BAB III

METODE PENELITIAN

Untuk pengumpulan data yang diperlukan dalam melaksanakan tugas akhir, ada beberapa cara yang telah dilakukan, antara lain:

1. Studi kepustakaan

Studi kepustakaan berupa pencarian data-data literatur dari masing-masing fungsi pada *library* OpenCV dan OpenRobotinoAPI, melalui pencarian dari internet, dan konsep-konsep teoritis dari buku-buku penunjang serta metode yang akan digunakan untuk melakukan pengolahan citra.

2. Penelitian laboratorium

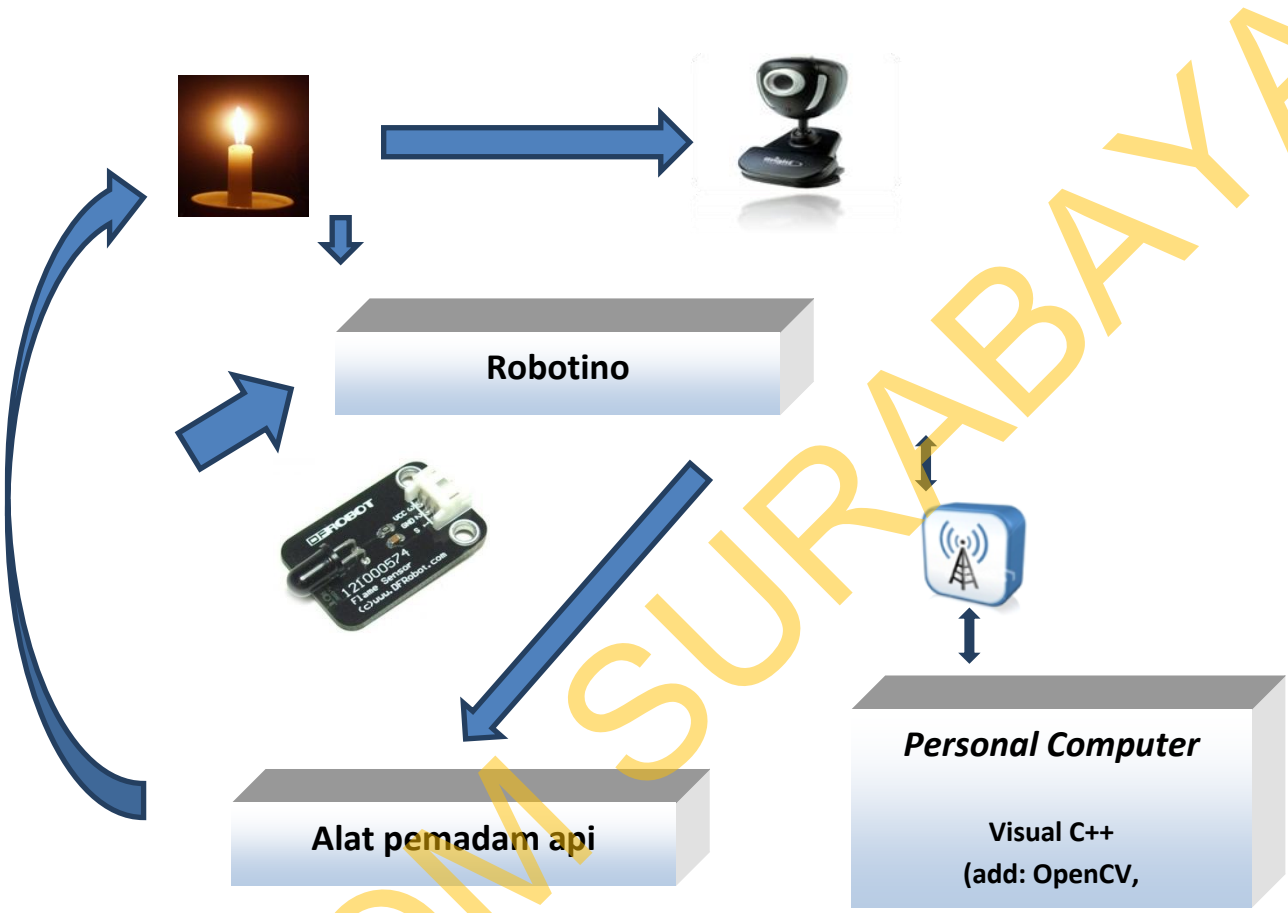
Penelitian laboratorium dilakukan dengan perancangan perangkat lunak, implementasi perangkat lunak, pengambilan data pengujian aplikasi dengan Robotino, kemudian melakukan evaluasi dari data hasil pengujian.

3.1. Perancangan Sistem dan Blok Diagram Sistem

Model penelitian yang akan dilakukan adalah model penelitian pengembangan. Untuk mempermudah dalam memahami sistem yang akan dibuat dapat dijelaskan melalui blok diagram pada Gambar 3.1.

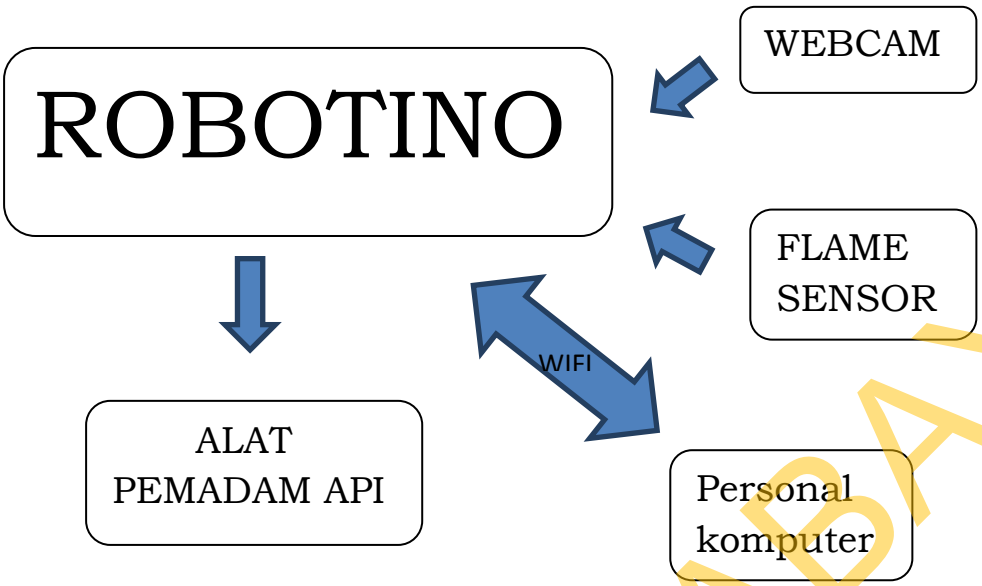
Seperti pada Gambar 3.1 pada inisialisasi awal Robotino akan melakukan pergerakan dengan berputar 360 derajat guna mendeteksi keberadaan api, setelah gambar api tertangkap oleh kamera Robotino akan bergerak menuju lokasi api tersebut, setelah jarak yg memungkinkan *flame detector* akan memastikan bahwa gambar yang tertangkap oleh kamera tersebut benar sebuah

api dengan inputan suhu panas. Jika gambar tesebut benar sebuah api maka Robotino akan mengeluarkan keluaran tegangan yang akan menggerakan aktuator (alat pemadam api), yang dapat memadamkan api tersebut.



Gambar 3.1 Blog diagram sistem secara umum

Pada Gambar 3.2 dijelaskan bahwa dalam proses kerjanya robotino akan mendapatkan 2 inputan berupa *webcam* dan *flame* sensor. Webcam digunakan untuk proses pendeteksian awal lokasi api pada suatu ruangan dan *flame* sensor beguna untuk memastikan bahwa gambar yang dideteksi oleh kamera tersebut adalah benar api sungguhan dengan mendeteksi suhu dari api tersebut.



Gambar 3.2 Blog diagram input dan output

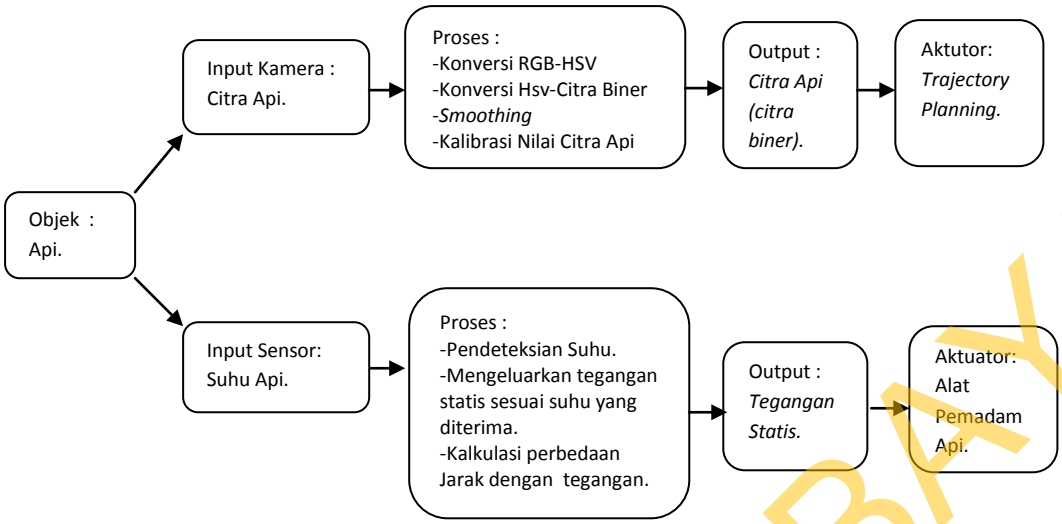
.

Keluaran yang dihasilkan berupa tegangan yang akan digunakan untuk menggerakan aktuator berupa serangkaian alat pemadam api.

1.1.1 Cara kerja sistem secara keseluruhan

Pada Gambar 3.3 dijelaskan bagaimana cara kerja sistem dan pengolahan data api berupa gambar dan suhu (radiasi), proses awal menggunakan pengolahan citra dimana *webcam* akan menangkap gambar dari api dan mengubah nilai RGB ke HSV dilanjutkan dengan proses *tresholding*.

Setelah melalui proses tersebut dilakukan proses kalibrasi nilai HSV api agar citra yang didapatkan murni merupakan komposisi warna api, dengan metode ini objek di sekitar api tidak terdeteksi walaupun memiliki keidentikan warna. Setelah melewati proses kalibrasi nilai HSV akan didapatkan citra api yang sesungguhnya dan robot akan melakukan pergerakan menuju sumber api tersebut.



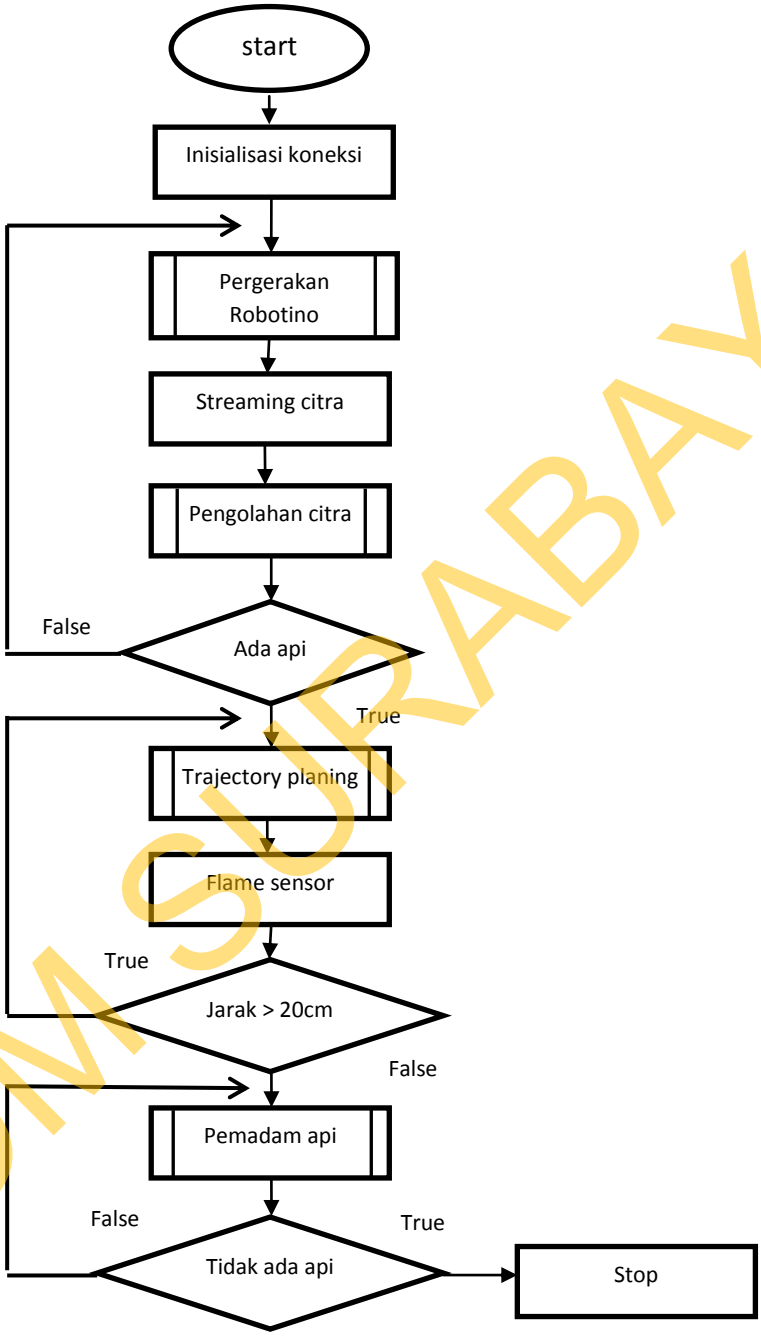
Gambar 3.3 Blog diagram sistem secara keseluruhan.

Setelah robot berada 20cm dekatnya dari api sensor akan bekerja dan mengeluarkan keluaran berupa tegangan sebesar 5v. Tegangan tersebut akan memicu aktuator berupa alat pemadam api untuk bekerja dan memadamkan api.

1.2 Perancangan Perangkat Lunak

Dalam perancangan perangkat lunak, *compiler* yang digunakan adalah Microsoft Visual C++ 2008. Untuk *library* yang digunakan pada pengolahan citra yaitu *library* OpenCV v2.1 dan *library* OpenRobotinoAPI digunakan untuk mengintegrasikan Robotino dengan PC.

Kemudian dalam penulisannya atau dalam pembuatan program, akan meliputi bagian-bagian penting dalam setiap langkah-langkah per bagian sesuai dengan algoritma atau logika sekuensial dari awal sampai output. Berikut adalah algoritma program secara global.



Gambar 3.4 *Flowchart* sistem secara global

3.3 Inisialisasi Koneksi

Tahap-tahap inisialisai Robotino meliputi cara-cara *setting* koneksi Robotino dan pergerakan Robotino. Untuk kendali Robotino digunakan OpenRobotinoAPI (*Application Programming Interface*) yaitu *library* aplikasi

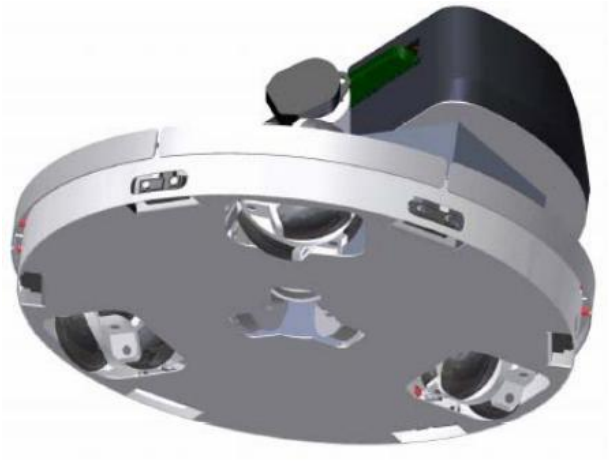
programming yang dibuat khusus untuk Robotino, yang diciptakan untuk mempermudah *user* dalam membuat program pada Robotino. *Library* ini memungkinkan akses penuh terhadap sensor dan *actuator* pada Robotino. Komunikasi antara Robotino dengan PC melalui jaringan TCP atau UDP menggunakan media *wireless.*

Untuk menghubungkan koneksi *wireless* dari PC ke *access point* Robotino digunakan prototipe fungsi com.setAddress yang digunakan untuk memberikan alamat IP yang akan diakses. Untuk simulasi pada program Robotino® SIM dapat digunakan alamat IP 127.0.0.1:8080. Untuk koneksi langsung ke *access point* Robotino, secara *default* alamat IP yang digunakan adalah 172.26.1.1.

Untuk mengakses fungsi tersebut dibutuhkan deklarasi *header* file yang terletak pada rec/robotino/com/all.h, dimana pada awal program harus dideklarasikan terlebih dahulu.

3.4 Pergerakan Robotino

Robotino memiliki sistem pergerakan *omni-directional drive* dimana terdapat 3 buah roda yang digunakan untuk menggerakan Robotino. Berikut gambar sistem *omni-directional drive* Robotino pada Gambar 3.5.



Gambar 3.5 *Omni-Directional Drive* Pada Robotino

Untuk untuk menggerakan Robotino digunakan fungsi

```
void drive(int x, int y, int z)
{
    omniDrive.setVelocity( x , y , z);
}
```

Parameter vx adalah parameter kecepatan pada sumbu x (+ maju, - mundur) dan vy adalah parameter kecepatan untuk sumbu y (+ kanan, - kiri), dengan ketentuan parameter vx dan vy dalam satuan mm/s. Dan omega merupakan parameter kecepatan sudut dengan ketentuan parameter omega dalam satuan deg/s.

Untuk mengakses fungsi tersebut dibutuhkan deklarasi *header* file yang terletak pada rec/robotino/com/all.h, dimana pada awal program harus dideklarasikan terlebih dahulu.

Pada saat tahap inisialiasi Robotino akan melakukan manuver dengan berputar 360° untuk mencari lokasi dari api.

3.5 *Streaming* Citra

Untuk menampilkan data citra yang sudah tersimpan pada *Iplimage* kedalam *window* baru digunakan prototipe fungsi pada *library* OpenCV yaitu *cvShowImage* (const char *name, const CvArr **image*). Dengan ketentuan parameter *const char *name* adalah nama *window* dan *const* CvArr **image* adalah *Iplimage* yang ditampilkan. Berikut program yang digunakan untuk menampilkan data citra secara *streaming*:

```
cam.setStreaming(true);
cvShowImage( "image", img1);
cvShowImage( "thresholded", thresholded);
c = cvWaitKey(10);
```

Untuk *refresh* citra yang ditampilkan pada *window* dibutuhkan fungsi *cvWaitKey*. Ini dikarenakan OS memiliki waktu minimum dalam menjalankan *threads* secara bergantian. Fungsi ini tidak memberikan *delay* persis seperti parameter yang telah set, namun *delay* tergantung *threads* yang sedang berjalan pada komputer saat itu. Nilai yang dikeluarkan dari fungsi ini adalah kode untuk penekanan tombol atau -1 apabila tidak ada tombol yang ditekan selama waktu yang ditentukan.

3.5.1 Penerimaan Data Citra

Setiap data citra yang dikirimkan dari *webcam* Robotino diakses dengan *pointer* bertipe *const unsigned char*. Karena resolusi *default* dari Robotino adalah 320x240 maka data untuk 1 citra yang dikirimkan adalah sebanyak 230400, dimana pada 1 *pixel* citra terdapat 3 *channel*, dan pada setiap *channel* berukuran 8bit. Ketika fungsi set*Streaming* bernilai *true*, fungsi *imageReceivedEvent* akan

terus melakukan *streaming* citra hingga koneksi diputuskan atau saat *setStreaming* bernilai *false*. Penyimpan data ke dalam format *Iplimage* akan ditunjukan seperti potongan program berikut:

```
Iplimage *ima =cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,3);

for (int i = 0; i < dataSize; i++)
{
      ima->imageData[i] = *(data+i)
}
```

Data citra yang ditangkap adalah data citra dengan ruang warna RGB dan disimpan langsung pada variabel Iplimage (*Intel Image Processing Library*) yaitu stuktur data untuk penyimpanan data citra pada OpenCV. Namun urutan *channel* data dalam Iplimage adalah BGR sehingga untuk menampilkan warna sesungguhnya, harus dikonversikan terlebih dahulu dengan fungsi cvCvt*Color* seperti baris perintah seperti berikut:

```
cvCvtColor(img2,img1, CV_BGR2RGB);
```

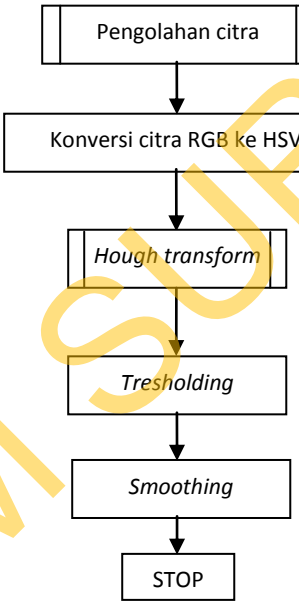
Setelah itu proses dilanjutkan dengan mengubah komposisi RGB menjadi HSV dengan menggunakan perintah seperti berikut:

```
cvCvtColor(img2,hsv, CV_BGR2HSV);
```

Ketika fungsi set*Streaming* diset dengan nilai *true*, maka fungsi *imageReceivedEvent* langsung menangkap data citra secara *streaming*. Untuk mengakses fungsi *imageReceivedEvent* dibutuhkan deklarasi *header* file yang terletak pada rec::Robotino::com:Camera.h, dimana pada awal program harus dideklarsikan terlebih dahulu. Fungsi *imageReceivedEvent* perlu dideklarasikan sebagai *class* baru karena fungsi tersebut adalah fungsi *virtual*, namun untuk penulisannya harus disertakan fungsi induknya (*parent*) karena *class* yang dibuat adalah *class* turunan (*inheritance*). Berikut potongan *class* yang dideklarasikan:

```
class MyCamera : public Camera
{
public:
MyCamera()
{
}
void imageReceivedEvent( const unsigned char* data,
unsigned int dataSize,
unsigned int width,
unsigned int height,
unsigned int numChannels,
unsigned int bitsPerChannel,
unsigned int step );
};
```

3.6 Pengolahan Citra



Gambar 3.6 *Flowchart* Pengolahan Citra

Metode utama yang digunakan pada proses pengolahan citra dalam aplikasi ini adalah *color filtering*. Dalam proses *color filtering* untuk mempermudah melakukan filter terhadap warna tertentu tanpa terpengharuh intensitas cahaya digunakan ruang warna HSV (Dhiemas, 2011), maka diperlukan proses konversi citra dari ruang warna RGB ke ruang warna HSV.

Hasil *color filtering* dalam ruang warna HSV tersebut berupa citra biner atau hitam putih, dengan menentukkan batasan nilai *threshold* pada warna yang akan dideteksi.

Hough transform digunakan untuk mencari titik tengah dari citra api menggunakan *circle detection*. Untuk mengurangi *noise* pada gambar digunakan proses *smoothing* dengan menggunakan *Gaussian Filtering* untuk mendapatkan efek *blur* pada gambar.

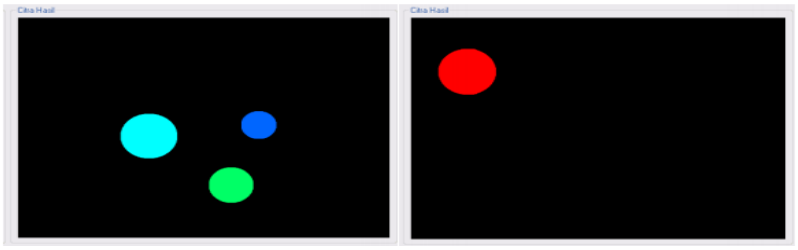
3.6.1 Konversi dari Ruang Warna RGB ke Ruang Warna HSV

HSV merupakan singkatan dari *Hue, Saturation* dan *Value*, sedangkan *color space* berarti ruang warna. Ruang warna adalah suatu metode yang digunakan untuk merepresentasikan warna menjadi suatu bentuk yang bisa diperhitungkan dengan angka, secara khusus yang terkait disini adalah tiga atau empat nilai atau komponen warna. ruang warna meminjamkan dirinya sendiri untuk untuk menghasilkan representasi suatu warna, terutama untuk representasi digital, seperti sebagai suatu hasil cetakan digital atau tampilan pada media elektronik.

Hue merupakan salah satu elemen dalam ruang warna HSV yang mewakili warna sehingga toleransi *hue* juga akan mempengaruhi nilai warna terseleksi dalam proses segmentesi. Nilai *hue* direpresentasikan dalam bentuk lingkaran dan memiliki rentang berupa sudut antara 0^o - 360^o. Penggambaran elemen warna *hue* dapat dilihat pada Gambar 3.7.

Gambar 3.7 Lingkaran Elemen Warna *Hue*

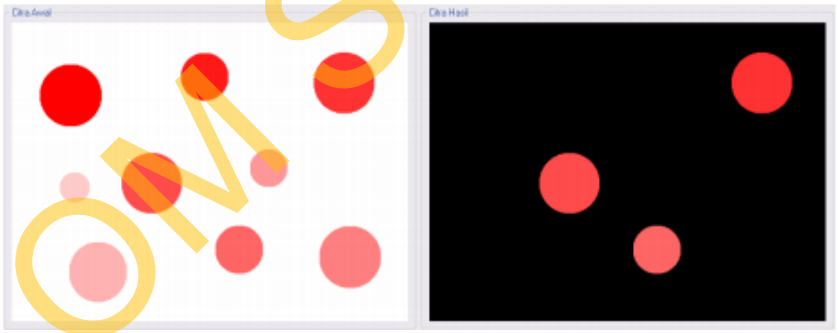
Oleh karena elemen warna *hue* berupa lingkaran dan dituliskan dalam sudut maka setiap operasi yang berkaitan dengan elemen warna *hue* (penambahan/pengurangan, perhitungan toleransi, filter warna) merupakan operasi sudut. Penambahan nilai *hue* sebesar n akan terjadi pergeseran sudut sebesar n^o searah jarum jam sedangkan untuk pengurangan sebesar n akan terjadi pergeseran sudut sebesar n^o berlawanan dengan arah jarum jam. Berikut pada Gambar 3.7 merupakan contoh penggunaan toleransi *hue* pada proses segmentasi, sebuah gambar beberapa lingkaran dengan nilai *hue* yang berbeda-beda akan tetapi memiliki nilai *saturation* dan *value* yang sama. Pada OpenCV *Hue* bernilai 0-179 dikarenakan data yang digunakan untuk pengolahan citra adalah 8 bit.



(a) (b)

Gambar 3.8 (a) Segmentasi dengan toleransi *hue* dengan rentang (-)120 – 150 (b) toleransi *hue* dengan rentang 30 - 360

Saturation merupakan salah satu elemen warna HSV yang mewakili tingkat intensitas warna. Pada nilai tingkat kecerahan (*value*) yang sama nilai *saturation* akan menggambarkan kedekatan suatu warna pada warna abu-abu. Pada sistem nilai saturasi memiliki rentang antara 0 (minimum) dan 1 atau 100% (maksimum). Berikut pada Gambar 3.8 pengaruh nilai toleransi *saturation* pada proses segmentasi. Pada kasus ini akan menggunakan sebuah citra dengan beberapa objek lingkaran yang memiliki tingkat *saturation* berbeda tapi memiliki nilai *hue* dan *value* sama. Berdasarkan contoh kasus tersebut dapat dilihat bahwa nilai toleransi *saturation* mempengaruhi tingkat kemurnian warna yang ikut terseleksi dalam proses segmentasi. Pada OpenCV *Saturation* bernilai 0-255 dikarenakan data yang digunakan untuk pengolahan citra adalah 8 bit. Pada Gambar 3.8 diberikan nilai 70 untuk *saturation*.



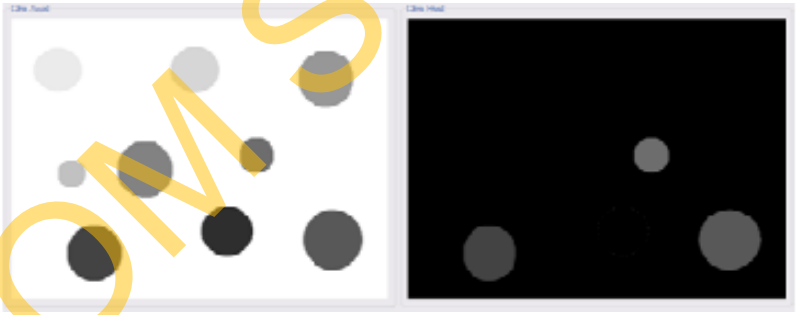
(a) (b)

Gambar 3.9 (a) Gambar asli (b) hasil *tresholding* dengan toleransi *saturation*

Dalam ruang warna HSV, untuk merepresentasikan tingkat kecerahan warna digunakan elemen *value*. Pada nilai *value* maksimum warna yang dihasilkan adalah warna dengan tingkat kecerahan maksimum sedangkan pada *value* minimum dihasilkan warna dengan tingkat kecerahan minimum (warna hitam). Berapapun nilai *hue* dan *saturation* warna, jika nilai *value* yang dimiliki

adalah 0 (minimum) maka warna yang dihasilkan adalah warna hitam. Nilai *value* maksimum adalah 1 (100%), di mana warna yang dihasilkan akan memiliki tingkat kecerahan maksimum.

Nilai toleransi elemen *value* akan mempengaruhi tingkat kecerahan warna objek yang ikut terseleksi dalam proses segmentasi warna. Berikut ini merupakan contoh kasus yang akan menunjukkan bagaimana pengaruh toleransi elemen *value* terhadap hasil segmentasi. Berikut ini Gambar 3.9 terdapat beberapa objek lingkaran dengan nilai *hue, saturation* yang sama tetapi memiliki nilai *value* yang berbeda-beda, dengan demikian terlihat pengaruh toleransi *value* tanpa dipengaruhi elemen warna *hue* dan *saturation*. Pada OpenCV *Value* bernilai 0 sampai 255 dikarenakan data yang digunakan untuk pengolahan citra adalah 8 bit. Pada Gambar 3.9 diberikan nilai 240 untuk *value*.



(a) (b)

Gambar 3.9 (a) Gambar asli (b) hasil *tresholding* dengan toleransi *value*

Berikut pada persamaan 3.1 sampai dengan persamaan 3.3 adalah rumus konversi citra dari ruang warna RGB ke ruang warna HSV secara umum (Kale, 2011):

$$V = \frac{\max(R,G,B)}{255} \quad \text{(3.1)}$$

$$S = \begin{cases} \frac{\max(R,G,B) - \min(R,G,B)}{\max(R,G,B)} & if\ V \neq 0 \\ 0 \end{cases} \quad \text{(3.2)}$$

$$H = \begin{cases} \begin{cases} \frac{60(G-B)}{\max(R,G,B) - \min(R,G,B)} & if\ \max(R,G,B) = R\ and\ G \geq B \\ 300 + \frac{60(B-G)}{\max(R,G,B) - \min(R,G,B)} & if\ \max(R,G,B) = R\ and\ B \geq G \end{cases} \\ 120 + \frac{60(B-R)}{\max(R,G,B) - \min(R,G,B)} \quad if\ \max(R,G,B) = G \\ 240 + \frac{60(R-G)}{\max(R,G,B) - \min(R,G,B)} \quad if\ \max(R,G,B) = B \end{cases} \quad \text{(3.3)}$$

Pada OpenCV untuk mengolah data dari ruang warna HSV, harus dengan ukuran 8 bit per *channel* pada IplImage karena data RGB yang didapat dari *webcam* Robotino adalah 8 bit. Oleh sebab itu diperlukan konversi sesuai dengan yang dibutuhkan OpenCV dengan ketentuan seperti pada persamaan 3.4.

$$V = 255,\ S = 255,\ H = \frac{H}{2(to\ fit\ to\ 255)} \quad \text{(3.4)}$$

Namun pada *library* OpenCV telah disediakan *function* untuk memproses konversi di atas, yaitu dengan menggunakan fungsi *cvCvtColor*. *Function cvCvtColor* adalah fungsi yang disediakan oleh *library* OpenCV, yang digunakan untuk konversi ruang warna RGB (*Red Green Blue*) ke HSV (*Hue Saturation Value*) dan berikut baris perintah yang digunakan.

```
cvCvtColor(Const CvArr *src, Const CvArr *src, int code)
```

Dengan ketentuan parameter *code* yang digunakan adalah CV_BGR2HSV. Berikut baris perintah yang digunakan pada aplikasi untuk mengkonversikan ruang warna RGB ke HSV.

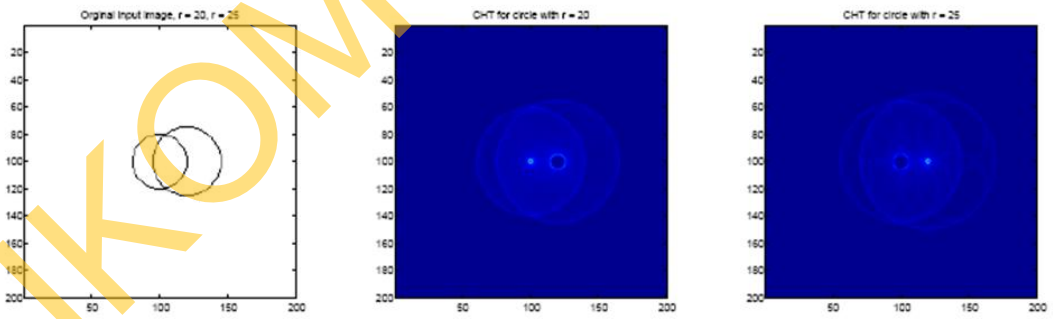
```
cvCvtColor(imgRGB,imgHSV, CV_BGR2HSV);
```

Citra hasil konversi akan disimpan ke dalam data citra imgHSV dan akan digunakan lebih lanjut dalam proses *color filtering*.

3.6.2 *Hough Transform Circle*

Hough Transform adalah teknik transformasi citra yang dapat digunakan untuk mengisolasi atau dengan kata lain memperoleh fitur dari sebuah citra. Karena tujuan dari sebuah transformasi adalah mendapatkan suatu fitur yang lebih spesifik, *Classical Hough Transform* merupakan teknik yang paling umum digunakan untuk mendeteksi objek yang berbentuk kurva seperti garis, lingkaran, elips dan parabola. Keuntungan utama dari *Hough Transform* adalah dapat mendeteksi sebuah tepian dengan celah pada batas fitur dan secara relatif tidak dipengaruhi oleh derau atau noise.

Hough transform circle membentuk lingkaran sepanjang tepian yang ditemukan dengan jari-jari sebesar r. Setelah penggambaran lingkaran sepanjang garis tepian selesai, maka dicari daerah yang paling banyak dilewati garis dan kemudian daerah tersebut diasumsikan sebagai titik tengah citra yang dicari, seperti pada Gambar 4 berikut.



Gambar 4 (a) Citra Input, (b) Pencarian Lingkaran dengan r= 20, dan (c) Pencarian Lingkaran dengan r= 25

Berikut adalah potongan program untuk Hough Transform Circles:

```
CvSeq* circles = cvHoughCircles(thresholded, storage,
CV_HOUGH_GRADIENT, 2, thresholded->height/4, canny, center,
min_radius, max_radius);
```

Dimana *tresholded* adalah tempat penyimpanan data gambar yang akan dideteksi ada tidaknya sebuah lingkaran, sedangkan *storage* berfungsi sebagai tempat array seperti *buffer* untuk menyimpan data ouptut fungsi cvHoughCircles, CV_HOUGH_GRADIENT merupakan mode yang digunakan dalam pendeteksian lingkaran.

3.6.3 *Thresholding*

Untuk melakukan *filter* terhadap warna tertentu, maka data citra dikonversikan ke dalam citra biner dengan memanfaatkan *thresholding*. *Thresholding* adalah proses mengubah suatu citra berwarna atau berderajat keabuan menjadi citra biner atau hitam putih, sehingga dapat diketahui daerah mana yang termasuk objek dan *background* dari citra secara jelas (Gonzales dan Woods, 2002). Citra hasil *thresholding* biasanya digunakan lebih lanjut untuk proses pengenalan obyek serta ekstraksi fitur. Tipe data dari hasil proses *thresholding* adalah tipe data *float*, yaitu antara 0 sampai dengan 1. Dengan parameter yang di set sebelumnya maka data citra yang jika melebihi batas yang ditentukan akan dibuat menjadi 1 atau putih dan jika di bawah batas yang ditentukan maka akan dibuat menjadi 0 atau hitam. Berikut pada persamaan 3.5 adalah rumus *thresholding* yang digunakan.

$$f(x,y) = \begin{cases} 1 \; if \; f(x,y) > T\,\text{min} \;\&\; f(x,y) < T\,max \\ 0 \; if \; f(x,y) \leq T\,\text{min} \;\&\; f(x,y) \geq T\,max \end{cases} \quad \text{.....(3.5)}$$

Pada rumus di atas nilai T min dan T max digunakan sebagai batas *filter* dengan jarak maksimun dan minimum nilai *threshold*. Dimana setiap nilai dari parameter minimum hinggga maksimum akan di isi dengan nilai 1. Sehingga untuk mendapatkan citra biner sesuai *filter*, digunakan ruang warna HSV untuk parameter *filter* sesuai warna yang diharapkan. Ketika nilai T min dan T max di isi dengan batas *filter* parameter HSV yang ditentukan, maka nilai data citra yang berada pada luar rentang filter tersebut akan bernilai 0, sedangkan nilai data citra yang berada pada dalam rentang filter tersebut akan bernilai 1.

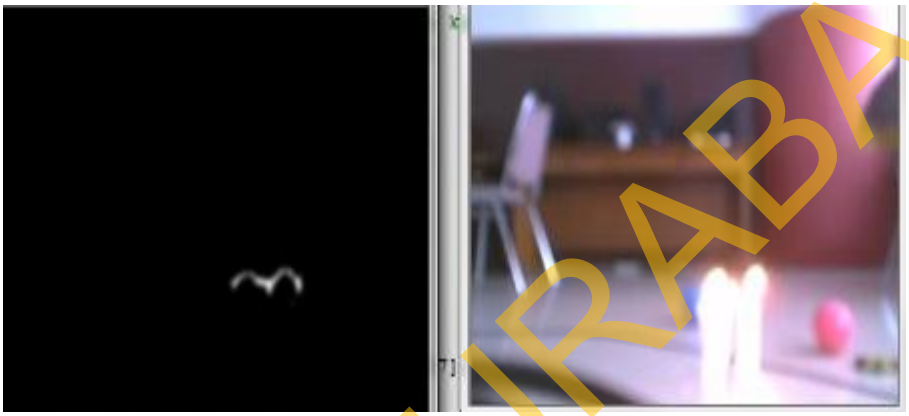
Namun pada *library* OpenCV telah disediakan fungsi untuk memproses *thresholding*, yaitu dengan menggunakan *cvInRangeS*. Dengan ruang warna HSV, maka dapat menggunakan baris perintah berikut.

```
cvInRangeS(hsv, cvScalar(hsv_min, sat_min, val_min),
cvScalar(hsv_max, sat_max, val_max, 0),thresholded);
```

Dengan memberikan nilai batas bawah (minimum) HSV, serta nilai batas atas (maksimum) HSV, maka nilai hasil *threshold* akan didapatkan sesuai rentang tersebut. Jika diprogram secara manual menggunakan rumus *thresholding*, maka dapat ditulis kode program sebagai berikut.

```
int h , s , v;
int x;

x = 0;

if ( 105 < h < 125 )

      if ( 10 < s < 20 )

            if ( 79 < v < 200 )

                        x = 1;

            else
      else
else
```

Seperti Gambar 4.1 hasil *thresholding*, *filter* mendeteksi warna HSV di bawah (96, 2, 254) dan HSV di atas (180, 43, 256) akan membuat citra menjadi warna hitam atau 0. Dan sebaliknya ketika *filter* mendeteksi warna HSV di atas (94, 2, 254) dan HSV di bawah (180, 43, 256) maka citra akan dibuat menjadi putih atau 1.



Gambar 4.1 Filter *Thresholding* HSV

3.6.4 *Smoothing*

Data citra yang telah melalui proses *thresholding* selalu memberikan beberapa *noise*, oleh sebab itu dilakukan proses *smoothing* yang digunakan untuk menghilangkan *noise* pada citra dengan menggunakan *Gaussian filtering* untuk mendapatkan efek *blur* pada citra. Melalui proses *thresholding* maka akan didapatkan citra biner, namun terdapat *noise* yang akan menganggu dalam proses pendeteksian bentuk pada citra. Oleh sebab itu digunakan *smoothing* citra, yang diharapkan akan mengurangi atau menghilangkan *noise*.

Pada *library* OpenCV disediakan *function* untuk proses *smoothing*, *function cvSmooth* adalah *function* yang disediakan oleh *library* OpenCV, yang

digunakan untuk penghalusan citra (*smoothing*) yang bertujuan menghilangkan *noise* citra. Berikut adalah baris perintah untuk menggunakan *cvSmooth.*

```
cvSmooth( const CvArr* src, CvArr* dst int
smoothtype=CV_GAUSSIAN,int param1=3, int param2=0)
```

Dengan ketentuan parameter *smoothtype* yang digunakan adalah CV_GAUSSIAN. Berikut baris perintah yang digunakan pada program.

```
cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
```

Nilai parameter 1 pada fungsi di atas merupakan lebar bukaan untuk nilai pengali dimana nilai tersebut harus bernilai ganjil positif. Untuk parameter 2 merupakan ketinggian bukaan untuk nilai pengali dan harus bernilai ganjil positif. Berikut rumus secara matematika dari perhitungan Gaussian.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

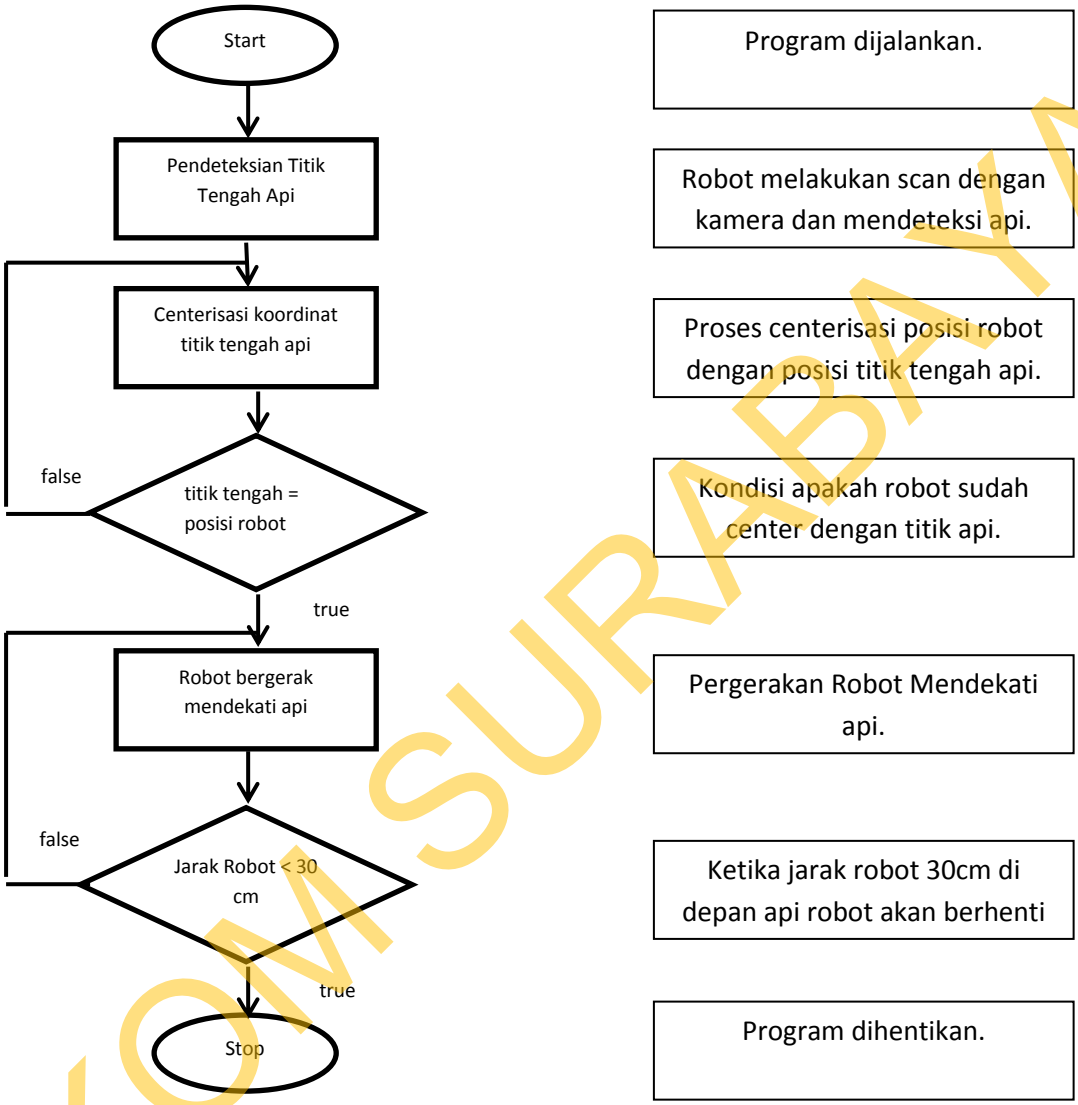
Jika diprogram secara manual, maka penulisan *code* program sesuai rumus perhitungan adalah sebagai berikut.

```
int G[9,9];
int x=0;
int y=0;
int phi = 3,14;
int sd = 1;

G[x,y]=1/2 * phi * sd * sd * (e^((x^2 + y^2) / 2 * sd^2))
```

3.7 *Trajectory Planing*

Robotino bergerak sesuai dengan koordinat yang telah ditentukan menggunakan metode *hough transform circle*, yang mencari titik tengah dari citra api. Robot akan bergerak mengikuti koordinat tersebut kemudian ketika jarak robot berada 20 cm di depan api maka sensor akan mengirimkan data berupa

tegangan keluaran yang akan diterima oleh I/O Robotino, sehingga aktuator berupa alat pemadam api akan memadamkan api tersebut.



Gambar 4.2 flowchart trajectory planing.

Fungsi pergerakan *omnidrive* :

```
void drive(int x, int y, int z)
{
      omniDrive.setVelocity( x , y , z);
}
```

Potongan program utama :

```
if (scan==13)
{
	drive (0,0,30);
	scan=0;
	cout << " //System Run " << endl;
}
```

Potongan program utama di atas merupakan program untuk menjalankan inisialisasi awal pada saat Robotino berputar melakukan manuver 360° mencari lokasi api di sekitarnya. Untuk dapat berputar 360° variable vx diberi nilai 0, vy diberi nilai 0, dan variable omega diberi nilai 30, sehingga robot akan berputar ke kanan. Setelah Robotino mendeteksi adanya api maka robot akan bergerak mendekati api tersebut, berikut adalah potongan program untuk menjalankan robot mendekati api.

```
void grab_fire()
{

	int waktu_sampling;

	eror=160-x;
	delta_eror=eror-eror_1;

	array_koordinat[counter]=x;
	array_eror[counter]=eror;
	array_waktu[counter]=waktu_sampling;

	hasil_perhitungan=perhitungan_eror(eror,delta_eror);

	drive(100,0,hasil_perhitungan);

	cout<< "Fire Detected & Target Locked " << endl << endl;

	eror_1=eror;
	counter++;
}
```

Program di atas merupakan potongan program ketika Robotino mendeteksi api. Metode *Hough Transform Circle* digunakan untuk mencari koordinat titik tengah dari citra api yang dideklarasikan dengan variabel x fungsi koordinat titik tengah adalah untuk menjaga posisi robot tetap berada di jalur yang

benar, ketika posisi robot sudah benar berada di tengah maka robot akan bergerak mendekati api dengan memanfaatkan fungsi *omni-drive* dengan memberi nilai pada variabel vx sebesar 100, variabel vy sebesar 0, dengan begitu robot akan bergerak maju dengan kecepatan sebesar 100 pada sumbu x dan mendekati api, serta kecepatan sudut atau omega sebesar nilai dari variabel hasil_perhitungan, yang merupakan fungsi untuk menentukan sudut belok atau putar robot agar posisi api selalu berada tegak lurus dengan robot.

3.8 *Flame* sensor

Flame sensor disini merupakan alat pendeteksi kedua yang digunakan untuk mendeteksi suhu dari api, setelah pengolahan citra bekerja dan *trajectory planing* dijalankan sensor akan mendeteksi keberadaan api dengan mengeluarkan keluaran berupa tegangan antara 0.5v – 4.85v. Perbedaan tegangan keluaran dipengaruhi oleh jarak sensor dari api, dimana pada jarak yang ditentukan yaitu 20cm maka sensor akan mengeluarkan tegangan keluaran sebesar 4.85v yang akan masuk melalui I/O robotino dan akan berfungsi untuk menggerakan aktuator berupa alat pemadam api.

3.9 Alat Pemadam Api

Alat pemadam api ini digunakan sebagai aktuator dari robot pemadam api. Alat ini terbuat dari rangakaian motor DC, selang, alat penyemprot (*wiper*), dan tabung air yang terbuat dari plastik. Bagian-bagian dari alat pemadam api dapat dilihat pada Gambar 4.2

(a)

(b)

Gambar 4.2 (a) Bagian-bagian alat pemadam api (b) *water tube*

Berikut adalah potongan program yang digunakan untuk aktuator pemadam api :

```
data1=analogInput0.value();

if (data1 > 4)
      relay1.setValue (true);
else
      relay1.setValue (false);
```

Data1 adalah sebuah variabel yang digunakan untuk menyimpan nilai dari tegangan masukan dari sensor api, ketika nilai dari variabel data1 bernilai lebih besar dari 4 maka keluaran dari I/O robotino yang berupa relay akan bernilai *true* ini berarti switch dari relay akan terhubung dan tegangan 12v akan menggerakan motor dari alat pemadam api, sehingga alat pemadam api dapat bekerja sesuai dengan harapan.

3.10 Metode Pengujian dan Evaluasi Sistem

Untuk mengetahui apakah aplikasi yang dibuat dapat berjalan sesuai yang diharapkan, maka akan dilakukan pengujian dan evaluasi sistem untuk setiap tahapan-tahapan dalam pembuatan aplikasi. Dimulai dari *streaming* citra, *color filtering* menggunakan ruang warna HSV, *tresholding*, *smoothing*, *hough*

transform circles, koneksi Robotino, *flame* sensor, alat pemadam api, *trajectory planing*.

3.10.1 Pengujian dan Evaluasi Koneksi Robotino

Pengujian koneksi dilakukan dengan cara melihat apakah Robotino dapat terintegrasi dengan PC, kemudian dilakukan pengujian dengan melihat *access point* dari Robotino apakah sudah terhubung pada Wi-Fi PC. Kemudian setelah terhubung secara *wireless*, langkah berikutnya yaitu menghubungkan koneksi dari *console programming* ke alamat IP yang telah ditentukan pada Robotino. Secara *default* alamat IP yang digunakan adalah 172.26.1.1.

Diharapkan setelah terkoneksi Robotino dapat dikontrol dari PC secara *wireless* dan hal ini dapat dilihat dari pergerakan Robotino setelah penekanan perintah *start* dijalankan.

3.10.2 Pengujian dan Evaluasi Pergerakan Robotino

Untuk pengujian pergerakan Robotino dilakukan dengan cara mengubah parameter pada fungsi setVelocity secara manual, Kemudian dari data tersebut didapatkan arah pergerakan Robotino, dengan memberi nilai pada variabel vx, vy dan kecepatan sudut atau omega maka robot akan melakukan manuver berputar 360° dan melakukan pendeteksian api, setelah proses tersebut *hough transform circles* akan mendeteksi titik tengah dari citra api dan menjalankan *trajectory planing*. Robot akan bergerak menuju lokasi api dan memadamkan api.

Diharapkan setelah melakukan konfigurasi setVelocity dengan memberi nilai pada variabel vx, vy, dan kecepatan sudut maka robot dapat melakukan

manuver berputar 360° sambil mendeteksi api. Setelah medeteksi keberadaan api maka diharapkan prosedur *trajectory planing* akan dijalankan dan robot akan mencari keberadaan api dengan menghitung nilai titik tengah dari citra api dan bergerak menuju api.

3.10.3 Pengujian dan Evaluasi *Streaming* Citra Melalui Kamera Robotino

Untuk mengetahui apakah data citra sudah dapat diakses langsung melalui kamera Robotino, maka dilakukan pengujian dengan cara melihat data yang tampil pada PC apakah sudah sesuai dengan citra yang ditangkap oleh kamera Robotino. Kemudian citra yang tampil akan diuji apakah dapat menampilkan data citra secara *streaming.*

Diharapkan dengan pengujian ini kamera dari robotino dapat menampilkan citra dari lingkungan di sekitar api secara *streaming* dan dapat disimpan kedalam variabel yang dapat diakses dan diprogram secara lebih lanjut.

3.10.4 Pengujian dan Evaluasi Konversi RGB ke HSV

Untuk mengetahui apakah aplikasi dapat mendeteksi warna dengan ruang warna HSV yang sudah ditentukan sebelumnya, maka akan dilakukan pengujian dengan cara melihat tampilan citra setelah dilakukan *thresholding* pada warna HSV tertentu. Jika pada citra ditemukan data citra sesuai dengan filter warn a *thresholding* yang ditentukan sebelumnya, maka pada tampilan citra yang sudah dilakukan *thresholding* akan terlihat pada gambar warna putih.

Diharapkan dengan pengujian ini komponen warna dari RGB dapat diubah menjadi HSV, yang kemudian akan dilakukan proses *tresholding* untuk menjadikan komponen warna menjadi biner yaitu 0 dan 1. Setelah proses *tresholding* sukses maka proses akan dilanjutkan dengan *smoothing* yang digunakan untuk menghilangkan *noise* yang ada pada citra.

1. Tresholding Citra Api

Citra hasil *thresholding* biasanya digunakan lebih lanjut untuk proses pengenalan objek serta ekstraksi fitur. Pengujian thresholding dilakukan dengan kalibrasi HSV berdasarkan nilai masing-masing *hue*, *saturation*, dan *value* sampai menemukan komposisi warna dengan range terbaik untuk objek api. Tipe data dari hasil proses *thresholding* adalah tipe data *float*, yaitu antara 0 sampai dengan 1 dengan parameter yang di set sebelumnya maka data citra yang jika melebihi batas yang ditentukan akan dibuat menjadi 1 atau putih dan jika di bawah batas yang ditentukan maka akan dibuat menjadi 0 atau hitam. Pada pembahasan kali ini *tresholding* digunakan untuk citra api dengan potongan program sebagai berikut:

```
cvInRangeS(hsv, cvScalar(hsv_min, sat_min, val_min)
,cvScalar(hsv_max, sat_max, val_max, 0),thresholded);
```

Pengujian ini dapat dikatakan berhasil apabila filter ini sudah dapat mendeteksi citra api dan membuatnya menjadi citra biner bernilai 1 atau berwarna putih, sedangkan warna latar belakangnya menjadi bernilai 0 atau hitam dan juga dapat membedakan antara gambar api dengan api sungguhan.

2. Smoothing Citra Api

Melalui proses *thresholding* maka akan didapatkan citra biner, namun terdapat *noise* yang akan menganggu dalam proses pendeteksian bentuk pada citra. Oleh sebab itu digunakan *smoothing* citra, yang diharapkan akan mengurangi atau menghilangkan *noise*. Pada pembahasan kali ini *smoothing* digunakan untuk menghilangkan *noise* dari citra api setelah mengalami proses *tresholding*. *Gaussian filtering* diharapkan dapat menghilangkan *noise* dan membuat citra api terlihat lebih halus dengan efek blur. Berikut potongan program dari *Gaussian filtering smoothing*:

```
cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
```

Pengujian ini dapat dikatakan berhasil apabila filter ini sudah dapat menghilangkan *noise* akibat proses *tresholding* dan dapat memberikan efek blur, yang membuat citra terlihat lebih halus.

3.10.5 Pengujian dan Evaluasi *Flame* Sensor

Flame sensor digunakan sebagai alat pendeteksi sekunder dimana sensor ini akan bekerja pada saat robot telah menemukan citra api. Sensor dari DF ROBOT ini dapat mendeteksi api dari jarak 100 cm dengan keluaran tegangan sebesar 0.5v, sedangkan pada jarak 20 cm keluaran tegangan dari sensor akan berubah menjadi 5v. Pengujian pada sensor ini dilakukan dengan ujicoba terpisah terlebih dahulu untuk mengetahui kepekaan sensor terhadap perubahan jarak dengan api.

Pengujian ini dapat dikatakan berhasil apabila sensor dapat menghasilkan keluaran tegangan yang berbeda beda tergantung pada jarak atau kedekatan sensor terhadap api, yang seharusnya sensor akan mengeluarkan tegangan keluaran

sebesar 0.5v ketika berada 100cm di dekat api dan sensor akan mengeluarkan tegangan keluaran sebesar 5v ketika berada 20cm di dekat api.

3.10.6 Pengujian dan Evaluasi *Trajectory Planning*

Trajectory planning adalah sebuah perencanaan pergerakan robot dimana robot akan diprogram untuk berinteraksi dengan lingkungan sekitarnya seperti melakukan *manuver* berputar, bergerak maju, mundur, serta bergerak ke segala arah menggunakan *omnidrive*. Pengujian juga dilakukan untuk mengetahui jarak maksimal robot dalam mendeteksi api.

Pengujian ini dapat dikatakan berhasil apabila robot dapat melakukan pergerakan sesuai harapan, yaitu melakukan manuver berputar pada inisialiasi awal dalam mencari lokasi api, serta bergerak maju mendekati api ketika citra api terdeteksi oleh kamera *webcam* kemudian berhenti ketika jarak robot berada 20cm di depan api dan menjaga agar posisi api tetap berada di tengah.

3.10.7 Pengujian dan Evaluasi Sistem Keseluruhan

Setelah melalui seluruh proses pengujian di atas maka perlu dilakukan pengujian sistem secara keseluruhan. Dimulai dari mengintegrasikan Robotino dengan PC, kemudian menghubungkan koneksi Robotino dengan *console.* Selanjutnya melihat data citra yang ditangkap oleh kamera PC, dan melihat tampilan data citra yang ditampilkan *window image.* Setelah itu, melalui tahap pengolahan citra, yaitu ketika terdeteksi keberadaan api, maka robot akan bergerak mendekati api tersebut. Diantaranya maju, mundur, bergeser ke kiri, bergeser ke kanan, dan berhenti. Kemudian sebagai tambahan, kamera pada

Robotino juga mengirimkan citra yang disorot untuk ditampilkan pada PC secara *streaming*.

Percobaan akan dilakukan beberapa kali dengan variabel kontrol yang sama untuk menguji kinerja dari sistem dan persentase keberhasilan robot dalam mendeteksi keberadaan api serta memadamkan api. Robot akan ditempatkan pada suatu ruangan tertutup dengan pencahayaan tetap dan akan ditempatkan lilin dengan jarak dan posisi yang berbeda beda dari robot, selain itu akan diletakan benda-benda penganggu seperti bola berwarna dan gambar api, kemudian dilakukan pengujian apakah robot dapat mendeteksi api dari lilin tersebut serta memadamkannya dan mengabaikan beberapa objek penganggu seperti gambar api dan bola berwarna. Jika keseluruhan sistem telah berjalan sesuai dengan langkah-langkah tersebut, maka secara keseluruhan sistem ini sudah dikatakan baik.