TUGAS AKHIR – TM141585

PENDETEKSIAN POSISI DAN ARAH *POINTER* GPS (*GLOBAL POSITIONING SYSTEM*) SEBAGAI REPRESENTASI DARI POSISI DAN ARAH MOBIL

BINTANG SETYA BAGUS YUDIARNO
NRP 2110100005

Dosen Pembimbing
Arif Wahjudi ST, MT, Ph.D

JURUSAN TEKNIK MESIN
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2016

FINAL PROJECT – TM141585

DETECTION OF GOOGLE MAPS' GPS (GLOBAL POSITIONING SYSTEM) POINTER'S POSITION AND DIRECTION AS A REPRESENTATION OF AUTONOMOUS CAR'S POSITION AND DIRECTION

BINTANG SETYA BAGUS YUDIARNO
NRP 2110100005

Advisor
Arif Wahjudi ST, MT, Ph.D

MECHANICAL ENGINEERING
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

LEMBAR PENGESAHAN

PENDETEKSIAN POSISI DAN ARAH POINTER GPS (*GLOBAL POSITIONING SYSTEM*) SEBAGAI REPRESENTASI DARI POSISI DAN ARAH MOBIL

TUGAS AKHIR
Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
pada
Program Studi S-1 Jurusan Teknik Mesin
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember

Oleh:
BINTANG SETYA BAGUS Y
Nrp. 2110 100 005

Disetujui oleh Tim Penguji Tugas Akhir :
1. Arif Wahjudi, S.T, M.T, Ph.D. .....(Pembimbing)
   NIP. 197303222001121001
2. Prof. Dr. Ing. Ir. I Made Londen Batan, M.Eng
   .....(Penguji I)
   NIP. 195811061986011001
3. Ir. Bambang Pramujati, M.Sc., Eng, Ph.D. .....(Penguji II)
   NIP. 196912031994031001
4. Dinny Harnany, S.T, M.Sc. .....(Penguji III)
   NIP. 2100201405001

SURABAYA
Juli 2016

PENDETEKSIAN POSISI DAN ARAH POINTER GPS (*GLOBAL POSITIONING SYSTEM*) GOOGLE MAPS SEBAGAI REPRESENTASI DARI POSISI DAN ARAH MOBIL

Nama : **Bintang Setya Bagus Y**
NRP : **2110100005**
Jurusan : **Teknik Mesin FTI-ITS**
Dosen Pembimbing : **Arif Wahjudi ST,MT,Ph.D**

ABSTRAK

Otomotif, sarana bidang transportasi yang berfungsi dalam upaya memenuhi tuntutan ekonomi dimana tingkat kebutuhan mobilitas dan waktu dari masyarakat dituntut agar mencapai nilai efisiensi dan efektifitas kehidupan. Mobil, salah satu jenis kendaraan bermotor telah mengalami perkembangan yang sangat pesat. Teknologi mobil yang sedang dikembangkan saat ini adalah *self driving car*, Dimana mobil dapat berjalan dan mengemudi dengan sendirinya ke tempat tujuan yang telah ditetapkan oleh navigasi pada maps. Salah satu *self driving car* yang telah dikembangkan adalah Google *smart car*, dimana mobil ini menggunakan berbagai macam sensor untuk berjalan sesuai navigasi. Namun harga dari sensor tersebut cukup mahal. Maka dari itu dikembangkan metode baru yaitu hanya menggunakan kamera untuk menjalankan mobil sesuai dengan navigasi yang telah ditentukan oleh maps pada *smartphone.*

Pada tugas akhir ini, dilakukan perancangan dan pembangunan proram untuk mendeteksi pointer Google Maps dan pencarian arah posisi pointer Google Maps. Pendeteksian pointer *Google Maps* digunakan untuk mengetahui posisi mobil dalam keadaan sebenarnya. Sedangkan pencarian nilai sudut yang dibentuk oleh pointer Google Maps digunakan untuk mengetahui arah mobil dalam keadaan sebenarnya. Dalam proses pendeteksian, segmentasi citra terlebih dahulu dilakukan agar pointer Google Maps terpisah dari *backgroundnya*, hingga pointer

Google Maps terlihat secara jelas. Setelah proses segmentasi bentuk dari pointer Google Maps dapat dideteksi, sehingga pointer Google Maps dapat dikenali oleh kamera. Langkah berikutnya adalah mencari nilai sudut yang dibentuk oleh pointer Google Maps. Untuk itu diperlukan 2 garis sebagai representasi sudut bentukan pointer Google Maps. Garis yang pertama adalah garis yang menghubungkan titik ujung pointer dengan titik tengahnya. Garis yang kedua adalah garis lurus vertikal dengan titik awal yaitu titik tengah pointer. Setelah mendapatkan kedua garis tersebut dapat dihitung sudut yang dibentuk oleh pointer Google Maps.

Hasil yang diperoleh pada tugas akhir ini adalah kemampuan program pendeteksian secara *real-time* dengan menggunakan fungsi OpenCV "approxPolyDP" mencapai 98,45%. Nilai sudut hasil keluaran dari *image processing* kemudian di bandingkan dengan kompas digital menggunakan metode *paired sample t-test*. Dimana uji hipotesanya adalah, H_0: $\delta = \delta_0$, H_1 : $\delta \neq \delta_0$. dimana δ adalah perbedaan rata-rata antara kedua *samples*. Dari hasil perhitugan didapat nilai $t_{perhitungan}$ = 1,205 sedangkan dari tabel distribusi t didapat nilai t_{tabel} = 2,045 untuk $\propto = 0.05$. Karena nilai 1,205 < 2.045 maka nilainya berada pada daerah penerimaan, sehingga H_0 diterima yang menunjukkan selisih nilai sudut *image processing* dengan kompas adalah 0.

Kata kunci : *image processing*, OpenCV, *Object detection, Get pixel value*.

DETECTION OF GOOGLE MAPS' GPS (GLOBAL POSITIONING SYSTEM) POINTER'S POSITION AND DIRECTION AS A REPRESENTATION OF AUTONOMOUS CAR'S POSITION AND DIRECTION

Name : **Bintang Setya Bagus Y**
NRP : **2110100005**
Department : **Mechanical Engineering FTI-ITS**
Advisor : **Arif Wahjudi ST,MT,Ph.D**

ABSTRACT

Automotive, a transportation facility, used to fulfill economical demands which mobility and time necessity level from the society demanded to reached life's efficiency and effectivity. Cars, as one of vehicles' type, has made a rapid development. Car technology that is being developed recently is a self-driving car, which can drive itself to the designated destination by the car's map navigation. One of the self-driving car that has been developed is Google's smart car, which use a various types of sensors and radars to drive itself according to the navigation. However, the price of these sensors are fairly expensive. Hence, new method using just camera to drive the car according to the map's navigation in a smartphone is developed.

In this last project, program construction is made to detect Google Maps' pointer and search the Google Maps' position and direction. Google Maps' pointer detection is used for knowing the position of the car in real-time. Whilst the angle value detection from the Google Maps' pointer is used to know about the car's direction in real-time. In detection process, image segmentation was made in advance to separate the Google Maps' pointer from the background so that the Google Maps' pointer looked more distinct. After the segmentation process, the shape of the Google Maps' pointer can be detected, so the Google Maps' pointer recognisable by the camera. The next step was searching for the angle value made by the Google Maps' pointer. For this process,

2 lines was needed for angle representatives made by Google Maps' pointer. The first line was made by connecting pointer's front edge with its centroid. The second line is a vertical line, made through the pointer's centroid. After making both lines, then the angle value can be calculated.

The results achieved from this last project are the program's detection capability in real-time, using OpenCV function "approxPolyDP", reaching 98.45%. The angle value, resulting from image processing, then compared to a digital compass using paired sample t-test method. This method uses hypothesis tests which are, $H_0 : \delta = \delta_0$, $H_1 : \delta \neq \delta_0$, whereas δ is the mean difference between both samples. From the calculation result, $t_{calculation} = 1.205$ was obtained whilst $t_{table} = 2.045$ was obtained from t distribution table for $\alpha = 0.05$. Since $1.205 < 2.045$ then the value is inside the acceptable area, which made the hypothesis H_0, the difference between image processing angle value and digital compass angle value is 0, acceptable.

***Keyword* : *image processing*, OpenCV, *Object detection, Get pixel value*.**

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman sengaja dikosongkan*]

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi yang sangat pesat di era globalisasi saat ini telah memberikan banyak manfaat dalam kehidupan manusia. Semakin banyak inovasi-inovasi baru yang diciptakan dengan basis teknologi canggih, khususnya dalam bidang otomotif. Teknologi otomotif yang terbaru adalah *self driving car* dimana mobil dapat berjalan tanpa dikemudikan. Contoh nyata dari *self driving car* yang sudah ada salah satunya adalah *google self driving car*. Sistem kendali otomatis dari *google self driving car* ini didukung oleh beberapa macam peralatan seperti kamera, radar, sensor, GPS *receiver*, HMI(*human-machine interface*), *domain controller*, serta *mechatronic units*. Peralatan-peralatan sensor inilah yang menyebabkan teknologi *self driving car* menjadi mahal.

Di jaman yang modern ini hampir semua orang menggunakan *smartphone*. Salah satu fasilitas yang tersedia pada *smartphone* adalah aplikasi peta/*maps* yang memanfaatkan sistem GPS dan *satellite*. Aplikasi *maps* yang populer dan sering digunakan oleh banyak orang yaitu Google Maps. Google Maps merupakan salah satu aplikasi yang digunakan untuk mengetahui lokasi pengguna, jalan di sekitar pengguna, maupun tempat-tempat yang berada di sekitar, seperti *restaurant*, masjid, ATM, dan lain-lain. Dalam menggunakan aplikasi Google Maps, terdapat *pointer* yang berbentuk menyerupai segitiga berwarna biru, dimana *pointer* tersebut menunjukkan lokasi dari pengguna. Sedangkan untuk jalan yang akan dituju ditandai dengan warna biru pula. Untuk jalan yang lain (jalan yang tidak dituju) akan ditandai dengan warna putih. Aplikasi *Google Maps* dapat dimanfaatkan citranya untuk membuat sistem baru dalam mengendalikan *prototype self driving car.*

Pengolahan citra (*image processing*) adalah teknik mengolah citra yang mentransformasikan citra masukan menjadi

citra lain agar keluarannya memiliki kualitas yang lebih baik dibandingkan kualitas citra masukan. Pengolahan citra memiliki banyak manfaat, diantaranya adalah untuk meningkatkan kualitas citra, menghilangkan cacat pada citra, mengidentifikasi objek dan mengolah informasi yang berada pada citra. Dengan memanfaatkan teknologi tersebut, maka diharapkan adanya suatu aplikasi yang dapat menangkap suatu objek yang ada di depan kamera. Lalu mengidentifikasi dan mendeteksi objek, serta melakukan *tracking* objek secara *real-time*.

Deteksi dan pengenalan *pointer* Google Maps dengan pengolahan citra dapat dilakukan dengan *input* gambar maupun video. Pengenalan *pointer* Google Maps dengan objek deteksi berupa gambar merupakan tahap pengenalan *pointer* Google Maps tanpa *tracking*. Sedangkan pengenalan *pointer* dengan *input* citra *real time*, merupakan tahap pengenalan *pointer* Google Maps dengan *tracking*. Aplikasi pengenalan *pointer* Google Maps ini terdapat tiga proses, yaitu *pre-processing, processing* dan *post-processing*.

Dalam tugas akhir ini, objek berupa *pointer* Google Maps direncanakan akan diidentifikasi dengan menggunakan kamera tunggal secara *real-time* berdasarkan bentuk dan warna RGB (*red green blue*). Identifikasi dilakukan untuk menentukan posisi dan arah *pointer* sebagai representasi dari lokasi dan posisi arah mobil. Hasil dari pendeteksian tersebut kemudian diolah dan dianalisa apakah program tersebut bisa berjalan sesuai yang diharapkan.

1.2 Perumusan Masalah

Dalam penyusunan proposal tugas akhir ini, rumusan masalah yang akan dibahas adalah bagaimana merancang program untuk mendeteksi *pointer* beserta arah *pointer* GPS Google Maps secara *real time*?

1.3 Batasan Masalah

Untuk menyederhanakan tugas akhir ini, diberi batasan masalah sebagai berikut:

1. Google Maps dengan pengaturan arah utara berubah.
2. Penggunaan Google Maps hanya untuk *daytime*.
3. Citra dibatasi pada bentuk *pointer* Google Maps.
4. Cahaya hanya berasal dari layar *smartphone*.

1.4 Tujuan Penelitian

Tujuan dari proposal tugas akhir ini adalah Merancang dan membangun program untuk mendeteksi dan mengetahui arah *pointer* Google Maps secara *real time.*

1.5 Manfaat Penelitian

Manfaat dari proposal tugas akhir ini adalah sebagai berikut:

1. Untuk mengetahui tingkat pendeteksian, akurasi dan efisiensi, serta pemilihan parameter yang baik dalam mendeteksi *pointer* Google Maps.
2. Untuk membuat sistem yang bisa mengetahui posisi/arah pointer Google Maps dalam 2D atau bidang kartesian xy sehingga dapat diaplikasikan pada *prototype self driving car*.
3. Dasar dari aplikasi untuk membuat sistem yang sesuai dengan posisi dan arah dari *pointer* Google Maps.

[*Halaman sengaja dikosongkan*]

BAB II
DASAR TEORI

2.1 *Self Driving Car*

Perkembangan teknologi dalam bidang otomotif, khususnya dalam bidang mobil sangat pesat sekali. Pada awalnya mobil sepenuhnya di kontrol oleh manusia, namun pada saat ini, mobil mulai dikembangkan untuk dapat berjalan sendiri tanpa dikemudikan oleh pengemudi. Kedepannya mobil akan dikembangkan lagi agar tidak perlu pengemudi di dalamnya, sehingga mobil dapat berjalan dengan sendirinya. Teknologi yang berada pada sistem mobil digunakan untuk memudahkan pengemudi dalam mengendarai mobil. Contoh teknologi tersebut seperti *crash warning system, adaptive cruise control, lane keeping systems dan self-parking technology.*

National Highway Traffic Safety Administration (NHTSA) mengklasifikasikan perkembangan teknologi mobil dalam beberapa level [1]:

- *Level 0 (no automation*): Pengemudi mengendalikan semua fungsi dari mobil, seperti rem, gas, setir, dan lain-lain. Pengemudi juga yang menjalankan fungsi monitoring terhadap jalan dan keadaan sekitar, dan atas keselamatan pengemudi sendiri.
- *Level 1* (*function-specific automation*): Pengemudi tetap memiliki kontrol secara keseluruhan dan atas keselamatan pengemudi sendiri. Namun terdapat beberapa fungsi otomatis pada mobil yang digunakan apabila mobil dalam keadaan tidak normal/darurat. Seperti sistem *automatic braking* yang berfungsi ketika mobil terlalu dekat dengan objek di depan yang dapat menimbulkan potensi kecelakaan.
- *Level 2 (combined-function automation*): Sistem ini melibatan setidaknya 2 kontrol otomatis mobil yang dirancang untuk bekerja sama untuk membantu pengemudi dalam mengendalikan mobil. Seperti

sistem *adaptive cruise control* dan *system lane keeping* yang dikombinasikan agar pengemudi mengendalikan mobil dengan lebih aman dan nyaman.

- *Level 3* (*limited self-driving automation*): Sistem otomatis ini memampukan mobil agar dapat berjalan dan mengemudikan sendiri tanpa perlu pengemudi mengendalikan setir, gas, rem, dan fungsi mobil lainnya.Namun sistem otomatis ini hanya bekerja pada kondisi *traffic* tertentu. Apabila dalam keadaan darurat atau kondisi *traffic* tidak bisa di deteksi dengan system otomatis ini, maka fungsi mobil seperti setir, gas, rem, dan lain-lain dikembalikan ke pengemudi.
- *Level 4* (*Full Self Driving Automation*): Mobil sepenuhnya dapat dikendalikan oleh sistem kendali otomatis. Sistem otomatis ini dapat mengantisipasi *traffic* pada saat perjalanan dan menentukan navigasi secara otomatis agar sampai ke tempat tujuan. Pengemudi tidak perlu lagi mengendalikan mobil sepanjang perjalanan.

Self driving car merupakan mobil yang diciptakan untuk dapat berjalan sendiri dari tempat awal ke tempat yang dituju tanpa dikendalikan oleh manusia. Untuk itu mobil harus dapat menentukan navigasi ke tempat tujuan dengan otomatis. Keuntungan menggunakan teknologi *driverless car* diantaranya dapat mengurangi angka kecelakaan. Hal ini dikarenakan sistem otomatis mempunyai respon yang lebih cepat daripada manusia. Selain itu sepanjang perjalanan, pengemudi dapat melakukan aktivitas lain seperti membaca buku, menonton film dan lain-lain.

Berbagai macam sensor digunakan agar *self driving car* dapat berjalan tanpa dikemudikan. Fungsi lain dari sensor tersebut juga digunakan untuk mendeteksi dan melihat area sekitar *self driving car* agar mobil tersebut dapat berjalan dengan aman. Namun sensor yang digunakan dan program yang digunakan masih terus dikaji agar penumpang merasa nyaman dan tidak

takut akan terjadi kecelakaan akibat dari kesalahan pembacaan sensor dan program yang *error*

Gambar 2.1 *Google Self-Driving Car* [10]

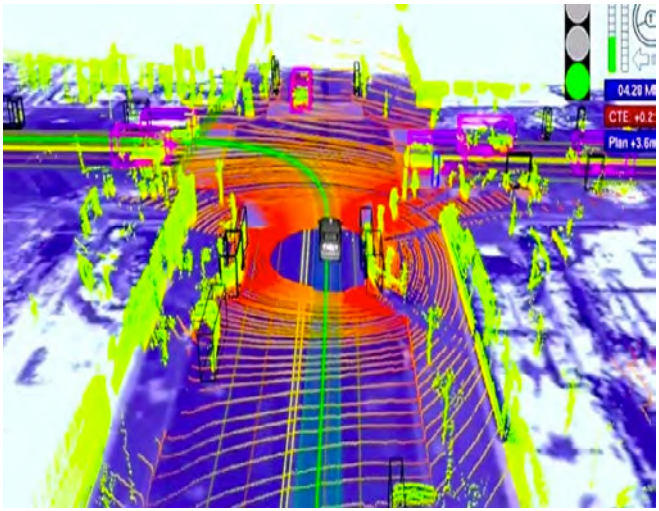
Gambar 2.1 menunjukkan mobil *self driving car* yang telah dikembangkan oleh *google*. Mobil *google* pertama kali diuji pada tahun 2009 di jalan bebas hambatan California. Setelah sukses pada pengujian di jalan bebas hambatan, pengujian dilakukan di lingkungan yang lebih kompleks, seperti jalan di perkotaan. Terdapat berbagai macam sensor yang digunakan agar mobil tersebut dapat mengemudi secara otomatis, yaitu [17]:

- *LIDAR (Light Sensing Radar):* berupa kamera yang dapat berputar 360°, terletak pada bagian atap mobil. Kamera ini dapat menangkap objek 3D di sekitar mobil untuk membantu mobil melihat adanya objek atau rintangan di sepanjang jalan. Alat ini dapat menghitung jarak antara objek dengan mobil berdasarkan waktu tempuh gelombang laser mengenai objek dan kembali

ke mobil. Alat ini dapat mendeteksi objek dengan rentang radius 200 meter.

- *Front Camera*: merupakan kamera yang digunakan untuk pandangan jarak dekat. Kamera ini digunakan untuk mendeteksi adanya pejalan kaki atau pengendara motor yang lewat di depan mobil. Selain itu, berfungsi juga untuk merekam informasi mengenai tanda jalan dan lampu lalu lintas yang kemudian diterjemahkan oleh *software* mobil.
- *Bumper Mounted Radar*: radar yang dipasang pada *bumper* depan dan belakang mobil ini berfungsi untuk mendeteksi adanya mobil di bagian depan dan belakang mobil. Dengan radar ini kecepatan mobil dapat ditentukan berdasarkan jarak dengan mobil di depan. Ketika tidak terdeteksi adanya mobil di depan maka mobil *google* ini akan berjalan semakin cepat, sedangkan apabila semakin dekat dengan mobil di depan maka mobil *google* akan memperlambat jalannya. Jarak yang dijaga dengan mobil di depan adalah kisaran 2-4 meter. Radar ini juga mencegah terjadinya tabrakan apabila ada pengendara motor atau pejalan kaki yang lewat di depan mobil.
- *Aerial*: alat ini digunakan untuk membaca informasi mengenai lokasi mobil dan mengurangi ketidakpastian penentuan lokasi mobil pada GPS yang dapat disebabkan oleh gangguan sinyal, maka alat ini berfungsi untuk membandingkan data GPS dengan data peta yang telah diambil sebelumnya dengan menggunakan sensor sehingga dapat meng-*update* lokasi mobil yang sebenarnya pada GPS.
- *Ultrasonic sensor*: *ultrasonic* dipasang pada roda mobil bagian belakang. Alat ini berfungsi untuk membantu mobil mendeteksi dan memperingatkan mobil ketika ada objek yang mendekat ke arah mobil.

- *Devices within the car*: peralatan yang berada di dalam mobil berupa *altimeter, accelerometer, dan tachymeter* dapat menentukan posisi mobil dengan akurat karena banyaknya parameter yang diukur. Hal ini memberikan data yang sangat akurat sehingga mobil dapat beroperasi dengan aman.

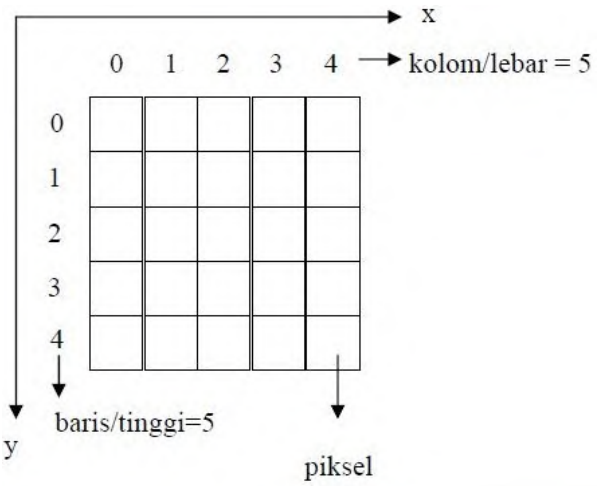


Gambar 2.2 Pembacaan Sensor *Google Self Driving Car* [11]

Gambar 2.2 menunjukkan hasil pembacaan sensor *google self-driving car* terhadap lingkungan di sekitarnya. Garis-garis lingkaran yang berwarna merah-oranye merupakan hasil pembacaan sensor LIDAR. Garis hijau merupakan lintasan yang harus dilalui mobil *google*. Warna hijau muda menunjukkan objek-objek yang ada di sekitar mobil *google*. Pada pojok kanan atas terdapat indicator dari lampu lalu-lintas yang didapat dari hasil pembacaan sensor kamera.

2.2 *Image Processing*/Pengolahan Citra Digital

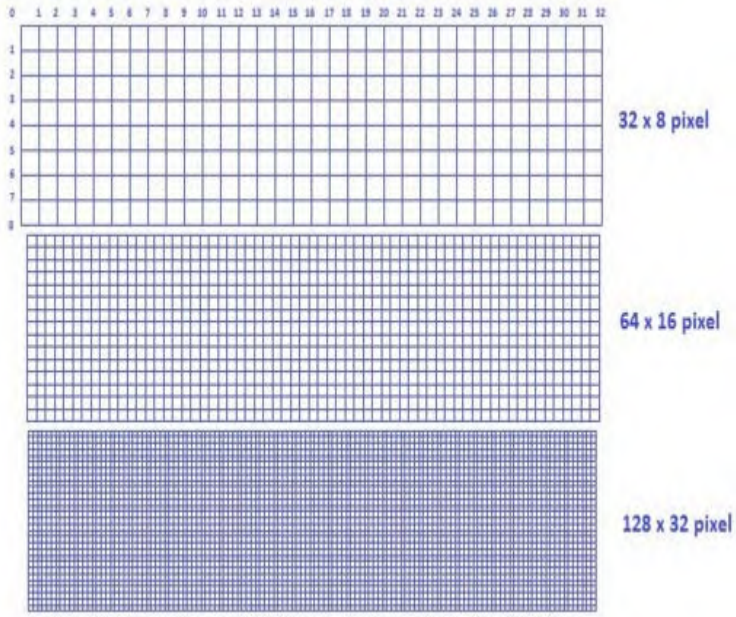
Image processing adalah suatu metode yang digunakan untuk melakukan operasi pada citra dengan tujuan untuk mengolah citra agar lebih sempurna dan mengambil beberapa informasi dari citra tersebut. Input dari pemrosesan ini berupa citra dan output dapat berupa citra atau informasi yang didapat dari input. Pengolahan citra pada dasarnya mencakup mengimpor citra, menganalisis serta memanipulasi citra, dan hasil output yang berbeda setelah dilakukan analisis pada citra. *Image processing* mempunyai banyak fungsi seperti: penajaman gambar, pendeteksian objek pada gambar, pengurangan *noise*, konversi gambar berwarna *grayscale* maupun sebaliknya dan lain-lain. Citra digital dapat didefinisikan sebagai gambar (*image*) dalam tampilan 2 dimensi yang disimpan dalam bentuk/format digital [2].



Gambar 2.3 Elemen Citra Digital [12]

Gambar 2.3 citra merupakan fungsi dua dimensi, f(x,y), dimana x dan y titik koordinat dan amplitudo f pada setiap pasang koordinat (x,y) disebut intensitas atau tingkat keabuan. Piksel adalah istilah yang digunakan untuk menunjukkan elemen citra

digital. Setiap piksel memiliki informasi berupa kandungan warna *red, green, blue* (RGB).

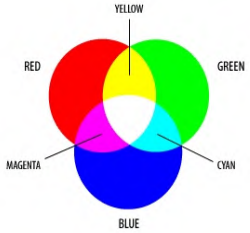


Gambar 2.4 Contoh Ukuran Piksel [18]

Gambar 2.4 menunjukkan contoh perbedaan ukuran piksel mulai dari 32 x 8 piksel, 64 x 16 piksel dan 128 x 32 piksel. Semakin besar ukuran piksel maka citra yang ditampilkan semakin besar, dan apabila citra diperbesar masih tampak dengan jelas. Sedangkan citra dengan ukuran piksel kecil, citra yang ditampilkan juga kecil dan apabila diperbesar citra tampak blur. Semakin banyak jumlah piksel dan semakin rapatnya piksel dalam citra, maka semakin bagus kualitas gambar dan dapat memberikan informasi yang lebih jelas.

2.2.1 Citra Warna RGB

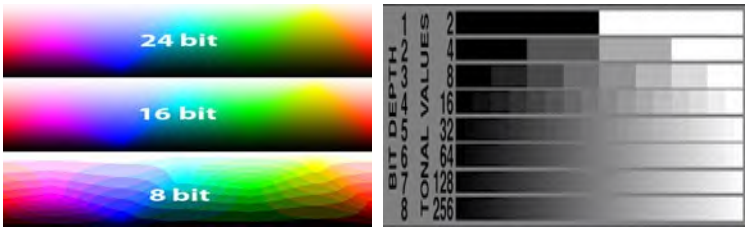
Citra pada setiap pikselnya mengandung unsur warna RGB (*Red Green Blue*). Dengan gabungan ketiga unsur tersebut, dapat menjadikan warna lain yang tampak pada citra [3]



Warna	R	G	B
Hitam	0	0	0
Merah	255	0	0
Hijau	0	255	0
Biru	0	0	255
Kuning	255	255	0
Magenta	255	0	255
Cyan	0	255	255
Putih	255	255	255
Abu-Abu	127	127	127
Orange	255	110	0
Ungu	128	0	255
Coklat	128	25	0
Pink	255	190	220
Navy	0	0	120

Gambar 2.5 Warna RGB dan Kombinasi RGB [13]

Gambar 2.5 menunjukkan citra berwarna direpresentasikan dalam beberapa *channel* yang menyatakan warna komponen-komponen penyusunnya. Citra warna memiliki satu set nilai tersusun yang menyatakan berbagai tingkat warna. Setiap piksel pada citra warna mewakili



Gambar 2.6 Bit Dalam Warna [14]

Gambar 2.6 menunjukkan perbandingan warna yang ada dalam gambar yang sama tapi dengan nilai *bit* yang berbeda.

Nilai *tonal values* didapatkan dari 2^n, dimana n merupakan nilai *bit*. Semakin besar nilai *tonal values* maka jumlah warna yang dihasilkan semakin banyak sehingga perubahan warna yang ada dalam citra semakin halus.

Citra warna (8 *bit*) adalah citra yang setiap pikselnya diwakili oleh 8 *bit*. Jumlah warna maksimum yang dapat digunakan adalah 256 warna. Setiap titik (piksel) pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar yaitu merah, hijau, dan biru yang biasa disebut citra RGB.

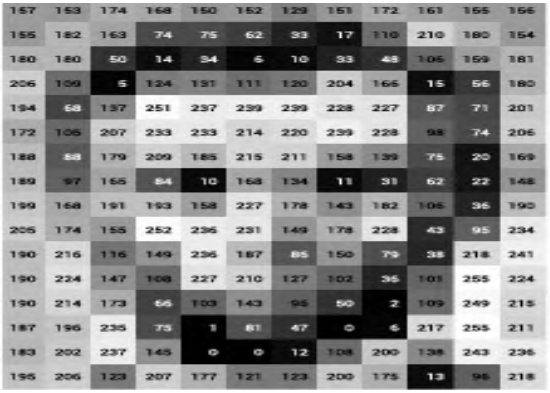
Citra warna (16 *bit*) adalah citra yang setiap pikselnya diwakili dengan 16 *bit*. Jumlah warna maksimum yang dapat digunakan adalah 65.536 warna. Setiap titik (piksel) pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar yaitu merah, hijau, dan biru yang biasa disebut citra RGB.

Citra warna (24 *bit*) adalah citra dengan setiap pikselnya diwakili dengan 24 *bit*. Jumlah warna maksimum yang dapat digunakan adalah 16.777.216 variasi warna. Variasi tersebut dapat memvisualisasikan seluruh warna yang dapat dilihat oleh penglihatan manusia. Setiap poin informasi piksel (RGB) disimpan ke dalam 1 *byte* data dengan 8 *bit* pertama menyimpan nilai biru, kemudian diikuti dengan nilai hijau pada 8 *bit* kedua dan pada 8 *bit* terakhir merupakan warna merah[3].

2.2.2 Citra *Grayscale*

Citra *grayscale* merupakan citra digital yang hanya memiliki satu nilai *channel* pada setiap pikselnya. Artinya nilai dari *Red = Green = Blue*, dimana nilai-nilai tersebut digunakan untuk menunjukkan intensitas warna pada setiap pikselnya. Maka konversi dapat dilakukan dengan mengambil rata-rata dari nilai R,G dan B [4]. Sehingga persamaan untuk mencari nilai *grayscale* pada setiap pikselnya:

$$Gray\ (i,j) = [\ \frac{(f\ red(i,j)+f\ green\ (i,j)+f\ blue\ (i,j))}{3}\ ] \quad \text{..... (2.1)}$$



Gambar 2.7 Nilai Citra *Grayscale* Pada Setiap Piksel [15]

Gambar 2.7 menunjukkan gambar dengan ukuran 12 x 16 piksel, dimana setiap pikselnya tertera nilai grayscale. Citra yang ditampilkan dari citra jenis ini terdiri atas variasi warna abu-abu, dimana warna hitam merupakan warna abu-abu dengan intensitas terendah (0) dan warna putih memiliki tingkat intensitas tertinggi (255). Citra *grayscale* berbeda dengan citra hitam putih, dimana citra hanya terdiri atas 2 warna, yaitu hitam dan putih saja. Pada citra *grayscale*, warna bervariasi antara hitam dan putih, tetapi variasi warna diantaranya sangat banyak. Citra *grayscale* disimpan dalam format 8 *bit* untuk setiap sample piksel, yang memungkinkan adanya 256 tingkat intensitas.

2.2.3 Citra *Threshold*

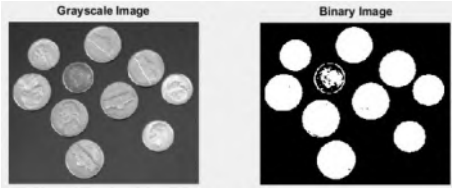
Thresholding adalah proses mengubah citra berderajat keabuan menjadi citra biner atau hitam putih sehingga dapat diketahui daerah mana yang termasuk objek dan *background* dari citra secara jelas. Citra hasil *thresholding* digunakan lebih lanjut untuk proses pengenalan objek serta ekstrasi fitur. Pada

aplikasinya sebelum citra dilakukan proses *thresholding*, citra dijadikan bentuk *grayscale* yang memiliki nilai piksel keabuan 0-255. Berikut adalah persamaan dalam proses *thresholding*:

$$f_0(x,y) = \begin{cases} 0, & f_1(x,y) < T \\ 255, & f_1(x,y) \geq T \end{cases} \quad \text{.....(2.2)}$$

Dimana:

$f_0(x,y)$: Nilai citra hasil *threshold*
$f_1(x,y)$: Nilai pixel keabuan
T : Nilai pemetaan piksel



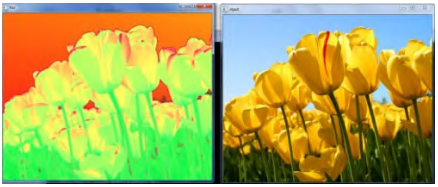
Gambar 2.8 Citra *grayscale* dan citra *threshold* [18]

Gambar 2.8 merupakan citra *grayscale* yang kemudian dilakukan proses *threshold* sehingga citra menjadi biner yaitu hitam dan putih. Nilai piksel *grayscale* dikelompokkan menjadi 2 kelompok, apabila nilai f_1 lebih kecil daripada nilai T, maka nilai piksel menjadi 0, yaitu warna hitam. Sebaliknya apabila nilai f_1 lebih besar daripada nilai T, maka nilai piksel menjadi 1, yaitu warna putih. Pada citra biner, batas antara objek dengan latar belakang terlihat jelas. Piksel objek berwarna putih, sedangkan piksel latar belakang berwarna hitam, ataupun sebaliknya.

2.2.4 HSV (*Hue Saturation Value*)

Selain model warna RGB terdapat juga model warna HSV dimana model ini terdapat 3 komponen yaitu *hue*, *saturation* dan *value*. *Hue* adalah sudut dari 0 sampai 360 derajat, biasanya 0 adalah merah, 60 derajat adalah kuning,

120 derajat adalah hijau, 180 derajat adalah *cyan*, 240 derajat adalah biru dan 300 derajat adalah warna *magenta*.



Gambar 2.9 Warna RGB dan HSV [19]

Gambar 2.9 menunjukkan perbedaan model warna RGB ke HSV. Model warna HSV lebih banyak digunakan untuk *image processing.* Berikut adalah persamaan untuk mengkonversi nilai piksel RGB ke nilai piksel HSV[4]:

$$V = \frac{1}{3}(R+G+B) \quad (2.3)$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)] \quad (2.4)$$

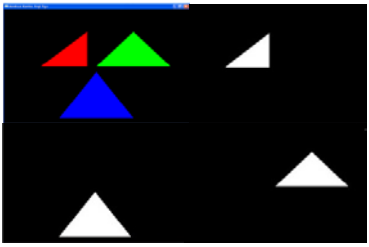
$$H = \begin{cases} \theta, & B \le G \\ 360 - \theta, & B > G \end{cases} \quad (2.5)$$

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\} \quad (2.6)$$

Hue menunjukkan jenis warna (seperti merah, biru atau kuning). *Saturation* sering dikenal dengan *chroma*, yaitu ukuran banyaknya warna putih yang bercampur pada hue. *Value* atau disebut juga intensitas yaitu ukuran seberapa besar kecerahan dari suatu warna atau seberapa besar cahaya datang dari suatu warna. *Saturation* dan *Value* ukurannya dalam bentuk persentase dari 0 sampai 100%. Suatu warna dengan nilai *value* 100% akan tampak paling cerah dan suatu warna dengan nilai *value* 0 akan tampak paling gelap[8].

2.2.5 Segmentasi Citra Dengan Model Warna HSV

Segementasi citra akan membagi-bagi suatu citra menjadi daerah-daerah atau objek-objek yang dimilikinya. Segmentasi citra merupakan suatu proses memecah suatu citra digital menjadi beberapa segmen/bagian agar obyek dapat dipisahkan dari *background*-nya [3]. Pada metode segmentasi citra dengan deteksi warna HSV, dilakukan pemilihan sampel piksel sebagai acuan warna untuk membentuk segmen yang diinginkan. Citra digital menggunakan model warna RGB sebagai standar acuan warna, oleh karena itu proses awal pada metode ini memerlukan konversi model warna RGB ke HSV [5].



Gambar 2.10 Segmentasi berdasarkan Warna

Gambar 2.10 menunjukkan hasil segmentasi warna merah, hijau, dan biru pada gambar yang sama. Untuk membentuk segmen sesuai dengan warna yang diinginkan maka ditentukan nilai toleransi pada setiap dimensi warna HSV, kemudian nilai toleransi tersebut digunakan dalam perhitungan proses *threshold*. Hasil dari proses *threshold* tersebut akan membentuk segmen area dengan warna sesuai toleransi yang diinginkan. Berikut ini merupakan proses segmentasi dengan HSV:

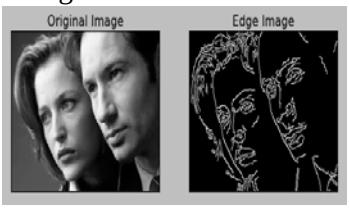
- Tentukan citra RGB, tentukan warna yang akan di segmentasi
- Transpose citra RGB ke HSV

- Lakukan filter warna pada citra berdasarkan nilai acuan (T) dan nilai toleransi (tol). Dengan x sebagai warna HSV pada piksel yang ada, maka warna yang tidak termasuk dalam rentang T-tol < x < T+tol diberi warna hitam.
- Transpose kembali citra ke RGB, tampilkan hasil filter.

2.2.6 *Edge Detection*

Edge detection (deteksi tepi) pada suatu citra adalah operasi yang dijalankan untuk mendeteksi garis tepi (*edges*) yang membatasi dua wilayah citra homogen yang memiliki tingkat kecerahan yang berbeda. Secara umum, *edge detection* dalam citra dinyatakan sebagai titik yang nilai warnanya berbeda cukup besar dengan titik yang ada di sebelahnya. Tujuan dari *edge detection* ini adalah[7]:

- Meningkatkan penampakan garis batas suatu daerah atau objek di dalam citra.
- Menandai bagian yang menjadi detail citra.
- Mencirikan batas objek dan berguna untuk proses segmentasi dan identifikasi objek.
- Memperbaiki detail citra yang kabur.
- Melihat apakah bentuk dari suatu objek antar tepinya tersambung atau tidak



Gambar 2.11 *Canny Edge Detection* [20]

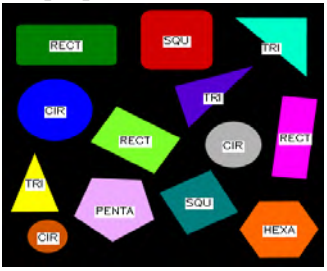
Gambar 2.11 merupakan hasil deteksi *edge* dari sebuah gambar yang memiliki beberapa objek. Algoritma *canny edge detection* merupakan salah satu teknik *edge detection* yang

cukup popular penggunaanya dalam pengolahan citra. Salah satu alasannya adalah ketebalan *edge* yang bernilai satu piksel yang dimaksudkan untuk melokalisasi posisi *edge* pada citra secara sepresisi mungkin. Algoritma *canny edge detection* secara umum beroperasi sebagai berikut:

- Penghalusan/*smoothing* untuk mengurangi dampak *noise* terhadap pendeteksian *edge.*
- Menghitung potensi gradien citra.
- *Non-maximal suppression* dari gradien citra untuk melokalisasi *edge* secara presisi.
- Histerisis *thresholding* untuk melakukan klasifikasi akhir.

2.2.7 *Shape Detection*

Metode ini digunakan untuk mendeteksi bentuk objek dalam citra. Objek yang dimaksud adalah berbentuk segitiga, segiempat, segilima, hingga objek berbentuk lingkaran. Pendeteksian bentuk berbeda dengan segmentasi warna dan pemisahan bentuk objek dengan *background* citra. Pendeteksian digunakan untuk memberikan informasi kepada sistem kamera bahwa bentuk tersebut adalah segitiga atau segi-empat, segi-lima atau lingkaran dan lain-lain. Sehingga sistem kamera dapat mengerti bentuk objek, sama seperti pemikiran manusia.[23]



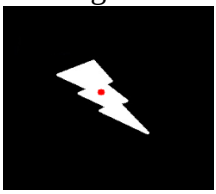
Gambar 2.12 Pendeteksian bentuk objek citra [21]

Gambar 2.12 merupakan hasil citra yang telah dilakukan proses pendeteksian. Dalam pendeteksian bentuk, citra terlebih

dahulu dilakukan proses segmentasi, agar mendapat gambaran bentuk objek yang akan di deteksi. Pencarian kontur berperan besar dalam mendeteksi bentuk. *Noise* diminimalisasi agar tidak salah dalam pendeteksian bentuk. Pendeteksian ini memanfaatkan jumlah simpul dan nilai dari sudut bentuk objek, apabila jumlah simpul 3, maka dideteksi sebagai bentuk segitiga. Jumlah simpul 4 maka objek di deteksi sebagai persegi dan seterusnya. Nilai sudut dapat divariasikan agar bentuk dapat dideteksi secara akurat.

2.2.8 *Image Moment*

Image moment digunakan untuk mengetahui posisi dari *centre of mass* objek. Properti-properti dari gambar yang dapat ditentukan melalui *image moment* antara lain luas objek atau total intensitas warna, *centroid* dan informasi mengenai orientasi gambar.



Gambar 2.13 *Centre of mass* objek [22]

Gambar 2.13 menunjukkan *centre of mass* dari sebuah objek. Untuk menentukan koordinat dari *centre of mass* sebuah objek yang telah di segmentasi dapat dicari dengan menggunakan persamaan:

$$\bar{x} = \frac{M_{10}}{M_{00}} \ldots\ldots\ldots\ldots\ldots \quad (2.7)$$

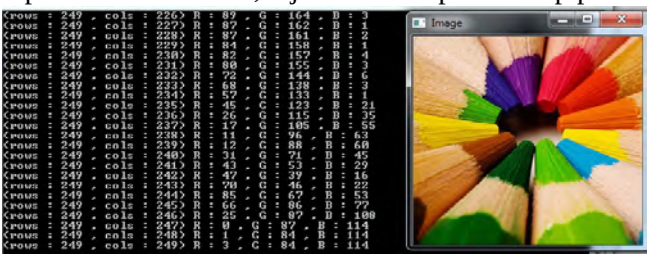
$$\bar{y} = \frac{M_{01}}{M_{00}} \ldots\ldots\ldots\ldots\ldots \quad (2.8)$$

Dimana:
M_{00} = *Area/Volume*
M_{10} = *Sum over X*
M_{01} = *Sum over Y*

Penggunaan *centre of mass* dapat digunakan pada objek yang bentuknya tidak beraturan, dan arah orientasi objek yang berubah. Posisi *centre of mass* dari sebuah objek dapat berubah apabila pada saat proses segmentasi terdapat terdapat *noise*.

2.2.9 *Scanning Pixel Value*

Dalam sebuah citra, pada setiap pikselnya memiliki nilai warna merah, hijau dan biru. Nilai masing-masing warna tersebut pada setiap pikselnya dapat kita ketahui dengan melakukan proses *scanning* piksel. Proses ini membaca komposisi warna merah, hijau dan biru pada setiap pikselnya.



Gambar 2.14 Proses *Scanning* piksel

Gambar 2.14 menunjukkan hasil dari proses *scanning* piksel. *Rows* dan *cols* menunjukkan lokasi koordinat piksel. Setiap piksel mengandung nilai dari RGB dan kemudian ditampilkan dalam program. Proses *scanning* piksel dapat dilakukan dalam berbagai arah:

- Dari kiri ke kanan dimulai dari sebelah atas
- Dari kiri ke kanan dimulai dari sebelah bawah
- Dari kanan ke kiri dimulai dari sebelah atas
- Dari kanan ke kiri dimulai dari sebelah bawah
- Dari atas ke bawah dimulai dari sebelah kiri

- Dari atas ke bawah dimulai dari sebelah kanan
- Dari bawah ke atas dimulai dari sebelah kiri
- Dari bawah ke atas dimulai dari sebelah kanan

2.2.10 *OpenCV*

OpenCV (*Open Source Computer Vision*) adalah sebuah *Open Source BSD-licensed library* yang berisi lebih dari 500 algoritma yang telah dioptimalisasi dengan baik untuk pengolahan gambar dan video analisis yang bisa didapatkan dari *http://SourceForge.net/projects/opencvlibrary*. *Library* ini ditulis dengan bahasa C dan C++, serta dapat dijalankan dengan Linux, Windows, dan Mac OS X. OpenCV dirancang untuk efisiensi komputasional dan dengan fokus yang kuat pada aplikasi *real-time.*

OpenCV terdiri dari berbagai *library open source* yang dikhususkan untuk melakukan pengolahan citra. Tujuannya adalah agar komputer mempunyai kemampuan yang mirip dengan cara pengolahan visual pada manusia. *Library* ini dibuat untuk bahasa C/C++ sebagai optimasi aplikasi *real-time*. OpenCV memiliki API (*Application Programming Interface*) untuk pengolahan tingkat tinggi maupun tingkat rendah. Pada OpenCV juga terdapat fungsi-fungsi siap pakai untuk menampilkan, menyimpan, serta mengakuisisi gambar dan video.

Gambar 2.15 *Software* yang digunakan dalam pengolahan citra [9]

Gambar 2.15 penggunaan openCV dapat berjalan diberbagai *platform* dan *Operating System*. *Platform* yang

biasa digunakan yaitu dengan menggunakan Eclipse (minimum versi OpenCV 2.0), Microsoft Visual studio 2008 hingga Microsoft Visual Studio 2013 (minimum versi OpenCV 2.0), Android (minimum versi OpenCV 2.4.2), Java (minimum versi OpenCV 2.4.4), iOS (minimum versi OpenCV 2.4.2), Windows (minimum versi OpenCV 2.0), Linux (minimum versi OpenCV 2.0). *Software* microsoft visual studio, eclipse, dan lain sebagainya yang telah dipadukan dengan openCV, digunakan untuk menjalankan program.

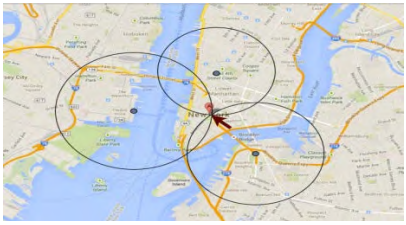
2.3 *GPS (Global Positioning System)*

Global positioning system (GPS) merupakan suatu sistem yang digunakan untuk menentukan koordinat posisi dari suatu *device* maupun sebagai alat navigasi. GPS biasanya terdiri dari GPS *receiver* dan satelit. GPS memanfaatkan data yang didapat dari satelit untuk mengkalkulasi posisi di bumi. Pada umumnya semua GPS bekerja dengan cara yang sama, yang membedakan hanya banyaknya satelit yang dapat dikomunikasikan oleh GPS. Kebanyakan *receiver* dideskripsikan sebagai 12 *channel*, yang berarti GPS dapat berkomunikasi dengan 12 satelit.

Gambar 2.16 Sistem GPS di berbagai macam *device* [24]

Gambar 2.16 merupakan *device* yang telah dilengkapi sistem GPS *rece*iver. *Device* dapat berupa *smartwatch, smartphone, tablets,* komputer, alat navigasi pada mobil, dan alat-alat lainnya. GPS pada *smartphone* biasa dimanfaatkan untuk navigasi dengan bantuan aplikasi *map* seperti *Google Map*.

Dengan memanfaatkan data-data informasi yang didapatkan satelit, GPS *receiver* yang ada pada *smartphone* dapat mengolah data-data tersebut untuk menentukan koordinat posisi dari pengguna *smartphone* serta menavigasi lokasi yang akan dituju oleh pengguna.



Gambar 2.17 Sistem GPS satelit [25]

Gambar 2.17 merupakan sistem kerja satelit untuk mengkalkulasi koordinat posisi GPS *receiver*. Dalam menentukan posisi untuk proses navigasi, GPS *receiver* menggunakan proses matematis yang disebut dengan *trilateration*. Dalam geometri, *trilateration* merupakan proses untuk menetukan lokasi titik relatif dengan pengukuran jarak menggunakan geometri dari beberapa lingkaran, bola maupun segitiga. Untuk *trilateration* dua dimensi (2D) pada GPS digunakan geometri lingkaran dari radius satelit.

Untuk mendeteksi koordinat posisi dalam bentuk tiga dimensi (3D), *trilateration* memanfaatkan geometri dari *sphere* dan membutuhkan empat jarak untuk solusi yang lebih unik dibanding menggunakan tiga. GPS *receiver* menggunakan *trilateration* 3D untuk memberitahu pengguna a.) lokasi pengguna serta b.) ketinggian pengguna di bumi saat itu. Dalam melakukan pengkalkulasian, setiap GPS *receiver* membutuhkan data sebagai berikut:

1. Lokasi dari paling tidak empat satelit GPS diatasnya dan;

2. Jarak antara *receiver* dengan masing-masing satelit GPS,

GPS *receiver* menentukan posisi dan ketinggian dengan menganalisa sinyal radio yang ditransmisikan oleh satelit-satelit dan mengukur waktu yang dibutuhkan oleh sinyal untuk menempuh jarak dari satelit ke *receiver*. Jika *receiver* dapat mendeteksi lebih dari empat satelit maka tingkat keakuratannya meningkat.

2.4 Kompas *Smartphone*

Kompas pada *smartphone* memanfaatkan fungsi dari magnetometer. Magnetometer sendiri merupakan alat yang digunakan untuk mengukur tingkat kemagnetan dari suatu material atau digunakan untuk mengukur kekuatan medan magnet. Dalam kasus ini, magnetometer dimanfaatkan untuk mengetahui arah dari medan magnet yang ada di bumi. Dengan memanfaatkan informasi kekuatan medan magnet bumi maka *smartphone* dapat mengolah data tersebut menjadi informasi lain berupa kompas.

2.5 *Mean Tests for Paired Samples*

Mean tests for paired samples merupakan metode untuk menguji kesamaan atau *equality* rata-rata dari dua *samples*. Untuk melakukan uji perbedaan rata-rata dari *treatment* tertentu, *items* dari sumber yang sama dikerjakan dua perlakuan yang sama dan kemudian hasilnya diukur. Uji ini hanya diperuntukkan bagi data yang berpasangan (*paired data*) saja.

Uji *paired-data t* digunakan untuk menguji hipotesis *null* H_0: $\delta = \delta_0$, dimana δ adalah perbedaan rata-rata antara kedua *samples*. Nilai dari δ_0 merupakan nilai hipotesa dari δ, dan dan biasanya dihipotesakan $\delta_0 = 0$; dimana tidak ada

perbedaan diantara *samples* yang dikenakan dengan *treatments*. Rata-rata dari perbedaan $\bar{d}$ digunakan untuk mengestimasi δ. Nilai dari $\bar{d}$ didapatkan dari perhitungan perbedaan d_i (i = 1, 2, 3, …, n) dari data X_{i1} dan X_{i2}.

$$d_i = X_{i1} - X_{i2} \text{……..(2.9)}$$

$$\bar{d} = \frac{\sum d_i}{n} \text{……..(2.10)}$$

Untuk nilai *test statistic t*:

$$t = \frac{(\delta - \delta_0)\sqrt{n}}{s_d} \text{……....(2.11)}$$

dimana $\bar{d} = \delta$. Nilai dari standard deviasi dari perbedaannya (s_d) tidak diketahui, maka dapat dicari dengan menggunakan persamaan sebagai berikut:

$$s_d = \left(\frac{\sum d_i^2}{n-1} - \frac{n}{n-1}\bar{d}^2\right)^{1/2} \text{……...(2.12)}$$

Nilai dari *degree of freedom* (υ) untuk distribusi *t* didapat dari:

$$\upsilon = n - 1 \text{……(2.13)}$$

Dengan nilai *n* merupakan banyaknya *samples*. *Abscissa scale* pada kurva OC untuk *paired t test* dua sisi merupakan:

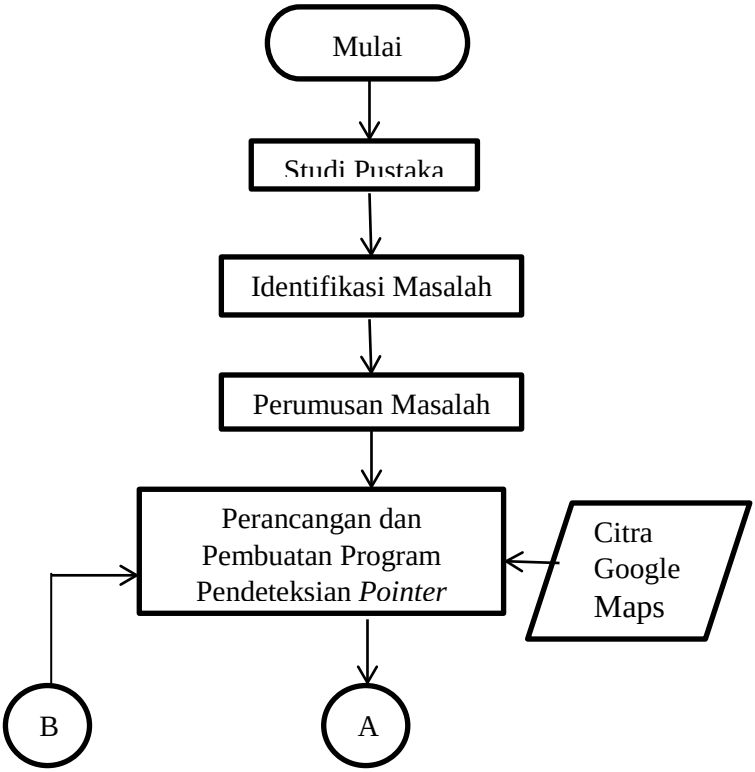
$$d = \frac{|\delta - \delta_0|}{s_d} \text{……(2.14)}$$

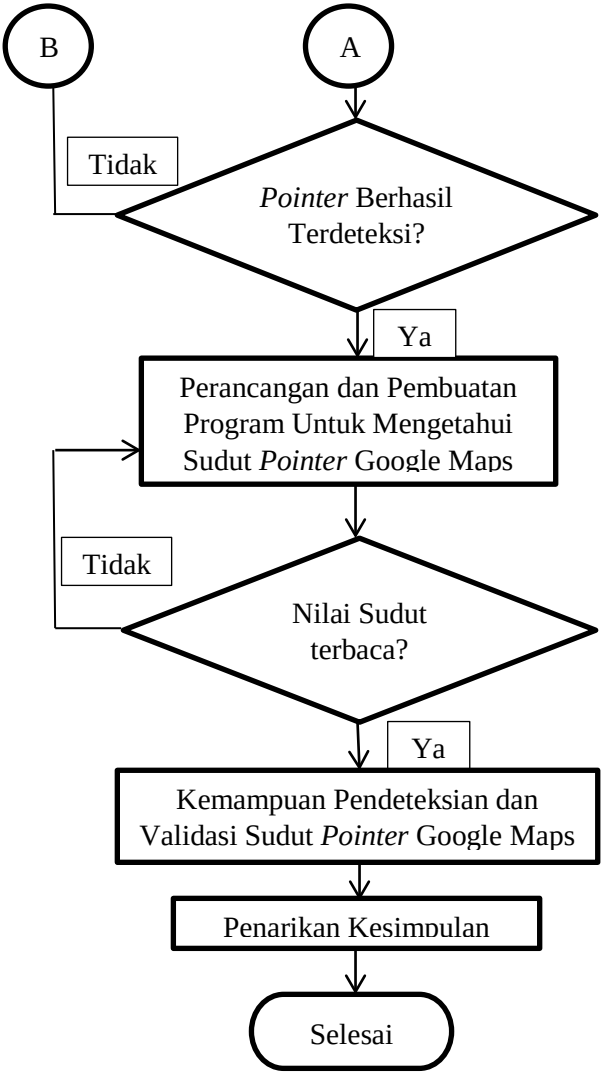
Nilai *t* yang didapat dari persamaan kemudian dibandingkan dengan nilai *t* yang didapat dari tabel distribusi *t*. Untuk mendapatkan nilai *t* dari tabel distribusi *t* dibutuhkan data *degree of freedom* (υ) serta nilai *significance level* (α). Apabila nilai *t* yang didapat dari persamaan memiliki nilai yang lebih besar dari nilai *t* yang didapat dari tabel maka hipotesa H_0 ditolak karena berada di luar area yang dapat diterima. Sedangkan apabila nilai *t* yang didapat dari persamaan sama besar atau lebih kecil dari nilai *t* yang didapat dari tabel maka hipotesa H_0 gagal ditolak.

BAB III
METODE PENELITIAN

Pada Bab III ini akan memaparkan langkah-langkah sistematis dan terarah yang akan dijadikan sebagai acuan kerangka penelitian yang bertujuan untuk menciptakan aplikasi deteksi dan nilai sudut dari *pointer* Google Maps ketika berputar ke kanan maupun ke kiri. Tahapan penelitian seperti yang tertera pada *flowchart* berikut:

3.1 Flowchart Penelitian





Gambar 3.1 *Flowchart* penelitian

Gambar 3.1 merupakan *flowchart* penelitian secara keseluruhan. Langkah awal dalam penelitian yaitu studi pustaka.

Dimana pada proses ini penulis melakukan kajian yang berhubungan tentang Google Maps dan *image processing.* Dengan kajian ini penulis mendapatkan ilmu yang berkaitan tentang penelitian yang akan dikerjakan. Langkah selanjutnya adalah identifikasi masalah dimana pada proses ini penulis menganalisa permasalahan yang sedang terjadi dan mengetahui sebab dari permasalahan tersebut. Langkah selanjutnya yaitu perumusan masalah, dimana pada proses ini penulis merumuskan masalah – masalah yang ada. Langkah selanjutnya yaitu proses perancangan dan pembuatan program pendeteksian *pointer* Google Maps, dimana pada proses ini penulis memulai merancang bangun program untuk mendeteksi *pointer* google maps. Input yang digunakan dalam program yaitu citra *pointer* Google Maps secara *real*-time. Tanda apabila *pointer* dapat terdeteksi adalah terdapat tulisan "*pointer"* pada bagian tengah bentuk *pointer* Google Maps. Kemudian terdapat proses *lopping*, apabila *pointer* tidak terdeteksi, maka kembali ke proses perancangan dan pendeteksian *pointer* google maps. Namun Apabila *pointer* berhasil terdeteksi, maka lanjut ke proses berikutnya yaitu proses perancangan dan pembuatan program untuk mengetahui nilai sudut α *pointer* Google Maps ketika berputar ke kanan maupun ke kiri. Nilai sudut tersebut ditampilkan dalam sebuah *windows*. Lalu terdapat *looping*, apabila nilai sudut α tidak terbaca dalam *windows*, maka kembali ke proses perancangan dan pembuatan program untuk mengukur sudut α. Namun apabila sudut α dapat ditampilkan dalam sebuah *windows* maka lanjut ke proses selanjutnya yaitu proses mengetahui kemampuan pendeteksian *pointer* dan validasi nilai sudut α *pointer* Google Maps. Selanjutnya adalah penarikan kesimpulan dari apa yang telah dikerjakan dalam penelitian dan selesai.

3.2 Tahap Persiapan

Tahap persiapan ini terdiri dari tiga tahapan yaitu studi pustaka, identifikasi masalah dan perumusan masalah. Studi

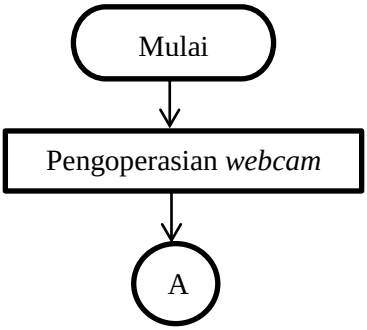
pustaka adalah acuan referensi penulis untuk mendalami permasalahan yang akan diteliti mengenai pembuatan aplikasi program untuk *image processing* dari buku-buku referensi dan jurnal yang berkaitan dengan permasalahan yang akan dibahas. Kemudian identifikasi masalah dilakukan untuk mengetahui permasalahan yang ada. Sedangkan perumusan masalah mencakup perancangan program untuk mendeteksi *pointer* Google Maps dan mencari nilai sudut *pointer* ketika berputar ke kanan maupun ke kiri.

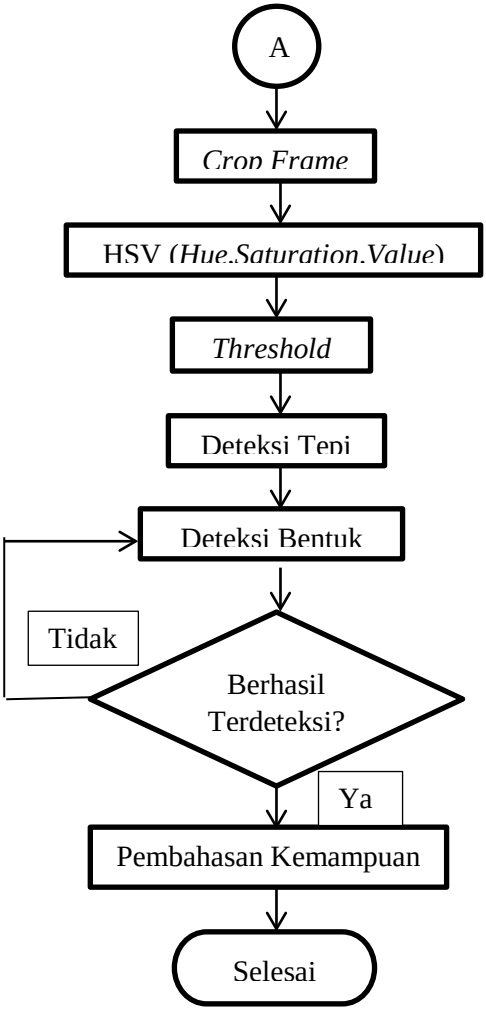
3.3 Tahap Perancangan Program

Pada tahap ini, perancangan program dilakukan untuk mendeteksi bentuk *pointer* Google Maps, sekaligus mencari nilai sudut dari *pointer* Google Maps ketika berputar ke kanan maupun kiri. Dalam pembuatan dan perancangan program ini dibutuhkan beberapa peralatan yaitu *webcam*, laptop dan *smartphone*. *Webcam* digunakan dalam *image processing* untuk menangkap gambar yang ada pada *smartphone*. Laptop digunakan untuk merancang program sekaligus menjalankan program. *Smartphone* digunakan untuk menampilkan bentuk *pointer* Google Maps.

3.3.1 *Flowchart* Perancangan Program Deteksi *P*ointer Google Maps

Berikut ini adalah *flowchart* perancangan program untuk mendeteksi *pointer* Google Maps.





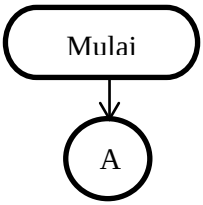
Gambar 3.2 *Flowchart* perancangan program deteksi *pointer* Google Maps

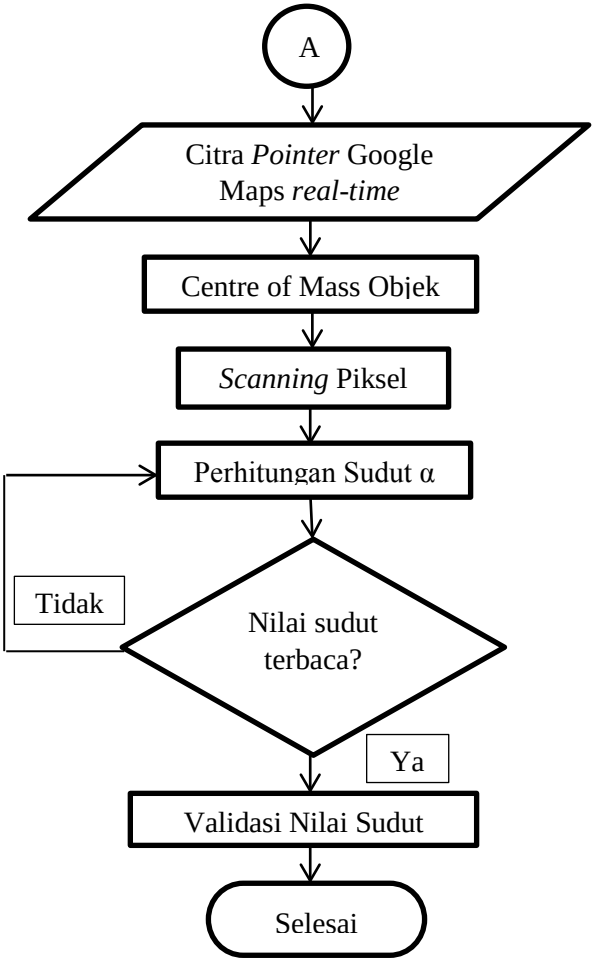
Gambar 3.2 merupakan *flowchart* perancangan program untuk mendeteksi pointer Google Maps secara *real-time*.

Input yang diinputkan yaitu berupa gambar Google Maps secara *real-time*. Langkah selanjutnya yaitu mengoperasikan *webcam*, yang digunakan untuk menangkap gambar pada layar *smartphone*. Langkah selanjutnya yaitu *crop frame* yang digunakan untuk memudahkan dalam proses pendeteksian *pointer*. Langkah selanjutnya yaitu melakukan konversi warna RGB menjadi HSV. Dengan menggunakan format warna HSV, dapat dilakukan pendeteksian *pointer* berdasarkan warna *pointer*. Langkah selanjutnya adalah melakukan proses *threshold* dengan tujuan agar bentuk objek *pointer* dapat terpisah dengan *background*. Langkah selanjutnya yaitu melakukan pendeteksian tepi untuk mengetahui piksel warna putih yang menjadi tepi dari bentuk *pointer* terputus atau terseambung dengan titik piksel tepi warna putih lainnya. Langkah selanjutnya yaitu mendeteksi bentuk dengan memanfaatkan fungsi CV*_ApproxPolyDP* yang ada pada OpenCV. Langkah selanjutnya yaitu proses *lopping* untuk menentukan apakah *pointer* dapat terdeteksi atau tidak. Sebagai tanda apabila *pointer* dapat terdeteksi yaitu terdapat tulisan "*pointer*" pada bagian tengah objek. Apabila objek *pointer* tidak terdeteksi maka kembali ke proses deteksi bentuk. Namun apabila berhasil dideteksi, lanjut ke proses kemampuan program dalam mendeteksi *pointer* dan selesai.

3.3.2 Flowchart Perancangan Program Pencarian Nilai Sudut *Pointer*

Berikut ini adalah *flowchart* dari perancangan program pencarian nilai sudut *pointer* Google Maps ketika berputar ke kiri maupun kanan.





Gambar 3.3 *Flowchart* perancangan program pencarian nilai sudut *pointer* Google Maps

Gambar 3.3 merupakan *flowchart* perancangan program untuk mengetahui nilai sudut α ketika *pointer* berputar ke kiri maupun ke kanan. Langkah awal dengan mengetahui titik *centre of mass* objek dari bentuk *pointer* Google

Maps. Kemudian melakukan proses *scanning* piksel untuk mendapatkan titik ujung *pointer*. Dari dua titik tersebut didapatkan 2 buah garis, yang dapat merepresentasikan nilai sudut α yang akan dicari. Garis yang pertama adalah garis vertikal yang didapat dari titik *centre of mass* objek. Garis yang kedua didapat dari titik *centre of mass* objek dan titik dari proses *scanning* piksel. Berdasarkan kedua garis tersebut, lalu ke proses perhitungan sudut untuk mendapatkan nilai sudut α. Setelah itu terdapat proses *lopping* untuk menenetukan apakah nilai sudut α dapat terbaca atau tidak. Apabila nilai sudut α tidak terbaca maka kembali ke proses perhitungan sudut α. Tanda nilai sudut α dapat terbaca yaitu dengan ditampilkannya nilai sudut α dalam *windows*. Setelah nilai sudut α terbaca, lanjut pada proses validasi nilai sudut *pointer* dengan nilai sudut kompas digital dan selesai.

3.3.3 Cara Kerja

Cara kerja dalam percobaan ini adalah dengan mendeteksi *pointer* Google Maps dengan input citra *real time.* Kemudian citra *real time* ini dianalisa dan diproses (*image processing*) menggunakan program C++ dari *library* openCV dan *Software* Visual Studio. Dari hasil program pendeteksian didapatkan *pointer* tersebut telah terdeteksi. Setelah didapatkan hasil *pointer* yang telah terdeteksi, dilanjutkan merancang program pencarian nilai sudut *pointer* Google Maps.

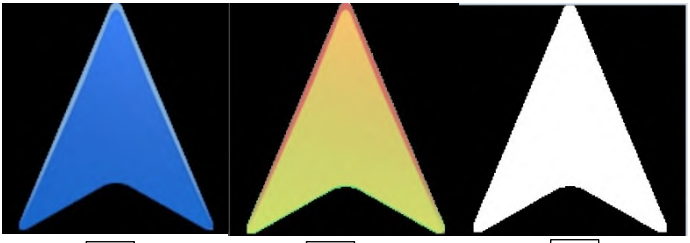
3.3.4 Proses dan Pengerjaan

Ada beberapa tahapan dalam proses dan pengerjaan tugas pada perancangan program yaitu *pre-processing*, *processing* dan *post-processing.*

1. *Pre-Processing Image*

Pre-processing adalah algoritma yang diterapkan pada gambar yang digunakan untuk mempertegas gambar atau objek yang akan diambil. Ini adalah fase penting dan umum dalam *image processing*. Langkah ini perlu diambil dalam

setiap program analisis gambar . Tanpa *pre-processing* yang tepat, maka segmentasi dan prosedur pengenalan selanjutnya dapat sangat rumit dan tidak efektif. Alasan utama melakukan *pre-processing* adalah untuk melakukan perbaikan citra sehingga pada gambar yang diproses tersorot wilayah yang diinginkan, khususnya pada kasus ini adalah pendeteksian *pointer Google Maps*.



Gambar 3.4 Tahap *pre-processing* (a) RGB *image*, (b) HSV *image*, (c) *thresholding*

Gambar 3.4 menunjukkan langkah-langkah pre-*processing* diterapkan pada contoh citra yang melibatkan dua proses utama yaitu HSV (*hue,saturation,value)* dan *thresholding*.

2. Processing Image

Processing Image merupakan proses yang digunakan untuk mengolah objek dari hasil *pre-processing* agar objek dapat dideteksi dan didapat informasi yang dibutuhkan. *Processing Image* yang digunakan dalam pengerjaan adalah *shape detection, centroid, scanning pixel* dan *bounding box*. Proses pendeteksian *pointer* digunakan untuk mengetahui posisi mobil dengan bantuan *gps* pada kondisi sebenarnya. Terdapat beberapa pendekatan dan cara yang dapat di implementasikan untuk mencari bentuk *pointer* dan kemudian di deteksi citranya. Dilakukannya segmentasi untuk daerah dengan bentuk karakteristik unik adalah

perhatian utama dalam mendeteksi *pointer*. Dalam hal ini *find contour* membantu untuk menemukan daerah yang diduga sebagai pointer. Setelah proses *find contour* dan *threshold*, bentuk dari kontur *pointer* tersebut belum bisa terdeteksi. Selanjutnya dilakukan pendeteksian dengan menggunakan *shape detection*. Metode ini mendeteksi bentuk berdasarkan jumlah ujung yang melengkung dari kontur yang telah didapatkan. Setelah *pointer* terdeteksi, kemudian dilakukan proses *bounding box* dimana area didalam *bounding box* merupakan *pointer* yang telah terdeteksi. Proses untuk pendeteksian citra *pointer* telah selesai, selanjutnya adalah proses untuk mengetahui nilai sudut yang dihasilkan dari *pointer*. Untuk mencari nilai sudut perubahan posisi *pointer* dibutuhkan beberapa metode yang digunakan sebagai dasar untuk mendapatkan nilai sudut *pointer*. Diperlukan pembuatan 2 garis pada citra *pointer* agar dapat mengetahui sudut yang akan dicari nilainya. Untuk dapat menggambar garis pada citra diperlukan 2 titik pada piksel. Titik yang pertama adalah titik *center of mass*, yang didapatkan dengan menggunakan fungsi *library image moment* pada *opencv*. Titik kedua merupakan titik ujung atas segitiga yang didapat dengan menggunakan metode *scanning pixe*l. Dari 2 titik tersebut dihasilkan garis pertama. Sedangkan garis kedua didapat dari titik *center of gravity (cg)* dan titik (cg + y) piksel. Sehingga dari kedua garis tersebut didapat nilai sudut yang nantinya akan digabungkan dengan sudut yang dihasilkan dari jalan dalam *image processing*.

3. *Post Processing*

Post processing merupakan proses yang digunakan untuk menampilkan hasil akhir dari citra awal *pointer* setelah dilakukan berbagai macam proses. Hasil akhir dari program adalah berupa *pointer* yang telah dideteksi dan nilai sudut yang didapatkan dari perubahan posisi *pointer*.

3.4 Tahap Analisa

Tahap analisa terbagi menjadi pembahasan dan penarikan kesimpulan. Tahap ini merupakan tahap implementasi dari program yang telah dirancang. Menganalisa hasil pendeteksian program dan nilai sudut yang didapatkan dari hasil *image processing* secara *real time.* Hasil sudut yang didapatkan dari program kemudian divalidasi dengan kompas digital lalu menganalisa kegagalan program secara *real-time*. Penarikan kesimpulan dan saran dari penelitian yang telah dilakukan menjadi tahapan terakhir dari penelitian ini.

[*Halaman ini sengaja dikosongkan*]

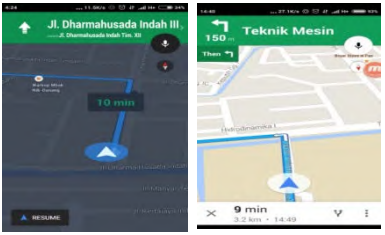
BAB IV
RANCANG BANGUN PROGRAM

4.1 Implementasi Sistem Perancangan Program

Program aplikasi deteksi *pointer* Google Maps dan nilai sudut yang dibentuk oleh *pointer* Google Maps dibuat dengan menggunakan teknologi *computer vision.* Perangkat lunak yang digunakan dalam pembuatan sistem adalah Microsoft Visual Studio dan OpenCV. Untuk mengetahui performa dari suatu program aplikasi deteksi *pointer* Google Maps dan nilai sudut yang dibentuk oleh *pointer* Google Maps, maka dibutuhkan pengujian dengan menggunakan data. Pengujian data berupa *image* secara *real-time* memiliki spesifikasi sebagai berikut:

a. Warna *pointer* Google Maps berwarna biru atau pada kondisi *day mode*
b. Setting kompas pada Google Maps arah utara dapat berubah-ubah.
c. Letak *pointer* Google Maps dalam layar *smartphone* tidak berpindah tempat.

Dalam sistem ini perlu dilakukan pengaturan Google Maps terlebih dahulu, sebelum dilakukan *image processing.* Pengaturan Google Maps yang pertama adalah pada kondisi *day mode*.

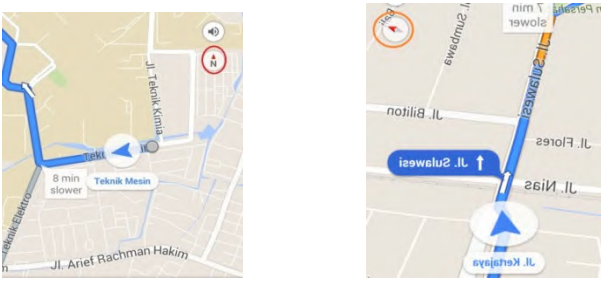


Gambar 4.1 Perbedaan Warna *Pointer* Google Maps

Gambar 4.1 menunjukkan bahwa warna *pointer* Google Maps tidak selalu berwarna biru, namun dapat berubah menjadi

warna putih. Google Maps memiliki 2 macam mode yaitu *day mode* dan *night mode*. Dimana Google Maps pada kondisi *day mode* dan *night mode* memiliki beberapa perbedaan. Pada kondisi *day mode*, *pointer* Google Maps berwarna biru, lingkaran di sekitar area *pointer* berwarna putih dan jalan yang tidak dituju/jalan sekitar *pointer* berwarna putih. Sedangkan pada kondisi *night mode*, *pointer* Google Maps berwarna putih, lingkaran di sekitar area *pointer* berwarna biru dan jalan yang tidak dituju/jalan sekitar berwarna hitam. Dalam *image processing* yang digunakan adalah Google Maps dengan kondisi *day mode*, karena antara *pointer* dan jalan dipisahkan dengan lingkaran berwarna putih, sehingga *pointer* yang berwarna biru dapat diproses, dan tidak terganggu oleh jalan yang berwarna biru pula.

Pengaturan Google Maps berikutnya adalah arah utara kompas yang dapat berubah untuk memudahkan dalam pendeteksian dan perhitungan sudut.



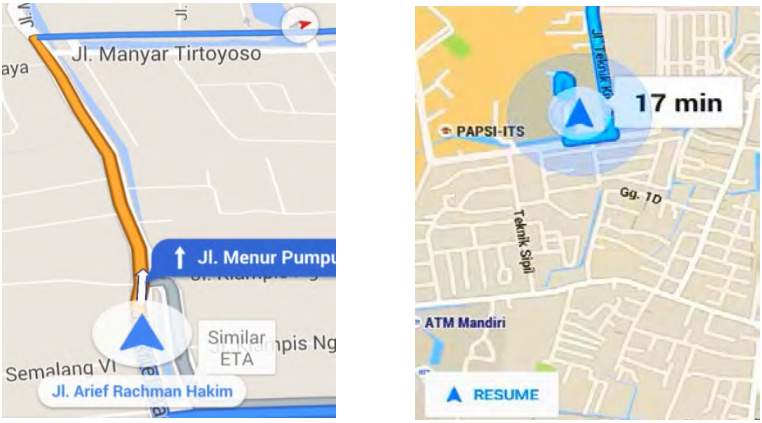
(a) (b)

Gambar 4.2 (a)Arah utara diam/tidak berubah, (b) Arah utara dapat berubah-ubah

Gambar 4.2 menunjukkan bahwa pada saat kita melakukan navigasi pada Google Maps, kompas pada Google Maps dapat diatur terlebih dahulu. Terdapat dua pengaturan kompas pada Google Maps, yang pertama adalah arah utara kompas Google Maps diam atau tidak berubah. Pada pengaturan

ini, arah ujung *pointer* sebagai arah depan dapat menghadap ke segala arah. Pengaturan yang kedua yaitu kompas Google Maps arah utara dapat berubah-ubah. Perbedaannya adalah, pada pengaturan ini arah ujung *pointer* sebagai arah depan hanya mampu berputar tidak lebih dari 90° *clockwise* maupun *counter clockwise.*

Pengaturan Google Maps berikutnya adalah letak *pointer* Google Maps dalam layar smartphone tidak berpindah tempat.



(a) (b)

Gambar 4.3 (a) Letak *pointer* tidak re-center, (b) *Pointer* re-center

Gambar 4.3 menunjukkan perbedaan posisi *pointer* dalam tampilan layar smartphone. Letak *pointer* dalam gambar (a) tidak dalam keadaan posisi *re-center*, sehingga untuk melihat letak *pointer* pada saat navigasi, pengguna harus menggeser layar untuk melihat posisi *pointer* sedang dimana. Sedangkan letak *pointer* dalam gambar (b) dalam keadaan posisi re-center, sehingga letak posisi *pointer* selalu di tengah atau tidak berubah. Dari pengaturan Google Maps yang telah disebutkan, akan mempermudah dalam *image processing* untuk pendeteksian *pointer* Google Maps dan pencarian nilai sudut yang dibentuk oleh *pointer* Google Maps.

Mesin pengolah yang digunakan untuk *image processing* adalah computer dengan spesifikasi sebagai berikut:

a. Processor Intel(R) Core(TM) i5-2430M CPU @2,40GHz
b. Memory 6,00 GB
c. System Type 64-Bit Operating System
d. Operating System Windows 7 Ultimate

4.2 Konstruksi Program

Pada tahap konstruksi ini membahas tahap-tahap penerapan metode yang digunakan dalam pembuatan sistem pendeteksian *pointer* Google Maps dan pencarian nilai sudut yang dibentuk oleh *pointer* Google Maps. Proses konstruksi ini dimulai dari tahap deteksi *pointer* Google Maps, lalu pembuatan dua garis yang merupakan representasi arah dari *pointer* Google Maps, dan setelah itu dihitung nilai sudutnya berdasarkan dua garis tersebut. Konstruksi sistem dibangun ke dalam kode program menggunakan tools yang telah ditentukan. Program yang digunakan untuk menerapkan metode tersebut kedalam kode program adalah Microsoft Visual Studio 2012 dengan library OpenCV dengan bahasa pemrograman C++.

4.2.1 Deteksi *Pointer* Google Maps

Kode program 1 : *Pre-processor Directive*

```
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/core/core.hpp>
#include <math.h>

using namespace cv;
using namespace std;
```

Pre-processor directive disebut juga pengarah *compiler* karena fungsinya untuk mengatur proses kompilasi. *#include* merupakan satu jenis pengarah *pre-processor* yang

digunakan untuk membaca *file* yang dinamakan *file* judul (*header file*). iostream, conio.h, stdio.h, windows.h, math.h, dan string.h merupakan file header yang merupakan standard library dari C++. Sedangkan <opencv2/highgui/highgui.hpp>, <opencv2/imgproc/imgproc.hpp> dan <opencv2/core/core.hpp> merupakan file *header* yang hanya ada pada *library* opencv. Setiap *file header* berhubungan dengan perintah masukan (*input*), perintah keluaran (*output*), dan fungsi-fungsi yang digunakan dalam suatu program. *Namespace* merupakan grup atau kumpulan *class, object, function* yang dikelompokkan dalam satu nama. *Using namespace std* adalah standar *namespace* dari C++ yang dapat kita gunakan untuk memanggil *class/object/function* yang terdapat di dalam C++. Sedangkan *using namespace cv* digunakan untuk memanggil *class/object/function* yang terdapat pada OpenCV.

Kode Program 2: Open Webcam

```
int main()
{
      VideoCapture cap (1);
```

Sebelum menuliskan fungsi *VideoCapture* pada pemrograman, terlebih dahulu dituliskan *int main().* Pernyataan ini mendeklarasikan fungsi utama, bahwa suatu program C++ dapat berisi banyak fungsi, yang harus selalu memiliki sebuah fungsi utama (main function), Fungsi adalah modul yang berisi kode-kode untuk menyelesaikan masalah-masalah tertentu. Tanda "{" (kurung kurawal) menandakan dari awal program dimulai. Pada awal program yaitu VideoCapture cap (1), merupakan sebuah fungsi dari OpenCV yang digunakan untuk meng-aktifkan kamera. Untuk nilai (1) maka yang akan aktif adalah kamera

eksternal, apabila nilai (0) maka yang akan aktif adalah kamera *default* yaitu kamera bawaan yang ada pada laptop.

Kode Program 3: Pembuatan Trackbar

```
namedWindow
(“control_pointer”,CV_WINDOW_NORMAL);

cvCreateTrackbar(“HueL”,”control_pointer”,
&HueL_pointer,255);
cvCreateTrackbar(“HueM”,”control_pointer”,
&HueM_pointer,255);

cvCreateTrackbar(“SaturationL”,”control_po
inter”,&SaturationL_pointer,255);
cvCreateTrackbar(“SaturationM”,”control_po
inter”,&SaturationM_pointer,255);

cvCreateTrackbar(“ValueL”,”control_pointer
”,&ValueL_pointer,255);
cvCreateTrackbar(“ValueM”,”control_pointer
”,&ValueM_pointer,255);

cvCreateTrackbar(“erode”,”control_pointer”
,&Ero_pointer,255);
cvCreateTrackbar(“Dilate”,”control_pointer
”,&Dil_pointer,255);
cvCreateTrackbar(“Blur”,”control_pointer”,
&Blur_pointer,255);
```

Pembuatan *trackbar* digunakan untuk mengatur nilai *hue saturation* dan *value*. Namun tidak hanya itu, bisa juga digunakan untuk mengatur nilai *erode*, *dilate* dan *blur*. Masing-masing warna memiliki nilai *hue*, *saturation*, *value* yang berbeda. Nilai *hue*, *saturation*, *value*, dicari nilainya

agar sesuai dengan warna yang akan dilakukan *image processing*. Pada penelitian ini warna yang akan di proses adalah warna biru dari *pointer* Google Maps. *Range* nilai dalam *trackbar* tersebut mulai dari 0 sampai 255. Dengan mengetahui nilai dari *hue*, *saturation*, *value* dapat memisahkan objek berwarna dengan *background*-nya.

Kode Program 4: *Crop* Gambar

```
Rect kotak (180,110,140,110);
cropimage = frame(kotak);
```

Pada saat kamera *webcam* aktif, otomatis akan menampilkan *windows* keluaran dari kamera yang beresolusi 640x480 piksel. Ukuran tersebut terlalu besar untuk memproses *pointer* Google Maps saja. Maka dari itu diperlukan sebuah *windows* baru dengan ukuran yang lebih kecil dan diambil dari *windows* sebelumnya yang mempunyai ukuran 640x480 piksel. Dalam penulisan kode diatas didapat ukuran *windows* yang sesuai untuk *pointer* Google Maps. Nilai 180 dan 110 merupakan letak koordinat x,y yang akan diambil dari 640x480 piksel. Sedangkan nilai 140, 110 yaitu pertambahan piksel ke arah x dan y.

Kode Program 5: HSV dan Threshold

```
cvtColor(cropimage,hsv,COLOR_BGR2HSV

inRange(hsv,Scalar(HueL_pointer,Saturati
onL_pointer,ValueL_pointer),Scalar(HueM_
pointer,SaturationM_pointer,ValueM_point
er),thr);

erode(thr,thr,getStructuringElement(MORP
H_CROSS,Size(Ero_pointer+1,Ero_pointer+1
),Point(Ero pointer,Ero pointer)));
```

```
dilate(thr,thr,getStructuringElement(MORPH
_CROSS,Size(Dil_pointer+1,Dil_pointer+1),P
oint(Dil_pointer,Dil_pointer)));

GaussianBlur(thr,thr,Size((Blur_pointer+1)
,
(Blur_pointer+1)),Blur_pointer,Blur_pointe
r);
```

Setelah didapat ukuran *windows* tertentu yang sesuai dengan *pointer* Google Maps, langkah selanjutnya adalah memproses gambar yang ada di *windows* tersebut. Gambar terlebih dahulu diubah formatnya dalam bentuk format warna HSV. Pada OpenCV fungsi tersebut tersedia dengan menuliskan code BGR2HSV. Trackbar digerakkan untuk mencari nilai *hue*, *saturation*, *value*. Nilai tersebut masuk dalam kode inRange, Proses *erode*, *dilate* dan *blur* dapat dilakukan untuk mempertegas gambar. Hasil keluaran dari kode inRange berupa gambar biner yaitu putih dan hitam. Dalam hal ini gambar atau objek yang diambil yaitu *pointer* Google Maps telah terpisah dari *background*-nya.

Kode Program 6: Canny Edge Detector

```
Canny (thr,edge,10,10,5,false);
```

Pada OpenCv terdapat salah satu fungsi yang digunakan untuk mendeteksi tepi dari sebuah objek/gambar yaitu *canny edge detector*. Dalam mendeteksi tepi dari sebuah gambar, terlebih dahulu gambar harus diubah menjadi biner yaitu hitam dan putih. Deteksi tepi digunakan untuk membatasi antara daerah putih dan hitam. Dengan deteksi tepi ini, bentuk objek yang akan dideteksi menjadi lebih jelas.

Kode Program 7: Morphological Transformations

```
morphologyEx(edge,edge,MORPH_GRADIENT,no
Array(),Point(0,0),1);
```

Operasi morfologi adalah teknik pengolahan citra yang didasarkan pada bentuk segmen atau *region* dalam citra. Operasi ini dapat diterapkan pada citra biner ataupun *grayscale*. Pada saat pendeteksian tepi, tentunya banyak *noise* yang berakibat pendeteksian tepi terputus. Maka dari itu operasi morfologi dibutuhkan untuk menutup lubang-lubang/ruang kosong yang ada pada garis tepi. Hasil operasi morfologi kemudian dimanfaatkan untuk pengambilan keputusan dengan analisis lebih lanjut.

Kode Program 8: Contour

```
vector<vector<Point>>contours;

findContours(edge.clone(),contours,CV_RE
TR EXTERNAL,CV CHAIN APPROX SIMPLE);
```

Kontur dapat didefinisikan sebagai garis yang terhubung dengan titik-titik piksel secara terus menerus (sepanjang batas) dan memiliki warna atau intensitas yang sama. Untuk mendapatkan kontur yang lebih akurat, maka gambar perlu dijadikan biner dan dideteksi tepinya. Fungsi “find contour” dapat menemukan *boundary* objek meskipun bentuk dari objek tidak beraturan, namun fungsi ini belum bisa digunakan untuk mendeteksi objek yang telah didapatkan konturnya. Perlu pendefinisian lebih lanjut terhadap kontur yang telah didapatkan.

Kode Program 9: ApproxPolyDP

```
vector<Point> approx;

for (int i = 0; i < contours.size(); i++)
{
approxPolyDP(Mat(contours[i]),approx,arcL
ength(Mat(contours[i]), true)*0.02,
true);

if (3 <= approx.size() <=4)
{
setLabel(detection,"Pointer",contours[i])
;
} }
```

“ApproxPolyDP” merupakan salah satu fungsi dari OpenCv yang digunakan untuk memperkirakan jumlah garis lengkungan. Sebelum ApproxPolyDp ini digunakan, maka terlebih dahulu objek yang akan dideteksi harus ditemukan konturnya. Dengan menggunakan fungsi ini, kontur tersebut dapat dideteksi bentuknya. Kontur pertama kali diidentifikasi dengan jumlah simpul yang didapat. Dari jumlah simpul yang didapat, objek tersebut dapat diidentifikasi. Apabila jumlah simpul 3 maka dapat dikatakan objek tersebut segitiga. Jumlah simpul 4 adalah persegi, jumlah simpul 5 adalah segilima, jumlah simpul 6 adalah segienam, dan seterusnya. Setelah proses identifikasi selesai, nama dari objek tersebut dapat ditampilkan.

Kode Program 10: Bounding Rectangle

```
vector<vector<Point>>contours_poly(contou
rs.size());
vector<Rect> boundRect( contours.size()
);

for( int i = 0; i < contours.size(); i++
)
{
approxPolyDP( Mat(contours[i]),
contours_poly[i], 15, true );
boundRect[i] = boundingRect(
Mat(contours_poly[i]) );

Scalar color = Scalar(0,255,255);
rectangle(detection, boundRect[i].tl(),
boundRect[i].br(), color, 2, 8, 0 );
rectangle(frame(kotak),
boundRect[i].tl(), boundRect[i].br(),
color, 2, 8, 0 );
}
```

Fungsi dari OpenCV yang lain adalah *bounding rectangle*, fungsi ini dapat digunakan untuk membatasi objek yang akan dideteksi dengan *background*. Sebelum fungsi ini digunakan, objek terlebih dahulu ditemukan konturnya. Setelah kontur ditemukan, lalu kontur itu akan otomatis dibatasi dengan *rectangle*. Hal ini seakan mempertegas bahwa objek yang akan dideteksi adalah objek yang berada pada *bounding rectangle*. Ukuran dari *bounding rectangle* ini dapat berubah-ubah sesuai dengan ukuran kontur yamg didapat. Letak dari *bounding rectangle* juga otomatis mengikuti kontur tersebut.

4.2.2 Pencarian Nilai Sudut *Pointer*

Kode Program 11: Centroid

```
float sumx=0,sumy=0;
float num_pixel=0;
for(int x=0;x<edge.cols; x++){
for(int y=0;y<edge.rows;y++){
int val =edge.at<uchar>(y,x);
if( val>=253)
{
sumx += x;
sumy += y;
num_pixel++;
}}}

Point p(sumx/num_pixel, sumy/num_pixel);
Moments m = moments(thr,false);
Point p1(m.m10/m.m00, m.m01/m.m00);
circle(detection, p, 5, Scalar(0,0,255),
-1);
```

Dalam OpenCv terdapat fungsi *moments* yang digunakan untuk mencari *centroid* atau *centre of mass* dari sebuah objek yang telah terdeteksi. Dalam hal ini *centroid* dibutuhkan dalam perhitungan nilai sudut. Objek yang akan dilakukan proses pencarian *centroid* ini terlebih dahulu dirubah dalam bentuk format biner. Karena pada prosesnya, untuk menentukan titik *centroid*, fungsi ini membaca satu-persatu piksel yang berwarna putih pada sebuah gambar. Setelah didapat total keseluruhan lokasi piksel berwarna putih berada, maka dengan otomatis, didapatkan lokasi titik centroid dari objek tersebut.

KodeProgram 12: Scanning Pixel

```
for(int j=detection.rows-1;j>=0;j--)
{
for(int i=detection.cols-1; i>=0;i--)
{
Vec3b intensity =detection.at<Vec3b>(j,i);
int blue = intensity.val[0];
int green = intensity.val[1];
int red = intensity.val[2];
int total = (red+green+blue)/3;

if(total>253)
{
baris=j;
kolom=i;
}}}

circle(detection,Point(kolom+2,baris),2,Sc
alar( 0, 0, 255 ),2,8 );
```

Dalam OpenCv tidak terdapat fungsi untuk melakukan *scanning* piksel. Namun kita dapat melakukan fungsi *scanning* piksel tanpa OpenCv. *Scanning* piksel berfungsi untuk mendapatkan nilai intensitas warna merah, hijau, dan biru dalam sebuah *image*. Dalam hal ini *scanning* piksel dapat dimanfaatkan untuk mengetahui lokasi piksel titik ujung segitiga sebagai arah depan mobil. Scanning piksel dapat dimulai dari 8 arah:

1. *Scanning* dari kiri ke kanan dimulai dari atas ke bawah.
2. *Scanning* dari kiri ke kanan dimulai dari bawah ke atas
3. *Scanning* dari kanan ke kiri dimulai dari atas ke bawah
4. *Scanning* dari kanan ke kiri dimulai dari bawah ke atas
5. *Scanning* dari atas ke bawah dimulai dari kiri ke kanan

6. *Scanning* dari atas ke bawah dimulai dari kanan ke kiri
7. *Scanning* dari bawah ke atas dimulai dari kiri ke kanan
8. *Scanning* dari bawah ke atas dimulai dari kanan ke kiri

Pada code diatas scanning dimulai dari kiri ke kanan dimulai dari atas ke bawah. Dengan *scanning*, titik piksel ujung *pointer* Google Maps dapat diketahui lokasinya pada koordinat x,y.

Kode Program 13: Perhitungan Sudut *Pointer* Google Maps

```
//PerhitunganSudut
line(detection,Point(sumx/num_pixel,sumy/
num_pixel),Point(kolom+2,baris),Scalar(0,
255,255),2,8);

line(detection,Point(sumx/num_pixel,
sumy/num_pixel)
,Point(sumx/num_pixel,sumy/num_pixel-
150), Scalar(0,255,255),2,8);

float x=(sumx/num_pixel-(kolom+2));
float y=(sumy/num_pixel-baris);

sudut_akhir= atan(x/y)*180/3.14;

printf("%f\n",sudut_akhir);
```

Dalam perhitungan nilai sudut dibutuhkan dua garis, sehingga dapat diketahui sudut alfa yang dicari nilainya. Dalam pembuatan sebuah garis pada OpenCV, minimal dibutuhkan 2 titik agar garis tersebut bisa tergambar pada *windows*. Titik yang akan digunakan dalam menggambar garis pada *windows* adalah yang pertama titik *centroid* dan yang kedua adalah titik ujung depan *pointer* Google Maps.

Garis pertama dibentuk dari titik *centroid* dan titik ujung *pointer* Google Maps. Sedangkan garis kedua dibentuk dari titik *centroid* dan titik *centroid* dikurangi 150 piksel (garis ini merupakan garis vertikal). Dengan dua garis tersebut, maka dapat dicari nilai alfa, dan hasil dari perhitungan sudut secara *real time* ditampilkan dalam *windows*.

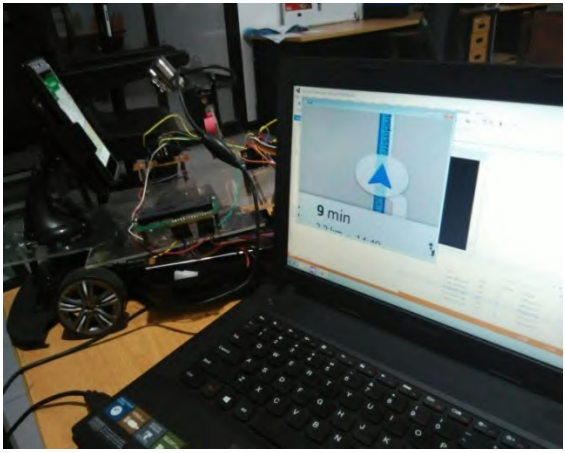
[*Halaman ini sengaja dikosongkan*]

Bab V
Pengujian Program dan Pembahasan

Pada langkah awal, pengerjaan program yang dilakukan adalah mendeteksi *pointer* google maps, agar kamera dapat mengenali pointer google maps. Setelah pointer terdeteksi, maka langkah selanjutnya adalah mencari nilai sudut yang didapat dari arah pointer. Proses pendeteksian pointer dan pencarian nilai arah pointer Google Maps telah tercantum pada gambar *flowchart* 3.2 dan gambar *flowchart* 3.3.

5.1 Pengoperasian *webcam*

Pada langkah awal dalam image processing yaitu mengoperasikan *webcam* eksternal yang telah tersambung dengan komputer / laptop. Fitur openCV yang digunakan adalah "*videoCapture*", dimana fitur ini mempunyai fungsi untuk menangkap sebuah gambar ataupun video.



Gambar 5.1 Pengoperasian *webcam*

Gambar 5.1 merupakan hasil dari fungsi "VideoCapture" pada openCV. Terlihat dari layar laptop menampilkan sebuah *windows*

yang menunjukkan bahwa *webcam* dapat dioperasikan dengan baik. Ukuran gambar yang mampu ditangkap oleh *webcam* adalah 640 x 480 piksel sedangkan format warna yang dihasilkan adalah RGB. Fokus pada webcam dapat diatur sesuai yang diinginkan. Pengaturan fokus pada webcam disesuaikan dengan jarak antara smartphone dan webcam. Pengaturan fokus ini bertujuan agar pada tahap awal penangkapan objek di depan kamera dapat terlihat dengan jelas. Webcam maupun jenis kamera yang lain merupakan alat yang peka terhadap cahaya. Maka dari itu cahaya yang berasal dari luar harus diminimalisir agar kamera dapat menangkap objek yang akan dideteksi dengan jelas.

5.2 *Crop* Gambar

Langkah berikutnya setelah webcam dapat dioperasikan dan diatur fokusnya, adalah mengambil bagian tertentu dari gambar awal dan dapat ditampilkan dalam windows.
`

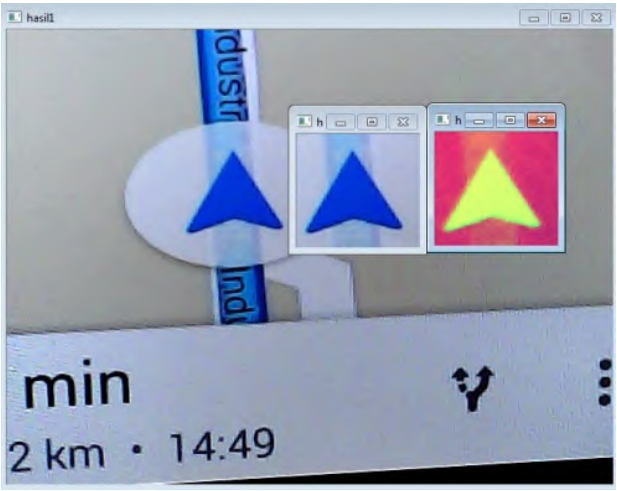
Gambar 5.2 *Crop* gambar

Gambar 5.2 merupakan hasil dari program untuk *crop* gambar. Terlihat terdapat *windows* berukuran kecil dari ukuran *windows awal*. *Crop* gambar membantu memudahkan dalam pendeteksian

pointer Google Maps, karena dalam *windows* yang berukuran kecil tidak terdapat warna biru yang lain dan bentuk objek yang lain. *Crop* gambar dapat digunakan ketika letak *pointer* tidak berubah posisi dalam tampilan smartphone. Maka dari itu Google Maps arah utara harus diatur untuk dapat berubah dan dilakukan *re-center*. Dengan pengaturan tersebut, letak *pointer* menjadi di tengah dan tidak berubah. Posisi kamera juga tetap, agar penangkapan gambar *pointer* tidak berubah atau keluar dari ukuran *frame* yang telah ditetapkan.

5.3 Konversi warna dari RGB ke HSV

Langkah berikutnya setelah dilakukan *crop* gambar adalah mengubah warna RGB menjadi HSV. Dalam *image processing* perlu dilakukan segmentasi gambar yang bertujuan untuk memisahkan objek yang akan dideteksi dengan *backgroundnya.*



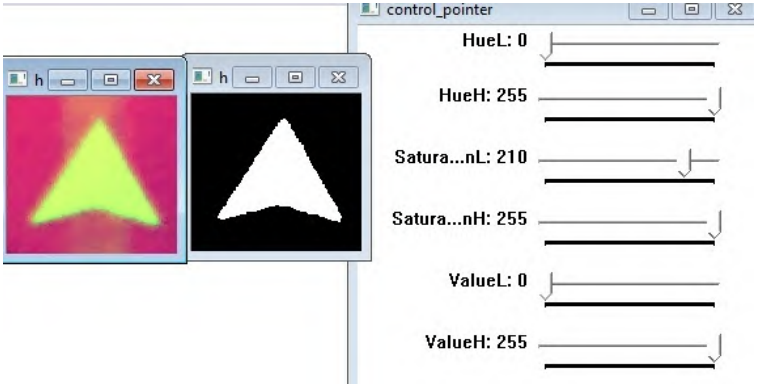
Gambar 5.3 Konversi warna RGB ke HSV

Gambar 5.3 merupakan hasil dari konversi warna RGB ke HSV. Terlihat format RGB dan HSV memiliki warna yang berbeda.

Format RGB memiliki kekurangan dalam proses segmentasi, karena tidak dapat memisahkan objek dengan *background*-nya sesuai dengan warna yang diinginkan. Hal ini dikarenakan format RGB setiap titik pikselnya merupakan campuran dari warna *Red,Green,Blue*. Sedangkan format warna HSV setiap titik pikselnya bukan merupakan campuran warna *Red,Green,Blue*, melainkan terdiri dari *hue, saturation, value,* dimana hue merupakan keluarga warna seperti merah, biru. *Saturation* merupakan banyaknya warna putih yang bercampur dengan *hue*. *Value* tingkat kecerahan dari warna tersebut. Dengan format warna HSV akan memudahkan dalam proses segmentasi.

5.4 *Threshold* & *Trackbar*

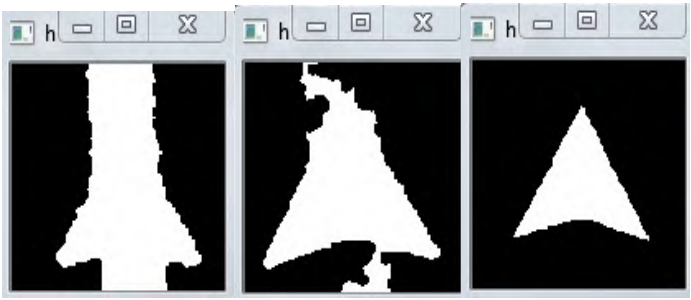
Langkah berikutnya setelah dilakukan konversi format warna ke HSV, adalah melakukan proses *threshold*, yaitu proses mengubah gambar dengan format warna HSV menjadi biner. Gambar biner merupakan format gambar dimana hanya terdapat warna hitam dan warna putih.



Gambar 5.4 Konversi gambar HSV menjadi gambar biner

Gambar 5.4 merupakan hasil konversi gambar HSV menjadi gambar biner. Dari hasil tersebut terlihat bahwa objek *pointer* Google Maps menjadi warna putih, sedangkan *background* dari

objek *pointer* berwarna hitam. Dengan gambar biner, kamera dapat membaca objek yang akan dideteksi. Proses *threshold* juga dapat disebut proses segmentasi, karena mampu memisahkan objek yang akan dideteksi dengan *background*-nya. Dalam proses *threshold* perlu dilakukan pencarian perameter nilai *hue,saturation* dan *value* yang tepat agar warna pointer yang akan dideteksi dapat terpisah dengan *background*-nya. Untuk itu perlu pembuatan *trackbar* yang digunakan untuk mencari nilai optimum HSV yang kemudian dijadikan gambar biner. Pencarian nilai optimum *hue*, *saturation* dan *value* dilakukan dengan menggerakkan *trackbar*. Dari *trackbar* tersebut dapat diketahui untuk nilai optimum dari *hue*, *saturation* dan *value*. Setelah beberapa kali percobaan untuk mengetahui nilai optimum HSV, didapat nilai optimumnya yaitu *hue* pada nilai 0, *saturation* pada nilai 210 dan *value* pada nilai 0.



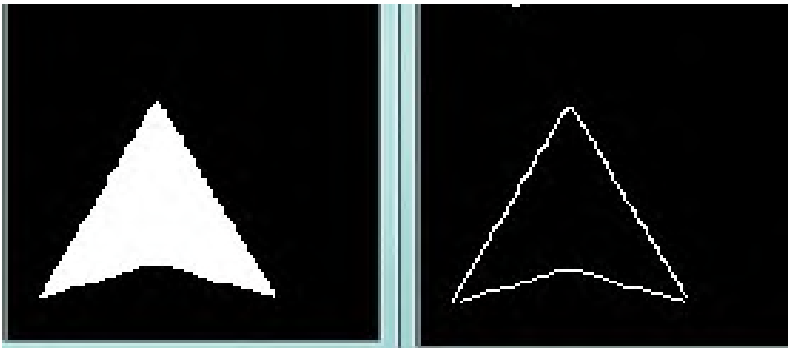
Gambar 5.5 Pengaturan *trackbar* mulai dari nilai *saturation* 0 sampai 210

Gambar 5.5 merupakan hasil dari *range* nilai *saturation* mulai dari 0-210. Hasil ini kemudian digunakan untuk proses pendeteksian objek. Hasil terbaik dilihat dari citra *threshold* yang bentuknya sama seperti *pointer*.

5.5 Deteksi Tepi

Langkah berikutnya setelah dilakukan proses threshold yang menjadikan gambar menjadi biner adalah mendeteksi tepi

dari gambar biner. Tepi yang dimaksud adalah batas antara piksel berwarna putih dan hitam.



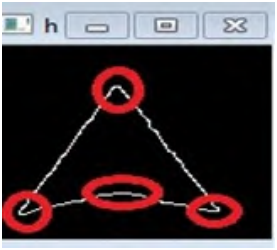
Gambar 5.6 Pendeteksian tepi *pointer*

Gambar 5.6 merupakan hasil dari pendeteksian tepi *pointer*. Dalam pendeteksian tepi ini menggunakan fungsi dari openCV yaitu *canny edge detector*. Sebelum gambar dideteksi tepinya, dapat dilihat bahwa batas antara piksel berwarna putih dan hitam terdapat noise yang menyebabkan batas piksel menjadi tidak halus layaknya seperti sebuah garis. Piksel berwarna putih yang menjadi batas tidak diketahui saling tersambung dengan piksel berwarna putih lainnya. Dengan pendeteksian tepi, maka dapat telihat jelas batas piksel berwarna putih dengan piksel sebelahnya yang berwarna hitam. Kemudian dapat telihat pula apakah masing-masing piksel berwarna putih terhubung atau tidak. Deteksi tepi yang dihasilkan fungsi "*canny"* perlu ditambahkan dengan operasi morfologi. Dimana operasi ini digunakan untuk mengisi runag kosong antar piksel yang berwarna putih pada tepinya. Sampai dengan tahap ini dapat disebut dengan *pre-processing.*

5.6 Pendeteksian Bentuk *Pointer* Google Maps

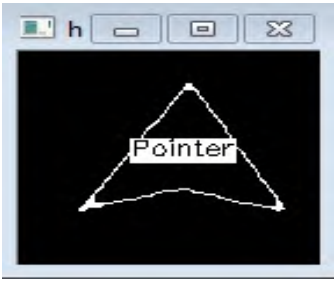
Langkah selanjutnya setelah objek *pointer* Google Maps dapat dideteksi tepinya adalah mendeteksi bentuk dari objek

pointer. Tahap ini merupakan tahap awal dari *processing image*, dimana kamera mulai dikenalkan dengan bentuk objek *pointer* Google Maps.



Gambar 5.7 *P*ointer yang berbentuk *irregular* / tidak umum

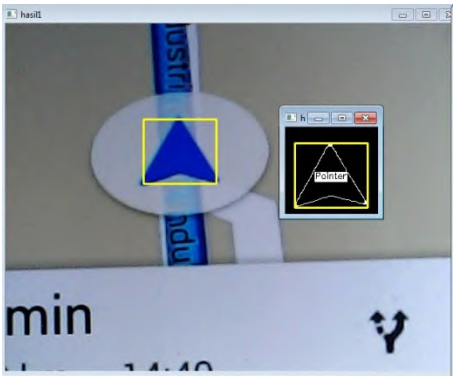
Gambar 5.7 menunjukkan bentuk dari objek *Pointer* yang berbentuk *irregular*. Bentuk ini memiliki 4 lengkungan seperti yang telah ditandai oleh lingkaran berwarna merah. Pada bentuk *regular shape* jumlah lengkungan sesuai dengan bentuknya. Contoh jika jumlah lengkungan tiga, maka disebut segitiga, jumlah lengkungan empat, disebut segiempat. Bentuk *irregular shape* objek *pointer* ini memiliki 4 lengkungan namun tidak dapat dikatakan segiempat. Bentuk *pointer* cenderung mendekati bentuk segitiga.



Gambar 5.8 Pointer terdeteksi

Gambar 5.8 menunjukkan objek *pointer* yang telah berhasil dideteksi. Tanda bahwa kamera dapat mengenali bentuk

objek *pointer* yang telah dideteksi adalah munculnya tulisan "*pointer*" pada bagian tengah objek. Dalam proses kamera mengenali objek *pointer*, perlu dilakukan pencarian kontur yang ada pada gambar. Fungsi pencarian kontur telah disediakan oleh openCV. Cara kerja dari pencarian kontur ini adalah dengan melakukan *scanning* setiap titik pikselnya. Dari titik piksel yang memiliki warna sama, otomatis akan dihubungkan titik-titik tersebut dan membentuk sebuah kontur objek yang belum dapat didefinisikan oleh kamera dan disimpan dalam pemrograman. Setelah didapatkan kontur objek, lalu menganalisa hasil kontur tersebut dengan tujuan agar objek *pointer* dapat terdeteksi. Pendeteksian objek menggunakan "CV_approxPolyDP" yang merupakan fungsi dari *library* OpenCV. Fungsi ini bertujuan untuk memperkirakan jumlah lengkungan yang ada pada kontur. Mengacu pada gambar objek *pointer* yang berbentuk seperti segitiga dan memiliki jumlah lengkungan 4 dapat dituliskan pada program ketika kontur memiliki jumlah lengkungan 3-4 maka dapat disebut dengan "*pointer*". Sehingga kamera dapat mengenali dan mendeteksi objek *pointer*.



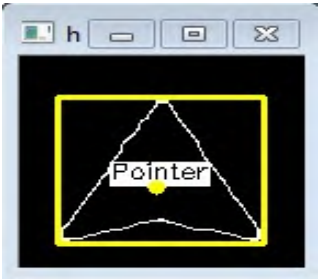
Gambar 5.9 *Bounding Box* Objek

Gambar 5.9 merupakan *pointer* yang telah terdeteksi dan ditandai dengan *bounding box*. Ukuran *bounding box* otomatis

akan mengikuti bentuk dari *pointer*. *Bounding box* ini dapat ditampilkan pada *windows* awal, yaitu *windows* yang belum dilakukan *image processing*. Dengan *bounding box* yang ditampilkan pada *windows* awal, dapat diketahui bentuk *pointer* dapat terdeteksi atau tidak.

5.7 *Centre of Mass* objek

Langkah selanjutnya setelah objek *pointer* Google Maps dapat terdeteksi adalah mengetahui nilai sudut *pointer* sebagai representasi dari arah mobil. Perlu diketahui bahwasanya *pointer* Google Maps arahnya dapat berputar ke kanan dan kiri. Hal ini terjadi ketika *pointer* melewati jalan berbelok kiri, kanan, bundaran maupun putar balik. Untuk menggambarkan garis dalam *windows* dibutuhkan dua titik. Proses pertama dalam pencarian nilai sudut yaitu menemukan titik *centre of mass* (*centroid*) dari bentuk *pointer* tersebut. Titik tengah ini akan menjadi titik pertama dalam pembuatan sebuah garis. Untuk mencari titik tengah pada sebuah objek, OpenCV telah menyediakan fungsinya yaitu dengan menggunakan *image moments*.



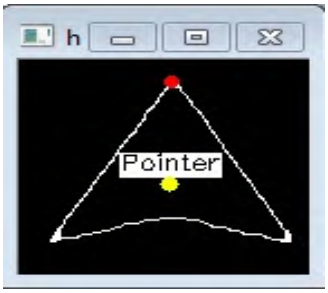
Gambar 5.10 Titik *centre of mass pointer*

Gambar 5.10 merupakan hasil dari fungsi *image moments* pada OpenCV. Dalam menggunakan fungsi ini terdapat rumusan untuk nilai x dan y yang telah dihitungkan oleh openCV. Dimana nilai x dan y merupakan koordinat dari titik tengah sebuah objek.

Setelah ditemukan koordinat titik tengah sebuah objek, titik ini dapat ditampilkan dalam bentuk lingkaran. Seperti yang terlihat pada gambar 5,13, titik kuning merupakan titik tengah dari *pointer*.

5.8 Scanning Piksel

Proses kedua dalam pencarian nilai sudut yaitu menemukan titik ujung atas dari bentuk pointer tersebut. Titik ujung atas pointer ini menandakan sebagai arah dari posisi pengguna. Titik ujung atas ini akan menjadi titik kedua dalam pembuatan sebuah garis. Untuk mencari titik atas ujung *pointer* ini dalam OpenCV tidak disediakan fungsinya. Namun titik ini bisa didapatkan dari metode *scanning* piksel pada tampilan objek pointer.

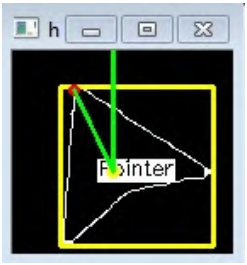


Gambar 5.11 Titik ujung segitiga

Gambar 5.11 terdapat titik merah yang merupakan hasil dari *scanning* piksel. Proses *scanning* ini dimulai dari atas ke bawah. Saat program menemukan titik piksel berwarna putih pertama kali, maka program akan menghentikan proses *scanning*. Hasil dari proses *scanning* ini didapat koordinat x dan y dan dapat ditandai dengan lingkaran berwarna merah.

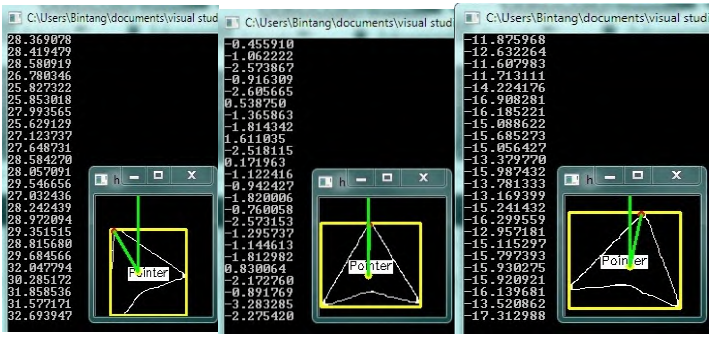
Dari hasil metode *centroid* dan *scanning* piksel, telah didapat 2 titik yaitu titik berwarna kuning dan titik berwarna merah. Selanjutnya adalah membuat 2 garis dari titik tersebut,

dimana dengan garis ini nantinya dapat diketahui sudut α(alpha) yang dicari.



Gambar 5.12 Penggambaran Garis Pada Pointer

Dalam menggambar sebuah garis pada *pointer*, OpenCv telah menyediakan fungsi *line*. Dimana fungsi ini dalam penulisan programnya, dibutuhkan 2 titik yang mempunyai lokasi koordinat x1,y1 dan x2,y2. Pada gambar 5.12 terdapat 2 garis, yaitu garis yang menghubungkan titik tengah/*centroid* dan titik ujung atas *pointer*, lalu garis kedua adalah garis vertikal berdasarkan titik tengah/*centroid*. Berdasarkan 2 garis tersebut, dapat dihitung nilai sudut α(alpha) pada program dan ditampilkan dalam sebuah *windows*.



Gambar 5.13 Nilai Sudut α(alpha)

Gambar 5.13 merupakan hasil dari program pencarian nilai sudut α(alpha) yang didapat antar garis vertikal dan garis

ujung *pointer*. Ketika garis ujung *pointer* segaris dengan garis vertikal, maka nilai sudut α(alpha) bernilai 0. Ketika garis ujung pointer posisinya di sebelah kiri garis *vertical*, maka sudut α(alpha) memiliki nilai tertentu dan bernilai positif. Sedangkan apabila garis ujung *pointer* di sebelah kanan garis vertikal, maka sudut α memiliki nilai tertentu dan bernilai negatif.

5.9 Pengujian Program Dalam Pendeteksian Pointer

Pengujian ini dilakukan untuk mengetahui seberapa efektif penggunaan metode yang diterapkan untuk mendeteksi *pointer*. Pengujian program dilakukan dengan membawa *smartphone* ke area *outdoor* dan dijalankan secara *real-time*. Lalu ketika dijalankan secara real-time gambar yang ditangkap oleh kamera disimpan kemudian dianalisa. Berikut adalah hasil pengujian pendeteksian *pointer* secara *real time*:

Tabel 5.1 Hasil Pengujian Program

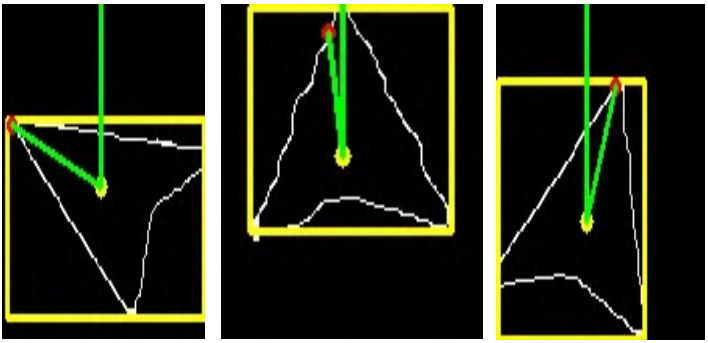
Jumlah Gambar	Terdeteksi	Tidak Terdeteksi	Kemampuan Pendeteksian
2000 gambar	1969 gambar	31 gambar	98,45%

$$\text{Kemampuan Pendeteksian} = \frac{\text{Jumlah gambar terdeteksi}}{\text{Jumlah total gambar}}$$

Dari data hasil pengujian diatas didapatkan tingkat kemampuan pendeteksian *pointer* sebesar 98,45%. Jumlah gambar yang disimpan dan digunakan untuk analisa saat *running* program bisa mencapai angka ribuan. Hal ini disebabkan *webcam* mampu mengambil gambar sebanyak 30 per detik. Maka dari itu perlu diambil *sample* sebanyak 2000 gambar yang telah mencakup kondisi *pointer* ketika berbelok ke kanan maupun ke kiri.

Dalam pendeteksian *pointer* terdapat beberapa gambar saat *running* program, dimana *pointer* tidak dapat terdeteksi.

Hasil ini dapat terlihat setelah membuka 2000 gambar yang telah disimpan saat *running* program



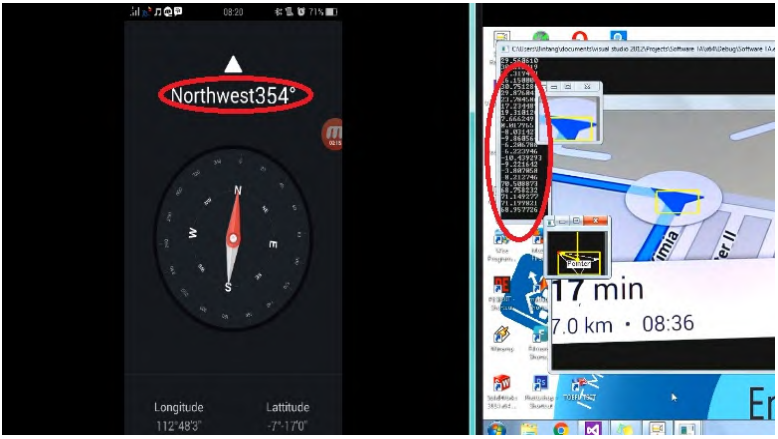
Gambar 5.14 Pointer Tidak Terdeteksi

Gambar 5.14 merupakan hasil dari beberapa gambar *pointer*, dimana program tidak mampu mendeteksi *pointer*. Terlihat pada gambar diatas, tulisan "pointer" pada bagian tengah bentuk tidak muncul. Kegagalan pendeteksian ini disebabkan oleh gambar *pointer* yang tepinya tidak terhubungkan atau terputus. Sehingga program tidak mampu membaca jumlah total lengkungan dari suatu kontur objek. Sebab lain yaitu posisi kamera *webcam* dan *smartphone* juga ikut berubah pada saat dijalankan secara *real time*, perubahan posisi ini disebabkan oleh getaran pada mobil. Getaran ini juga dapat membuat kamera tidak mampu dalam menangkap objek dengan baik.

5.10 Validasi Nilai Sudut *Pointer* Pada *Image Processing* dengan Kompas Digital di Area Outdoor

Sudut yang didapat dari hasil *image processing pointer* merupakan sebuah informasi penting yang nantinya sebagai *input* untuk mobil. Posisi *pointer* yang berada di google maps pada layar *smartphone*, merupakan representasi dari posisi dan arah pengguna. Pada kondisi awal yaitu pengguna google maps dalam keadaan diam (belum jalan), sensor GPS yang berada dalam

smartphone tidak mampu mendeteksi secara tepat dan akurat posisi pengguna maupun arah dari pengguna. GPS hanya mampu mengestimasikan bahwa posisi pengguna berada di sekitar daerah tersebut. Sedangkan untuk arah *pointer*, GPS tidak mampu mendeteksi arah pengguna secara akurat sehingga tampilan arah dari pointer berbeda dengan arah posisi pengguna pada kondisi sebenarnya. Untuk dapat GPS membaca lokasi dan arah pengguna, maka pengguna harus dalam keadaan bergerak (berjalan). Sudut dapat terjadi apabila pengguna google maps menghadap ke arah yang berbeda dari posisi awal. Nilai sudut yang didapatkan dari *image processing* kemudian divalidasi dengan menggunakan kompas digital secara *real-time*.



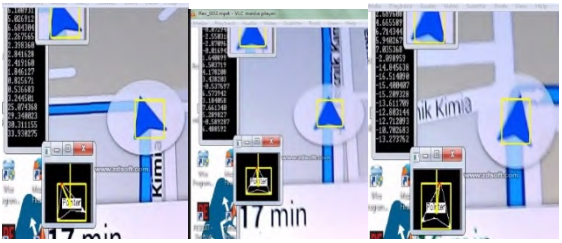
Gambar 5.15 Sudut pada kompas digital dan *image processing*

Gambar 5.15 menunjukkan nilai perubahan arah yang terlihat pada kompas dan *image processing*. Nilai sudut yang didapat dari kompas digital merupakan selisih dari posisi akhir dan posisi awal. Sedangkan nilai sudut dari *image processing* sudah ditampilkan dalam sebuah *windows.* Dari nilai sudut yang dihasilkan dari image processing kemudian di validasi dengan kompas digital untuk mengetahui bahwa nilai sudut yang dihasilkan *image processing* sama atau berbeda dengan kompas

digital. Nilai sudut kompas digital dan *image processing* dijadikan data yang kemudian diolah secara statistik menggunakan metode *paired sample t-test*.

Gambar 5.16 Cara pengambilan data sudut *real-time*

Gambar 5.16 merupakan cara yang dilakukan untuk mengambil data sudut pada kompas digital dan *image* processing. Pengambilan data membutuhkan 2 smartphone, satu digunakan untuk membuka aplikasi Google Maps dan satunya lagi digunakan untuk membuka aplikasi kompas digital. Kedua *smartphone* tersebut ditempelkan pada akrilik, lalu dipasang *webcam* untuk membaca objek *pointer* Google Maps. Akrilik ditutup dengan sebuah kain agar tidak ada cahaya yang mempengaruhi pembacaan nilai sudut *image processing*. Kemudian alat ini siap untuk dijalankan di area *outdoor*.



Gambar 5.17 Kontur jalan yang digunakan untuk pengambilan data.

Pengambilan data diambil ketika pointer melalui jalan lurus, berbelok ke kiri dan ke kanan. Hasil pembacaan nilai sudut

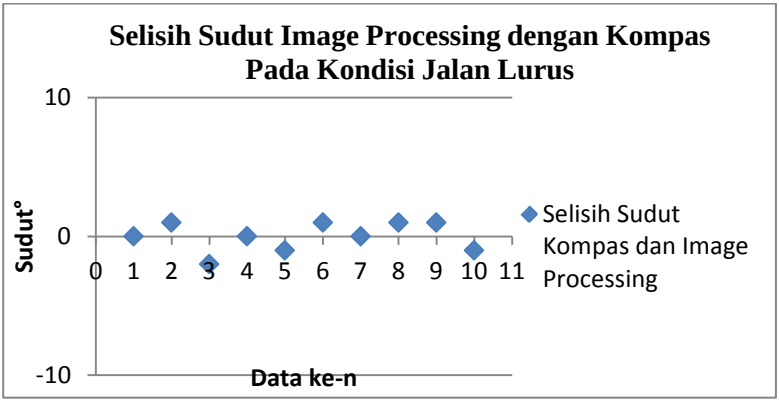
dari kompas digital dan *image processing* direkam menggunakan *screen* recorder. Data rekaman dari *screen recorder* setiap detiknya dianalisa dan dibandingkan hasil nilai sudut *image processing* dengan kompas digital. Berikut adalah data yang didapatkan dari kompas digital dan *image processing* saat *pointer* melewati jalan lurus, berbelok ke kiri dan berbelok ke kanan.

Tabel 5.2 Data Hasil Sudut dari Kompas Digital dan *Image Processing*

n	Kompas	Image Processing	di	di^2	
1	0	0	0	0	Lurus
2	1	0	1	1	
3	1	3	-2	4	
4	1	1	0	0	
5	1	2	-1	1	
6	1	0	1	1	
7	0	0	0	0	
8	1	0	1	1	
9	1	0	1	1	
10	0	1	-1	1	
11	0	3	-3	9	Belok Kiri
12	7	0	7	49	
13	15	30	-15	225	
14	52	67	-15	225	
15	55	75	-20	400	
16	4	25	30	900	
17	4	22	32	1024	
18	3	5	49	2401	
19	5	4	50	2500	
20	3	2	52	2704	

21	0	4	-4	16	
22	18	-13	31	961	
23	29	-16	45	2025	
24	37	-17	54	2916	
25	52	-49	101	10201	Belok
26	3	2	51	2601	Kanan
27	3	2	52	2704	
28	1	-3	59	3481	
29	4	-2	58	3364	
30	2	-4	59	3481	

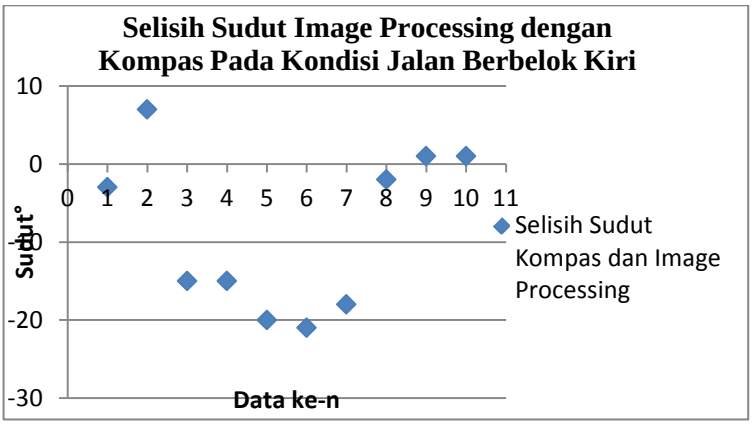
Berdasarkan data yang didapat dari tabel 5.2 ditampilkan dalam bentuk grafik. Berikut adalah grafik ketika *pointer* melewati jalan lurus, berbelok ke kiri dan berbelok ke kanan.



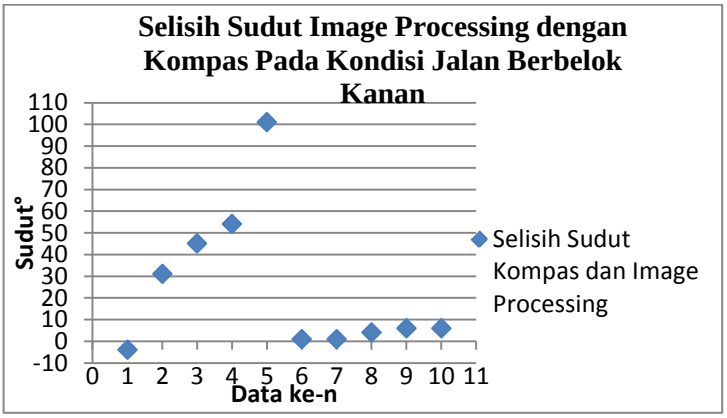
Gambar 5.18 grafik selisih nilai sudut kompas dengan *image processing* pada kondisi jalan lurus

Seperti terlihat pada grafik 5.1, selisih sudut nilai yang dihasilkan dari *image processing* dengan kompas digital nilainya tidak terlalu besar. Nilai sudut yang dihasilkan dari *image processing* pada kondisi jalan lurus mengalami *floating*, dimana

nilainya berkisar antara -1° sampai 1°. Hal ini disebabkan oleh adanya *noise* saat pembacan bentuk *pointer*.



Gambar 5.19 grafik selisih nilai sudut kompas dengan *image processing* pada kondisi jalan berbelok kiri



Gambar 5.20 grafik selisih nilai sudut kompas dengan *image processing* pada kondisi jalan berbelok kanan

Seperti terlihat pada grafik 5.2 dan grafik 5.3 terdapat selisih nilai sudut *image processing* dengan kompas digital. Hal

ini dikarenakan pergerakan nilai sudut yang didapatkan dari *image processing* dengan kompas digital berbeda. Nilai sudut yang dihasilkan dari *image processing* berubahnya tidak kontinyu. Sedangkan nilai sudut yang ada pada kompas digital berubah secara kontinyu. Pada saat *pointer* berputar dan kondisi jalan berputar, yaitu pada data ke 6 sampai 10, maka dalam pengambilan data kompas perlu dilakukan penyesuaian orientasi arah kompas. Hal ini dikarenakan pengaturan Google Maps arah utara dapat berubah-ubah sehingga *pointer* Google Maps hanya dapat berputar tidak lebih dari 90°, sedangkan data pada kompas yang dijadikan sebagai pembanding data *image processing* dapat berubah nilai hingga 360°. Penyesuaian orientasi kompas dilakukan dengan pengambilan data sudut *pointer* Google Maps dengan pengaturan arah utara tetap, kemudian nilai sudutnya diselisihkan dengan nilai sudut kompas yang ada di titik pengambilan data yang sama. Setelah penyesuaian orientasi kompas dengan kompas Google Maps didapatkan hasil selisih nilai sudut yang dihasilkan *image processing* tidak jauh berbeda dengan nilai sudut yang dihasilkan kompas.

Langkah selanjutnya adalah menganalisa hasil data tabel dengan menggunakan metode *paired sample t-test*, didapatkan hasil uji hipotesa:

$$H_0: \delta = \delta_0$$
$$H_1 : \delta \neq \delta_0$$

Dimana δ adalah selisih nilai sudut *image processing* dengan kompas dan δ_0 adalah 0.

Dari data tabel kemudian diolah, sehingga didapat nilai berikut:

$\propto = 0.05$ $\quad \bar{d} = 5.33$ $\quad t_{perhitungan} = 1.205$

$n = 30$ $\quad S_d = 24.243$

$t_{tabel} = 2.045$

Hasil nilai tersebut menunjukkan bahwa $t_{perhitungan} \leq t_{tabel.}$ sehingga H_0 gagal tolak. Dimana hipotesa H_0 adalah selisih nilai sudut *image processing* dengan kompas sama dengan 0.

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace cv;
using namespace std;

Mat frame;
Mat cropimage;
Mat hsv;
Mat thr;
Mat edge;
Mat edge1;
Mat reader;
Mat detection;

int hueL=0;
int hueH=255;

int saturationL=0;
int saturationH=255;

int valueL=0;
int valueH=255;

int baris=0;
int kolom=0;

int counter;

float sudut_akhir;

//Variabel shape detection
static double angle(Point pt1,Point pt2,Point pt0)
{
```

```
        double dx1 = pt1.x - pt0.x;
        double dy1 = pt1.y - pt0.y;
        double dx2 = pt2.x - pt0.x;
        double dy2 = pt2.y - pt0.y;
        return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 +
dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
}

void setLabel(Mat& im,const string label,vector<Point>&
contour)
{
        int fontface = FONT_HERSHEY_SIMPLEX;
        double scale = 0.4;
        int thickness = 1;
        int baseline = 0;

        Size text = getTextSize(label, fontface, scale,
thickness, &baseline);
        Rect r = boundingRect(contour);

        Point pt(r.x + ((r.width - text.width) / 2), r.y
+ ((r.height + text.height) / 2));
        rectangle(im, pt + Point(0, baseline), pt +
Point(text.width, -text.height), CV_RGB(255,255,255),
CV_FILLED);
        putText(im, label, pt, fontface, scale,
CV_RGB(0,0,0), thickness, 8);
}

int main ()
{
        VideoCapture cap (0);

        namedWindow("control_pointer",WINDOW_NORMAL);

        createTrackbar("HueL","control_pointer",&hueL,255
);
        createTrackbar("HueH","control_pointer",&hueH,255
);
```

```
	createTrackbar("SaturationL","control_pointer",&s
aturationL,255);
	createTrackbar("SaturationH","control_pointer",&s
aturationH,255);

	createTrackbar("ValueL","control_pointer",&valueL
,255);
	createTrackbar("ValueH","control_pointer",&valueH
,255);

	for (;;)
	{
		cap >> frame;

		Rect kotak(180,110,130,120);
		cropimage=frame(kotak);

		cvtColor(cropimage,hsv,CV_BGR2HSV);


	inRange(hsv,Scalar(hueL,saturationL,valueL),Scala
r(hueH,saturationH,valueH),thr);

		erode(thr, thr,
getStructuringElement(MORPH_ELLIPSE, Size(6,6)) );

		Canny(thr,edge,40,150);

		Size kernelSize (5,5);
		Mat element = getStructuringElement
(MORPH_RECT,kernelSize,Point(2,2));

	morphologyEx(edge,edge1,MORPH_CLOSE,element );

		//Shape detection
		detection = edge1.clone();
		vector<vector<Point>>contours;
		findContours(edge1.clone(),
contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
		vector<Point> approx;
```

```
		for (int i = 0; i < contours.size(); i++)
		{
			//Approximate contour with accuracy
proportional
			//to the contour perimeter

	approxPolyDP(Mat(contours[i]),approx,arcLength(Ma
t(contours[i]), true)*0.1, true);

			if (3 <approx.size() <4)

			{
				setLabel(detection,
"Pointer", contours[i]);
			}
		}


		cvtColor(detection,detection,CV_GRAY2BGR);
		//Bounding Box

	vector<vector<Point>>contours_poly(contours.size(
));
		vector<Rect> boundRect( contours.size() );

		for( int i = 0; i < contours.size(); i++ )
		{
			approxPolyDP( Mat(contours[i]),
contours_poly[i], 15, true );
			boundRect[i] = boundingRect(
Mat(contours_poly[i]) );
			Scalar color = Scalar(0,255,255);
			rectangle(detection,
boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );
			rectangle(frame(kotak),
boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );
		}
```

```
		//Centroid
		float sumx=0,sumy=0;
		float num_pixel=0;

		for(int x=0;x<edge1.cols; x++)
		{
			for(int y=0;y<edge1.rows;y++)
			{
				int val
=edge1.at<uchar>(y,x);
		if( val>=253)
				{
					sumx += x;
					sumy += y;
					num_pixel++;
				}
			}
		}

		Point p(sumx/num_pixel, sumy/num_pixel);
		Moments m = moments(thr,false);
		Point p1(m.m10/m.m00, m.m01/m.m00);

		circle(detection, p,4, Scalar(0,255,255),
-1);
		//circle(detection,Point(70,30),2,Scalar(
255,255, 255 ),2,8 );

		//Scanning Pixel
		for(int j=detection.rows-30;j>=10;j--)
		{
			for(int i=detection.cols-20;
i>=0;i--)
			{
				Vec3b intensity
=detection.at<Vec3b>(j,i);
				int blue = intensity.val[0];
				int green =
intensity.val[1];
				int red = intensity.val[2];
```

```
                                int total =
(red+green+blue)/3;

                                if(total>253)
                                {
                                        baris=j;
                                        kolom=i;
                                }
                        }
                }


        circle(detection,Point(kolom,baris),2,Scalar( 0,
0, 255 ),2,8 );


                //Perhitungan Sudut

        line(detection,Point(sumx/num_pixel,sumy/num_pixe
l),Point(kolom,baris),Scalar(0,255,0),2,8);
        line(detection,Point(sumx/num_pixel,
sumy/num_pixel) ,Point(sumx/num_pixel,sumy/num_pixel-
150), Scalar(0,255,0 ),2,8);

                float x=((sumx/num_pixel)-(kolom+2));
                float y=((sumy/num_pixel)-baris);

                sudut_akhir= atan(x/y)*180/3.14;

                printf("%f\n",sudut_akhir);

                //save frame
                stringstream file;

                cap.read(reader);
                if(reader.empty()) break;

                file << "D:/Capture/gmb" << counter <<
".jpg";
                counter++;
```

```
                imwrite(file.str(),detection);

                imshow ("hasil1",frame);
                //imshow ("hasil2",hsv);
                //imshow ("hasil3",thr);
                //imshow ("hasil4",edge);
                imshow ("hasil5",detection);
                //imshow ("hasil6",cropimage);

                if(waitKey(30) >= 0)
                        break;
        }
        return 0;
}
```

TABLE B.2 THE *t* DISTRIBUTION

	PROPORTION IN ONE TAIL					
	0.25	0.10	0.05	0.025	0.01	0.005
	PROPORTION IN TWO TAILS					
df	0.50	0.20	0.10	0.05	0.02	0.01
1	1.000	3.078	6.314	12.706	31.821	63.657
2	0.816	1.886	2.920	4.303	6.965	9.925
3	0.765	1.638	2.353	3.182	4.541	5.841
4	0.741	1.533	2.132	2.776	3.747	4.604
5	0.727	1.476	2.015	2.571	3.365	4.032
6	0.718	1.440	1.943	2.447	3.143	3.707
7	0.711	1.415	1.895	2.365	2.998	3.499
8	0.706	1.397	1.860	2.306	2.896	3.355
9	0.703	1.383	1.833	2.262	2.821	3.250
10	0.700	1.372	1.812	2.228	2.764	3.169
11	0.697	1.363	1.796	2.201	2.718	3.106
12	0.695	1.356	1.782	2.179	2.681	3.055
13	0.694	1.350	1.771	2.160	2.650	3.012
14	0.692	1.345	1.761	2.145	2.624	2.977
15	0.691	1.341	1.753	2.131	2.602	2.947
16	0.690	1.337	1.746	2.120	2.583	2.921
17	0.689	1.333	1.740	2.110	2.567	2.898
18	0.688	1.330	1.734	2.101	2.552	2.878
19	0.688	1.328	1.729	2.093	2.539	2.861
20	0.687	1.325	1.725	2.086	2.528	2.845
21	0.686	1.323	1.721	2.080	2.518	2.831
22	0.686	1.321	1.717	2.074	2.508	2.819
23	0.685	1.319	1.714	2.069	2.500	2.807
24	0.685	1.318	1.711	2.064	2.492	2.797
25	0.684	1.316	1.708	2.060	2.485	2.787
26	0.684	1.315	1.706	2.056	2.479	2.779
27	0.684	1.314	1.703	2.052	2.473	2.771
28	0.683	1.313	1.701	2.048	2.467	2.763
29	0.683	1.311	1.699	2.045	2.462	2.756
30	0.683	1.310	1.697	2.042	2.457	2.750
40	0.681	1.303	1.684	2.021	2.423	2.704
60	0.679	1.296	1.671	2.000	2.390	2.660
120	0.677	1.289	1.658	1.980	2.358	2.617
∞	0.674	1.282	1.645	1.960	2.326	2.576

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berikut kesimpulan yang didapat pada penelitian tugas akhir ini adalah Program untuk mendeteksi dan mencari nilai arah pointer GPS Google Maps sudah dirancang dan dibangun dengan kemampuan pendeteksian mencapai 98,45%, sedangkan pendeteksian arah *pointer* yang didapatkan dari *image processing* secara statistik tidak berbeda secara signifikan dengan sudut yang dianggap benar pada kompas.

6.2 Saran

Berdasarkan dari hasil pengujian dan pembahasan pada tugas akhir ini, terdapat saran pengembangan penelitian lebih lanjut. Saran yang diajukan untuk penelitian lebih lanjut menggunakan pengaturan arah utara kompas pada Google Maps tidak berubah. Hal ini disebabkan karena data nilai sudut yang didapat dari *image processing* dengan menggunakan pengaturan kompas googlemaps saat ini lebih rumit untuk divalidasi. Sehingga tingkat keakuratan dalam validasi rendah.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] NHTSA, 2013. "Preliminary Statement of Policy Concerning Automated Vehicles". ed. Washington, DC: US Department of Transportation, National Highway Traffic Safety Administration (NHTSA).

[2] Gonzales.R.C. and Richard, E. W.2002, "*Digital Image Processing*". University of Tennessee.

[3] Ian T.Y, Jan .J.G, Lucas J.V.V, 1995 "*Fundamentals of Image Processing*", Delft University of Technology.

[4] Anil K.J. 1989. *"Fundamentals of Digital Image Processing",University of California*.

[5] Kusumanto R.D, Tompunu A.N, dan Pambudi S.W, 2011 "Klasifikasi Warna Menggunakan Pengolahan Model Warna HSV". Politeknik Negeri Sriwijaya,Palembang.

[6] O'Reilly, 2008. "*Learning OpenCV*". O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[7] Benedictus, Widi H, Katon W. 2010 "Segmentasi Warna Citra Dengan Deteksi Warna HSV Untuk Mendeteksi Objek", , Universitas Kristen Duta Wacana Yogyakarta.

[8] Solomon C, Breckon T, 2011. *"Fundamentals of Digital Image Processing",* , Wiley-Blackwell.

[9] Yofiyanto.E. 2010. "KuliahInformatika". https://kuliahinformatika.wordpress.com/2010/02/13/buku-ta-thresholding-citra/

[10] Samarth B. 2013. "Practical OpenCV", , Apress.

[11] Sacan. D. 2016 "*Tecnologia Automotriz."* http://tecnologiautomotriz2016.blogspot.co.id/2015_12_01_archive.html

[12] Guizzo E. 2016 ."*How Google Self Driving Car Works"*. http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works

[13] Siddhi, Parama I.K. 2015. "Representasi Citra Digital." https://putuadisusanta.wordpress.com/2015/07/17/representasi-citra-digital

[14] Husainjr. 2013. " Aplikasi Pengolahan Citra Pada Warna." http://www.slideshare.net/husainjr/pcd-pertemuan-112

[15] Connor . 2012. "Raster and Vector." https://cobweb93blog.wordpress.com/college/year-12/new-media/unit-19/raster-and-vector/

[16] Levin, Golan. 2005. "Image Processing and Computer Vision." http://openframeworks.cc/ofBook/chapters/image_processing_computer_vision.html

[17] Pamungkas. Adi. 2011. "Morphological Operation". https://pemrogramanmatlab.wordpress.com/tag/dilasi/

[18] Hood, Paul. 2016. "How do Google Self Driving Car Work." http://www.alphr.com/cars/7038/how-do-googles-self-driving-cars-work

[19] Xenadreamer. 2015. "Digital Graphic Design." https://ucscsgdxandriaedwards.wordpress.com/

[20] Joseph, Lentin. 2012. ”How to Convert RGB to HSV using OpenCV”. http://www.technolabsz.com/2012/08/how-to-convert-rgb-to-hsv-using-opencv.html

[21] K.Hong. 2016. “Canny Edge Detection”. http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Canny_Edge_Detection.php

[22] Calumk. 2014. “Dissertation-opencv-recognizing specific shapes”. http://calumk.com/blog/0008/

[23] Rahman K.A.2013 . “OpenCV Python”. http://opencvpython.blogspot.co.id/2012/06/contours-3-extraction.html

[24] Opencv. 2016 ”Structural Analysis and Shape Descriptors.” http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[25] DC Rainmaker. 2011 “Sport Device Accuracy In Depth: Part I.” http://www.dcrainmaker.com/2011/06/2011-sport-device-gps-accuracy-in-depth.html

[26] NexGenDesign. 2009 “Lost In Tracking or Why Mobile GPS is Inaccurate?.” http://www.nexgendesign.com/lost-in-tracking-mobile-gps

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Penulis yang bernama lengkap Bintang Setya Bagus Yudiarno dilahirkan di Surabaya pada tanggal 20 Juli 1991. Merupakan anak pertama dari 4 bersaudara dari pasangan Wiwin Setya Bagus Y dan Luluk Indarini. Mengawali pendidikan formal di TK Bakti 5 Gresik, SD Muhammadiyah GKB Gresik, SMP Negeri 3 Gresik, SMA Muhammadiyah 1 Gresik dan diterima di jurusan Teknik Mesin FTI-ITS dan terdaftar dengan NRP 2110100005. Di jurusan Teknik Mesin ini, Penulis mengambil Bidang Studi Manufaktur, dan bergabung dalam Lab.Perancangan dan Pengembangan Produk. Penulis aktif dalam kegiatan lab sebagai asisten dan grader dalam praktikum mata kuliah grader. Penulis juga aktif dalam event yang diselenggarakan oleh mahasiswa mesin. Email yang bisa dihubungi yaitu: Setyabintang77@gmail.com.