49 detik yang hanya mempunyai selisih 0,09 detik, hal ini dikarenakan komputasi program ditangani oleh proses *scheduling* dari sistem operasi Raspbian.

6.3 Pengujian Waktu Proses Verifikasi Tanda Tangan

6.3.1 Tujuan

Pengujian waktu proses verifikasi data untuk mengetahui seberapa lama waktu proses yang dibutukan sistem untuk memberikan output verifikasi dari citra tanda tangan yang ditangkap kamera. Waktu proses dihitung dari mengambil citra tanda tangan dengan kamera, pemrosesan citra, hingga mengeluarkan hasil verifikasi.

6.3.2 Alat

Alat yang digunakan dalam pengujian waktu proses verifikasi tanda tangan untuk klasifikasi antara lain sebagai berikut:

1. Raspberry Pi 3 Model B
2. Pi Camera

3. LCD 16x2 dengan I2C
4. Push Button
5. Laptop
6. Kabel ethernet
7. Data latih dan data uji tanda tangan

6.3.3 Prosedur Pengujian

Agar berjalan dengan semestinya, dalam pengujian waktu proses verifikasi tanda tangan untuk klasifikasi dilakukan metode sebagai berikut:

1. Melakukan pemasangan Pi Camera dan *push button* pada Raspberry Pi.
2. Hubungkan Raspberry Pi dengan laptop menggunakan kabel ethernet.
3. Hubungkan Raspberry Pi dengan sumber daya.
4. Buka aplikasi VNC pada laptop untuk mengakses Raspberry Pi dan menjalankan program.
5. Memberikan *input* identitas nama pada sistem.
6. Hitung waktu pemrosesan sistem dari proses sistem dari proses pengambilan citra tanda tangan hingga sistem mengeluarkan hasil verifikasi.
7. Lakukan pengujian pada data uji tanda tangan asli dan palsu dari 30 penanda tangan.
8. Catat dan amati data hasil waktu pemrosesan dari sistem.

6.3.4 Hasil

Hasil pengujian waktu proses verifikasi pada data uji citra tanda tangan asli dapat dilihat pada Tabel 6.4.

Tabel 6.4 Waktu proses verifikasi tanda tangan asli

No.	Inisial Nama	Waktu Proses (s)					Rata-Rata (s)
1	ABD	0,264	0,206	0,216	0,216	0,247	0,230
2	ADN	0,246	0,212	0,203	0,214	0,211	0,217
3	AGN	0,204	0,247	0,211	0,238	0,248	0,230
4	ALN	0,224	0,238	0,232	0,256	0,228	0,236
5	AND	0,244	0,262	0,268	0,251	0,236	0,252
6	ANG	0,258	0,217	0,267	0,208	0,231	0,236
7	ANS	0,206	0,214	0,248	0,224	0,246	0,228
8	ARD	0,267	0,236	0,209	0,220	0,262	0,239
9	AYX	0,231	0,264	0,248	0,208	0,222	0,235
10	DYH	0,276	0,261	0,233	0,275	0,250	0,259
11	EDX	0,251	0,236	0,209	0,237	0,207	0,228
12	ELS	0,239	0,224	0,249	0,253	0,215	0,236
13	ENX	0,229	0,216	0,270	0,265	0,209	0,238
14	FRD	0,245	0,255	0,204	0,218	0,270	0,238
15	HDR	0,253	0,222	0,269	0,248	0,246	0,248

16	INT	0,220	0,266	0,209	0,226	0,246	0,233
17	IQB	0,204	0,250	0,273	0,202	0,212	0,228
18	KHR	0,230	0,260	0,205	0,259	0,244	0,240
19	MSB	0,279	0,241	0,203	0,248	0,261	0,246
20	OGY	0,278	0,268	0,237	0,249	0,253	0,257
21	PSP	0,240	0,215	0,229	0,206	0,265	0,231
22	PTR	0,240	0,214	0,251	0,210	0,207	0,224
23	RYN	0,238	0,259	0,278	0,222	0,206	0,241
24	RFD	0,225	0,248	0,245	0,258	0,251	0,245
25	RSM	0,267	0,271	0,251	0,240	0,237	0,253
26	RZT	0,216	0,237	0,213	0,274	0,258	0,240
27	SPT	0,233	0,208	0,223	0,276	0,240	0,236
28	UMR	0,232	0,251	0,206	0,265	0,263	0,243
29	YNS	0,212	0,226	0,255	0,242	0,261	0,239
30	ZKY	0,279	0,252	0,208	0,221	0,213	0,235
Rata-rata							**0,238**

Berdasarkan hasil pengujian waktu proses verifikasi tanda tangan asli pada Tabel 6.4, dari seluruh data latih tanda tangan asli yang diujikan, didapat nilai rata-rata waktu proses verifikasi sebesar 0,238 detik. Sedangkan untuk hasil pengujian waktu proses verifikasi pada data uji citra tanda tangan palsu dapat dilihat pada Tabel 6.5.

Tabel 6.5 Waktu proses verifikasi tanda tangan palsu

No.	Inisial Nama	Waktu Proses (s)					Rata-Rata (s)
1	ABD	0,263	0,219	0,248	0,247	0,245	0,244
2	ADN	0,248	0,221	0,217	0,253	0,257	0,239
3	AGN	0,271	0,247	0,210	0,266	0,225	0,244
4	ALN	0,225	0,271	0,214	0,260	0,217	0,237
5	AND	0,261	0,207	0,277	0,260	0,242	0,249
6	ANG	0,263	0,279	0,273	0,258	0,237	0,262
7	ANS	0,239	0,227	0,213	0,214	0,221	0,223
8	ARD	0,273	0,241	0,238	0,237	0,257	0,249
9	AYX	0,241	0,213	0,221	0,220	0,241	0,227
10	DYH	0,217	0,229	0,212	0,264	0,222	0,229
11	EDX	0,248	0,219	0,268	0,274	0,227	0,247
12	ELS	0,202	0,271	0,268	0,213	0,265	0,244
13	ENX	0,279	0,250	0,267	0,266	0,280	0,268
14	FRD	0,220	0,249	0,202	0,226	0,256	0,231
15	HDR	0,220	0,227	0,238	0,229	0,208	0,224
16	INT	0,245	0,216	0,270	0,206	0,232	0,234

17	IQB	0,208	0,230	0,272	0,255	0,252	0,243
18	KHR	0,252	0,260	0,211	0,267	0,278	0,254
19	MSB	0,207	0,226	0,219	0,242	0,249	0,229
20	OGY	0,280	0,216	0,266	0,266	0,238	0,253
21	PSP	0,267	0,212	0,219	0,203	0,237	0,228
22	PTR	0,255	0,275	0,269	0,233	0,234	0,253
23	RYN	0,259	0,240	0,232	0,272	0,280	0,257
24	RFD	0,223	0,206	0,261	0,247	0,223	0,232
25	RSM	0,220	0,248	0,276	0,260	0,221	0,245
26	RZT	0,243	0,255	0,256	0,234	0,266	0,251
27	SPT	0,264	0,242	0,223	0,224	0,259	0,242
28	UMR	0,264	0,250	0,251	0,265	0,273	0,261
29	YNS	0,234	0,276	0,212	0,224	0,265	0,242
30	ZKY	0,217	0,203	0,256	0,213	0,236	0,225
Rata-rata							**0,242**

Berdasarkan hasil pengujian waktu proses verifikasi tanda tangan asli pada Tabel 6.5, dari seluruh data latih tanda tangan asli yang diujikan didapat nilai rata-rata waktu proses verifikasi sebesar 0,242 detik.

6.3.5 Analisis

Berdasarkan hasil pengujian pada Tabel 6.4 dan Tabel 6.5, untuk setiap citra tanda tangan, sistem rata-rata memerlukan waktu proses 0,238 detik untuk verifikasi citra tanda tangan asli, dan 0,242 detik untuk verifikasi tanda tangan palsu. Perhitungan waktu proses dihitung mulai dari proses mengambil citra tanda tangan dengan kamera, menghitung ciri HOG citra, hingga menampilkan *output* verifikasi. Perbedaan waktu pemrosesan verifikasi citra tanda tangan pada masing-masing tanda tangan tidak mempunyai perbedaan yang signifikan.

BAB 7 PENUTUP

Bab penutup membahas tentang kesimpulan yang didapatkan serta saran yang dapat diberikan dari hasil penelitian ini. Penarikan kesimpulan mencakup mulai dari aspek perancangan, implementasi hingga hasil pengujian. Saran dari peneliti diberikan untuk pengembangan sistem ini pada penelitian di masa mendatang.

7.1 Kesimpulan

Berdasarkan rumusan masalah yang telah dibuat sebelumnya, serta hasil dari proses perancangan, implementasi, dan pengujian yang telah dilakukan, maka kesimpulan yang didapatkan yaitu:

1. Implementasi metode klasifikasi *Support Vector Machine* untuk verifikasi citra tanda tangan berdasarkan ciri *Histogram of Oriented Gradients* dapat diimplementasikan pada Raspberry Pi dengan bantuan pustaka pengolahan citra *OpenCV*. Sistem menggunakan *Pi Camera* untuk menangkap data uji citra tanda tangan dan *push button* sebagai pemicu kamera menangkap citra. Data latih disimpan dalam *Raspberry Pi* dan nantinya dimuat berdasarkan identitas nama yang dimasukkan dengan menggunakan *keyboard*. Sebelum dilakukan ekstraksi ciri fitur dengan HOG, data uji citra tanda tangan yang ditangkap kamera terlebih dahulu dilakukan preprocessing seperti merubah colorspace RGB ke *grayscale* citra, *adaptive thresholding*, *closing morphology*, dan deteksi koreksi kemiringan tanda tangan. Hasil keluaran ditampilkan pada LCD 16x2 berupa verifikasi apakah data uji termasuk kelas asli atau kelas palsu.
2. Pengujian akurasi sistem verifikasi citra tanda tangan dengan menggunakan *feature descriptor* HOG dan klasifikasi SVM dilakukan pada 300 data uji tanda tangan dari 30 penandatangan dengan masing-masing 5 tanda tangan asli dan 5 tanda tangan palsu. Pengujian ini didapatkan nilai akurasi verifikasi sebesar 87,3% untuk tanda tangan palsu dan asli. Hasil pengujian ini didapatkan dengan memakai data latih sebanyak 600 citra tanda tangan dari 30 penanda tangan dengan masing-masing penanda tangan terdapat data latih 10 tanda tangan asli dan 10 tanda tangan palsu yang dilakukan oleh penanda tangan lain.
3. Pada pengujian waktu proses pelatihan data untuk klasifikasi yang diujikan pada seluruh data latih dari 30 penanda tangan, rata-rata waktu yang dibutuhkan sistem dari proses memuat data, ekstraksi ciri citra hingga proses pelatihan data klasifikasi sebesar 1,45 detik. Untuk pengujian waktu proses verifikasi tanda tangan yang diujikan pada data uji, sistem memerlukan rata-rata waktu proses 0,238 detik untuk verifikasi citra tanda tangan asli, dan 0,242 detik untuk verifikasi tanda tangan palsu. Perhitungan waktu proses verifikasi tanda tangan dihitung mulai dari proses mengambil citra tanda tangan dengan kamera, *preprocessing*, menghitung ciri HOG citra, hingga menampilkan keluaran verifikasi

7.2 Saran

Saran yang dapat diberikan sebagai pengembangan untuk penelitian ini ke depannya sebagai berikut:

1. Masih dapat dilakukan pengembangan implementasi dari kombinasi ciri geometri lain, metode klasifikasi lain, dan metode *preprocessing* citra lain untuk meningkatkan nilai akurasi.
2. Untuk proses klasifikasi dan prediksi disarankan untuk memakai metode klasifikasi lain yang tidak memerlukan data latih tanda tangan palsu.
3. Karena objek pengujian berupa tanda tangan disarankan menggunakan kamera yang lebih tajam dalam menangkap citra objek secara *close-up* agar objek tanda-tangan pada citra tidak kabur.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR PUSTAKA

Basuki, A. & Ramadijanti, N., 2005. *Metode Numerik dan Algoritma Komputasi.* Yogyakarta: ANDI.

Chan, F. H. Y., Lam, F. K. & Zhu, H., 1998. Adaptive Thresholding by Variational Method. *IEEE Transactions on Image Processing,* Volume 7(3), pp. 468-473.

Cristianini, N. & Shawe-Taylor, J., 2000. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* s.l.:Cambridge University Press.

Dalal, N. & Triggs, B., 2005. Histograms of Oriented Gradients for Human Detection. *IEEE Computer Society.*

Dasgupta, J., Bhattacharya, K. & Chanda, B., 2016. A holistic approach for Off-line handwritten cursive word recognition using directional feature based on Arnold transform. *Pattern Recognition Letters,* Volume 79, pp. 73-79.

Fisher, R., Perkins, S., Walker, A. & Wolfart, E., 2003. *Morphology.* [Online] Available at: https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm [Diakses 15 August 2018].

Harahap, M. Y., 2005. *Hukum Acara Perdata Tentang Gugatan, Persidangan, Penyitaan, Pembuktian, Dan Putusan Pengadilan.* Jakarta: Sinar Grafika.

Harfiya, L. N., Widodo, A. W. & Wihandika, R. C., 2017. Verifikasi Citra Tanda Tangan Berdasarkan Ciri Pyramid Histogram of Oriented Gradient (PHOG) Menggunakan Metode Klasifikasi K-Nearest Neighbor. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* pp. 1162-1171.

Ilmi, R., Novianty, A. & Ahmad, U. A., 2015. Perancangan Dan Implementasi Histograms of Oriented Gradients dan Support Vector Machines (HoG+SVM) Untuk Deteksi Obyek Pejalan Kaki Pada Aplikasi Mobile Berbasis Android. *e-Proceeding of Engineering,* Volume 2, p. 3396.

IrishElectronics, 2018. *Irish Electronics.* [Online] Available at: http://irishelectronics.ie/SMD-Tact-Switch-Push-Button-6625 [Diakses 11 August 2018].

Jalil, A., 2015. Mendeteksi Keaslian Uang Kertas Rupiah Dengan Metode Thresholding Menggunakan Raspberry Pi. *Seminar Nasional Riset Ilmu Komputer,* pp. 74-82.

Kobayashi, T., Hidaka, A. & Kurita, T., 2008. *Selection of Histograms of Oriented Gradients Features for Pedestrian Detection.* Berlin Heidelberg: Springer-Verlag Berlin Heidelberg.

Malik, S., 2017. *Handwritten Digits Classification : An OpenCV (C++ / Python) Tutorial.* [Online] Available at: https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/

Miller, E., 2017. *Find the Bears: dlib.* [Online] Available at: https://hypraptive.github.io/2017/02/02/find-the-bears-dlib.html [Diakses 2 September 2018].

Nugroho, A. S., Witarto, A. B. & Handoko, D., 2003. Support Vector Machine. *Teori dan Aplikasinya dalam Bioinformatika*.

OpenCV, 2018. *OpenCV.* [Online] Available at: https://opencv.org/ [Diakses 4 August 2018].

Prakash, S., 2013. *Human Recognition using 2D and 3D Ear Images.* Kanpur: Indian Institure of Technology.

Raspberrypi.org, 2016. *Raspberry Pi 3.* [Online] Available at: https://www.raspberrypi.org [Diakses 7 August 2018].

Raspbian, 2018. *Raspbian.* [Online] Available at: https://www.raspbian.org [Diakses 19 August 2018].

Rosebrock, A., 2016. *OpenCV center of contour.* [Online] Available at: https://www.pyimagesearch.com/2016/02/01/opencv-center-of-contour/

Sembiring, K., 2007. *Tutorial SVM Bahasa Indonesia.* Bandung: Institut Teknologi Bandung.

Sidharta, H. A., 2017. *Introduction to Open CV.* [Online] Available at: http://binus.ac.id/malang/2017/10/introduction-to-open-cv/ [Diakses 12 August 2018].

Suprianto, 2015. *Pengertian Push Button Switch (Saklar Tombol Tekan).* [Online] Available at: http://blog.unnes.ac.id/antosupri/pengertian-push-button-switch-saklar-tombol-tekan/
[Diakses 18 August 2018].

Sutoyo, T. & Mulyanto, E., 2009. *Teori Pengolahan Citra Digital.* Yogyakarta: Andi.

Tomasi, C., 2012. Histograms of oriented gradients. *Computer Vision Sampler,* pp. 1-6.

Wichary, M., 2017. *International keyboard layouts in 2017.* [Online] Available at: https://medium.com/@mwichary/international-apple-keyboards-layouts-93437d7f9273
[Diakses 6 December 2018].

Widodo, A. W. & Harjoko, A., 2015. Sistem Verifikasi Tanda Tangan Off-line berdasar Ciri Histogram of Oriented Gradient (HOG) dan Histogram of Curvature (HoC). *Jurnal Teknologi Informasi dan Ilmu Komputer,* Volume 2(1), pp. 1-10.

Zhang, B., 2010. Off-line signature verification and identification by pyramid histogram of oriented gradients. *International Journal of Intelligent Computing and Cybernetics,* 3(4), pp. 611-630.