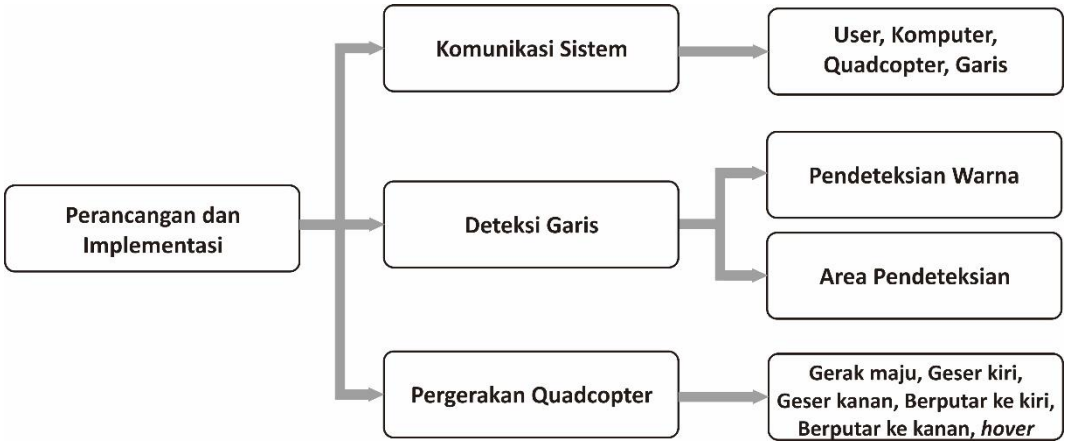
BAB 5 PERANCANGAN DAN IMPLEMENTASI

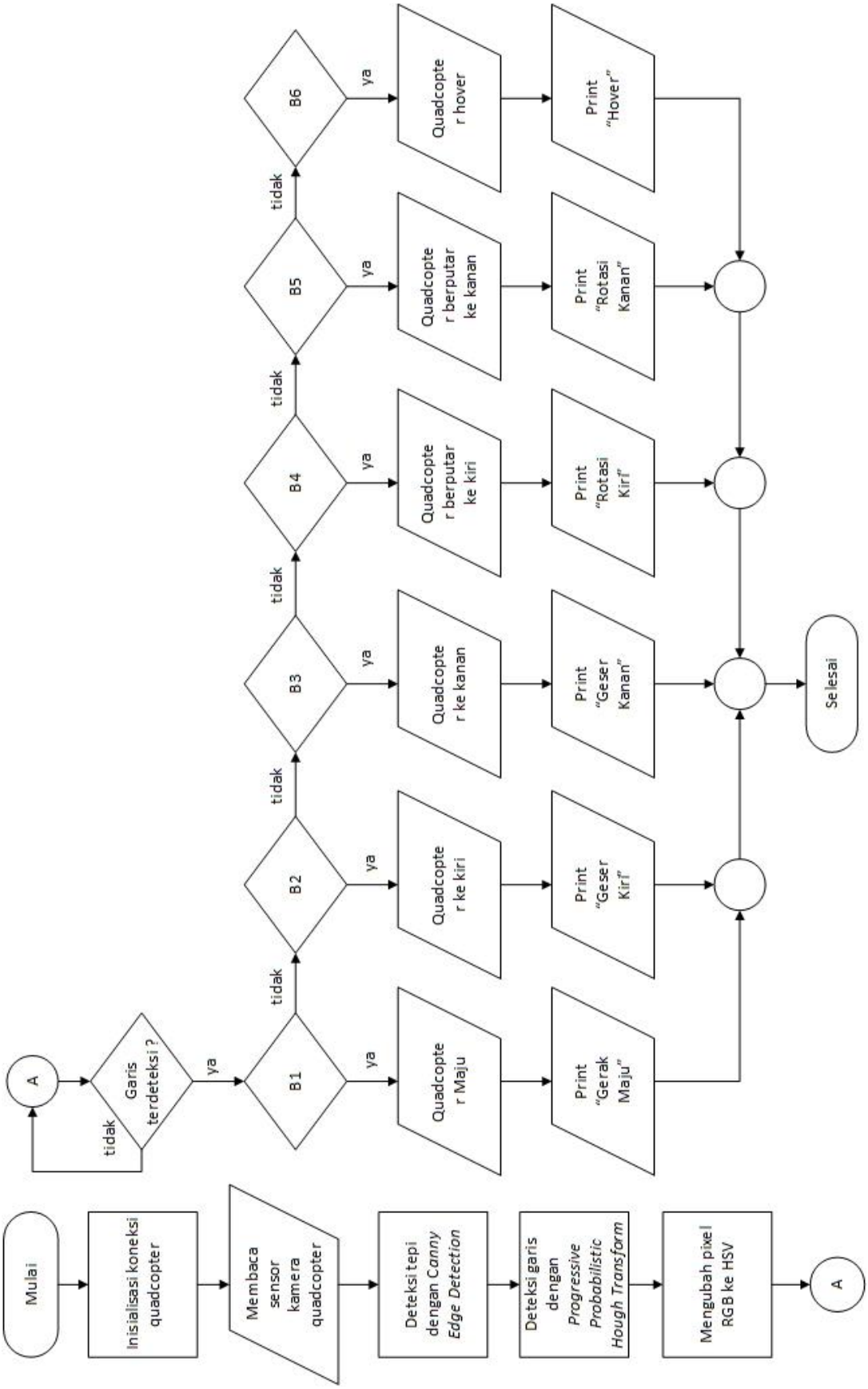
Pada bab ini akan menjelaskan mengenai proses perancangan dan implementasi dari sistem pengikut garis menggunakan *quadcopter*. Dalam perancangan dan implementasi ini akan dilakukan beberapa tahapan seperti pada Gambar 5.1. Pertama dilakukan perancangan komunikasi pada sistem. Pada komunikasi sistem terdapat empat komponen yaitu *user*, komputer, *quadcopter,* dan garis. Setelah itu dilakukan perancangan dan implementasi deteksi garis. Pada tahap deteksi garis dibagi menjadi dua yaitu pendeteksian warna dan area pendeteksian. Lalu tahap terakhir yaitu perancangan dan implementasi pergerakan *quadcopter*, dimana pergerakan ini didapat dari garis yang terdeteksi. Untuk gerakan yang bisa dilakukan oleh *quadcopter* ada lima yaitu gerak maju, geser kiri, geser kanan, berputar ke kiri, berputar ke kanan, dan *hover.*



Gambar 5.1 Tahapan Perancangan dan Implementasi

Setelah tahapan perancangan dan implementasi selesai dilakukan, maka akan dihasilkan hubungan yang akan membangun keseluruhan proses dari sistem. Hubungan tersebut seperti yang tertera pada Tabel 5.1.

Untuk *flowchart* sistem secara umum ditunjukkan pada Gambar 5.2. Pertama akan dilakukan inisialisasi koneksi dari *quadcopter.* Setelah itu akan dilakukan pembacaan sensor kamera *quadcopter*. Kemudian dilakukan proses deteksi tepi dengan menggunakan *canny edge detection*. Setelah mendapatkan tepian selanjutnya dilakukan proses mendeteksi garis menggunakan *progressive probabilistic hough transform*. Kemudian dilakukan pengubahan *pixel* dari *RGB* ke *HSV* untuk mendapatkan warna yang sesungguhnya. Setelah itu akan dilakukan pembacaan apakah garis sudah terdeteksi atau belum. Jika belum terdeteksi maka proses akan terus diulang sampai garis terdeteksi. Saat garis telah terdeteksi maka *quadcopter* bisa mengikuti garis secara otomatis tanpa memberikan *input* pergerakan.



Gambar 5.2 *Flowchart* sistem pengikut garis *quadcopter*

Pada Tabel 5.1 merupakan penjelasan dari kode yang ada pada Gambar 5.2. Sedangkan untuk koordinat yang ada pada *frame* akan dijelaskan pada sub bab 5.3 tentang pergerakan *quadcopter*.

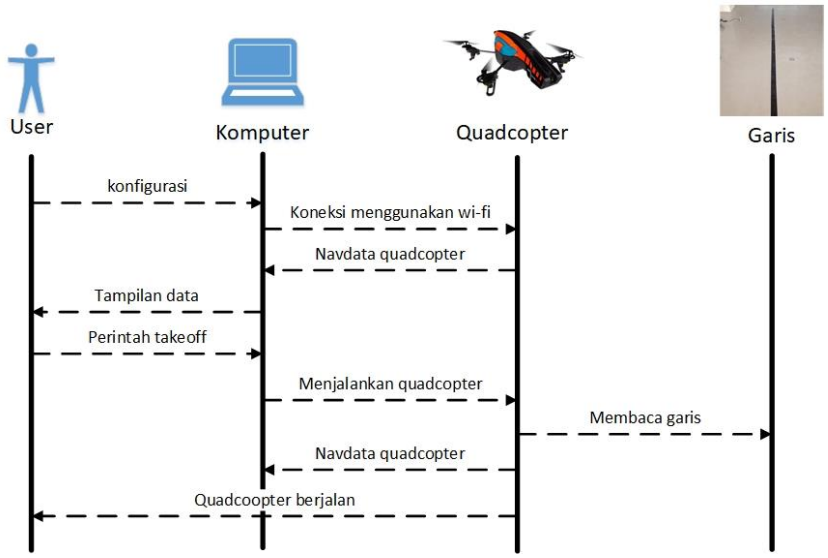
Tabel 5.1 Hubungan antar proses pada sistem

Kode Status	*Frame* Kamera	Gerakan *Quadcopter*
B1	Koordinat B, sumbu x > 110 & x < 220 Koordinat A, sumbu x > 110 & x < 220	Gerak Maju
B2	Koordinat B, sumbu x > 0 & x < 110 Koordinat A, sumbu x > 0 & x < 110	Geser Kiri
B3	Koordinat B, sumbu x > 220 & x < 330 Koordinat A, sumbu x > 220 & x < 330	Geser Kanan
B4	Koordinat B, sumbu x > 110 & x < 220 Koordinat A, sumbu x > 0 & x < 110	Berputar ke Kiri
B5	Koordinat B, sumbu x > 110 & x < 220 Koordinat A, sumbu x > 220 & x < 330	Berputar ke Kanan
B6	Koordinat B, sumbu x = 0 Koordinat A, sumbu x = 0	*Hover*

5.1 Komunikasi Sistem

5.1.1 Perancangan Komunikasi Sistem

Pada sistem ini, keseluruhan data yang diproses dijelaskan seperti pada Gambar 5.3. Pertama pengguna melakukan konfigurasi dikomputer dengan menjalankan program. Kemudian menghubungkan komputer dengan *quadcopter* menggunakan *Wi-Fi*. Setelah itu *quadcopter* mengirimkan *navdata* berupa data kamera dan data navigasi ke komputer. Komputer menampilkan data yang telah dikirimkan dari *quadcopter* sehingga terlihat oleh pengguna. Selanjutnya pengguna melakukan perintah *takeoff* ke komputer dan *quadcopter* mulai terbang. Kemudian *quadcopter* membaca garis menggunakan kamera yang berada dibawah. Setelah mendapatkan data kamera kemudian *quadcopter* mengirim *navdata* lagi ke komputer. Setelah itu pengguna bisa melihat *quadcopter* berjalan secara otomatis mengikuti garis.



Gambar 5.3 Alur pertukaran data pada sistem

5.1.2 Implementasi Komunikasi Sistem

Pada bagian ini dijelaskan bagaimana alur implementasi komunikasi sistem pada penelitian ini. Implementasi komunikasi pada sistem ini ditunjukkan pada Gambar 5.4. Komputer terhubung dengan *quadcopter* menggunakan komunikasi *Wi-Fi*. Kemudian *quadcopter* membaca garis menggunakan sensor kamera, sensor kamera yang dipakai adalah kamera bawah.

Gambar 5.4 Implementasi komunikasi sistem

5.2 Deteksi Garis

5.2.1 Perancangan Deteksi Garis

Dalam perancangan deteksi garis yang akan digunakan sebagai *input* dari sistem ini, diperlukan data kamera dari *quadcopter*. Data tersebut diolah dengan menggunakan *ROS* dan *OpenCV*. Setelah didapatkan data kemudian dilakukan beberapa proses seperti pada Gambar 5.5.

Seperti pada Gambar 5.5 data kamera *quadcopter* diolah dengan beberapa metode pengolahan citra diantaranya mendeteksi tepian garis menggunakan *canny edge detection*. Dengan menggunakan *canny edge detection* bisa meminimalkan jarak antara tepi yang dideteksi dengan tepi yang asli dan mudah dideteksi sehingga tidak menimbulkan kerancuan pada pengolahan citra selanjutnya. Proses selanjutnya yaitu mendeteksi garis dengan menggunakan *progressive probabilistic hough transform*. Pada proses ini merupakan peningkatan dari proses sebelumnya. Setelah mendeteksi garis kemudian mengubah warna *pixel* dari *RGB* ke *HSV*. Dengan melakukan pengubahan warna ke *HSV* maka akan didapatkan warna yang sesungguhnya. Setelah itu dilakukan pembacaan garis berdasarkan koordinat garis berada. Setelah semua proses selesai dilakukan maka hasil yang didapatkan akan ditampilkan dalam sebuah *frame*.

Gambar 5.5 Alur pendeteksian garis

5.2.1.1 Pendeteksian Warna

Dalam penelitian ini warna yang dideteksi adalah warna hitam, tidak ada warna yang lain. Warna hitam jika dijabarkan dalam bentuk *RGB* akan memiliki nilai *Red* = 0, *Green* = 0 dan *Blue* = 0. Tetapi dalam penelitian ini warna hitam akan diubah dari *RGB* ke *HSV*. Jika dalam bentuk *HSV* juga akan memiliki nilai 0 ditiap

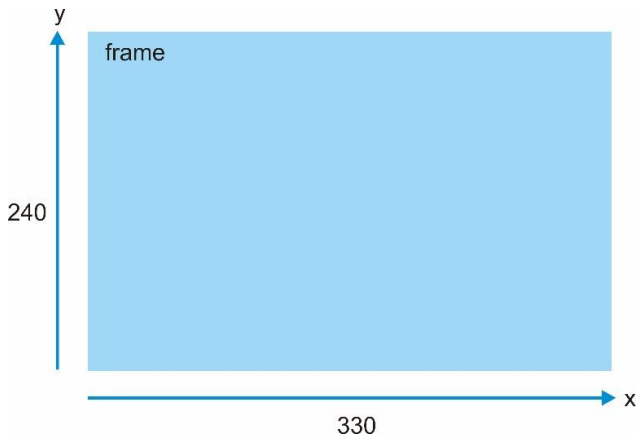
parameternya sehingga harus dibuat nilai minimal dan nilai maksimal dari warna hitam dalam bentuk *HSV*. Untuk mendapatkan nilai minimal dan maksimal tidak ada referensi pasti untuk mendapatkan nilai-nilai tersebut. Sehingga dalam penelitian ini dilakukan pengujian agar mendapatkan nilai minimal dan maksimal untuk warna hitam yang sesuai. Untuk hasil pengujian dibahas secara lengkap pada sub bab 6.1.

Berdasarkan hasil pengujian pada sub bab 6.1 didapatkan nilai *HSV* yang bisa digunakan untuk mendeteksi garis secara penuh yaitu:

1. *Hue*
   Dalam parameter nilai *Hue* didapatkan nilai minimal = 0 dan nilai maksimal = 180.
2. *Saturation*
   Dalam parameter nilai *Saturation* didapatkan nilai minimal = 0 dan nilai maksimal = 255.
3. *Value*
   Dalam parameter nilai *Value* didapatkan nilai minimal = 0 dan nilai maksimal = 10.

5.2.1.2 Area Pendeteksian

Dalam mendeteksi garis maka dibuat sebuah *frame.* Untuk pembuatan *frame* dilakukan beberapa pengujian agar mendapatkan *frame* yang akurat untuk mendeteksi garis. Pembuatan *frame* ini dibahas secara lengkap pada sub bab 6.2. Berdasarkan pengujian yang telah dilakukan didapatkan ukuran *frame* yang akurat untuk mendeteksi garis yaitu ukuran 330*240 *pixel.* Ukuran ini paling pas karena proses yang dilakukan dalam mendeteksi garis cukup cepat dan garis terdeteksi secara penuh. Untuk area pendeteksian bisa digambarkan seperti pada Gambar 5.6.



Gambar 5.6 Area pendeteksian

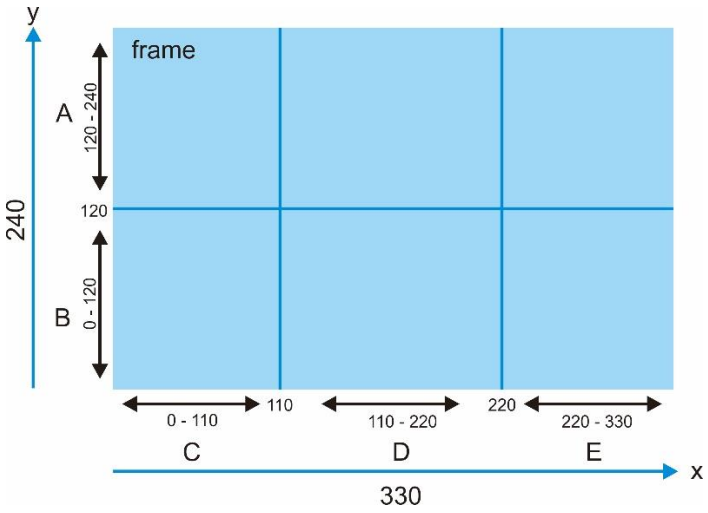
Frame ini nantinya digunakan untuk menampilkan hasil kamera dari *quadcopter* dan juga untuk fungsi pendeteksian. Dengan menggunakan ukuran *frame* 330*240 *pixel* dapat mempermudah dalam proses mendeteki garis dikarenakan *frame* akan dibagi lagi menjadi beberapa bagian. Dalam pembagian

frame menggunakan Persamaan 5.1 untuk pembagian pada sumbu x dan Persamaan 5.2 untuk pembagian pada sumbu y.

$$Pembagian\ sumbu\ x = \frac{sumbu\ x}{3} \quad (5.1)$$

$$Pembagian\ sumbu\ y = \frac{sumbu\ y}{2} \quad (5.2)$$

Berdasarkan persamaan 5.1 pada sumbu x akan dibagi menjadi 3 bagian, masing-masing bagian mempunyai ukuran 110 *pixel*. Sedangkan berdasarkan persamaan 5.2 pada sumbu y akan dibagi menjadi 2 bagian, masing-masing bagian mempunyai ukuran 120 *pixel*. Sehingga pada sumbu x dibagi dengan *range* 0-110 (C), *range* 110-220 (D), dan *range* 220-330 (E). Sedangkan pada sumbu y dibagi dengan *range* 0-120 (B) dan *range* 120-240 (A). Untuk pembagian lebih jelaskan bisa dilihat pada Gambar 5.7.



Gambar 5.7 Area pendeteksian dengan pembagian letak

5.2.2 Implementasi Deteksi Garis

Implementasi pendeteksi garis diawali dengan pengambilan data dari kamera *quadcopter*. Seperti pada *source code* baris ke 2 dilakukan pemanggilan data dari kamera *quadcopter* menggunakan ardrone/image_raw.

Kode Program 5.1 Implementasi kamera *quadcopter*

```
1  image_transport::ImageTransport it(node);
2  image_transport::Subscriber it_sub =
   it.subscribe("ardrone/image_raw", 1, chatterCallback);
```

Setelah selesai mengambil data dari kamera kemudian melakukan beberapa proses seperti berikut ini :

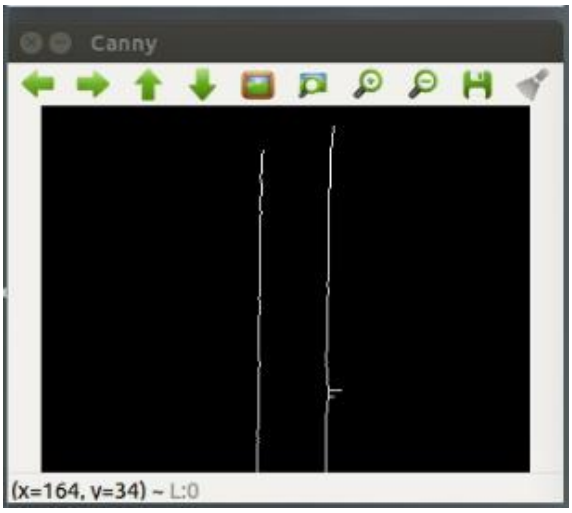
1. Mendeteksi tepian dengan *canny edge detection*

   Input dari pendeteksian ini menggunakan frame dan *output* menggunakan dst. Nilai minimal untuk *threshold* adalah 130 dan maksimal adalah 250 seperti terlihat pada Kode Program 5.2 baris ke 13.

Kode Program 5.2 Deteksi *canny edge detection*

```
1   Mat frame, frame1, frame2, roiFrame1, roiFrame2;
2   Mat hsv, gray,threshold;
3
4   vector<vector<Point> > contours;
5   vector<Vec4i> hierarchy;
6
7
8   cv_bridge::CvImagePtr bridge;
9   bridge = cv_bridge::toCvCopy(newimage,
    sensor_msgs::image_encodings::BGR8);
10  frame1 = bridge->image;
11  resize(frame1,frame,Size(320,240));
12  Mat dst;
13  Canny(frame, dst, 130, 250, 3);
```

Implementasi dari pendeteksian garis tepi dengan menggunakan *canny edge detection* bisa dilihat pada Gambar 5.8. Terlihat tepian dari garis terdeteksi membentuk sebuah garis lurus.



Gambar 5.8 Implementasi Deteksi dengan *Canny Edge detection*

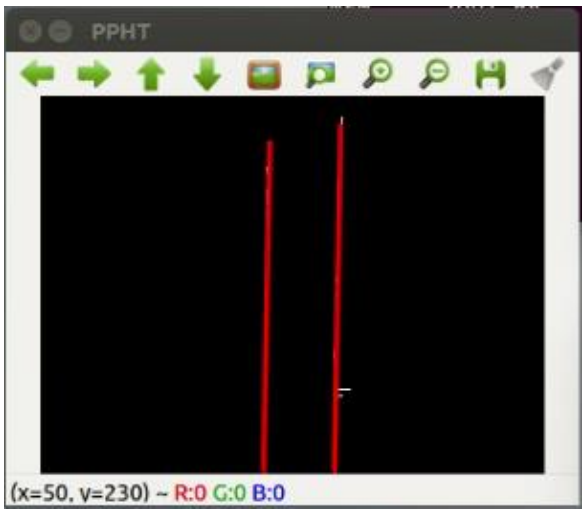
2. Mendeteksi garis dengan *progressive probabilistic hough transform*

   Input dari pendeteksian ini menggunakan hasil dari pendeteksian sebelumnya yaitu dst. Nilai minimal *threshold* adalah 130, nilai minimal dari *point* yang dapat mendeteksi garis adalah 150 dan maksimal celah diantara 2 *point* adalah 80. Nilai-nilai ini bisa dilihat pada Kode Program 5.3 baris ke 3.

Kode Program 5.3 Deteksi *progressive probabilistic hough transform*

```
1   vector<Vec4i> lines;
2   // detect the lines
3   HoughLinesP(dst, lines, 1, CV_PI/180, 130, 150, 80 );
4   for( size_t i = 0; i < lines.size(); i++ )
5   {
6      Vec4i l = lines[i];
7      // draw the lines
8      line( frame, Point(l[0], l[1]), Point(l[2], l[3]),
    Scalar(0,0,255), 2, CV_AA);
9      line( cdst, Point(l[0], l[1]), Point(l[2], l[3]),
    Scalar(0,0,255), 2, CV_AA);}
```

Implementasi dari pendeteksian garis tepi dengan menggunakan *progressive probabilistic hough transform* bisa dilihat pada Gambar 5.9. Terlihat tepian dari garis terdeteksi membentuk sebuah garis lurus berwarna merah.



Gambar 5.9 Implementasi deteksi dengan *progressive probabilistic hough transform*

3. Mengubah *pixel RGB* ke *HSV*

   Input yang diubah ke *HSV* adalah roi1 dan roi2 kemudian roiFrame1 dan roiFrame2 sebagai *output*.

Kode Program 5.4 Pengubahan pixel

```
1  cvtColor(roi1, roiFrame1, CV_BGR2HSV );
2  cvtColor(roi2, roiFrame2, CV_BGR2HSV );
```

Nilai *HSV* yang digunakan untuk mendeteksi warna hitam. Nilai pada Kode Program 5.5 didapatkan dari hasil pengujian yang sudah dijelaskan pada sub bab 6.1.

Kode Program 5.5 Parameter nilai HSV

```
1  int iLowH = 0,   iHighH = 180,   iLowS = 0,   iHighS = 255,   iLowV
   = 0,  iHighV = 10;
```

4. Mendeteksi garis dengan mencari kontur garis

   Pada pendeteksian mencari kontur dilakukan dengan 2 proses karena *frame* yang digunakan untuk mendeteksi dibagi menjadi 2. Baris ke 2 dan 18 *findcountours* digunakan untuk mendapatkan kontur pada warna *HSV*. Baris ke 5 dan 21 adalah nilai *pixel* dari kontur garis. Pada penelitian ini *pixel* yang digunakan adalah antara 1000 sampai 5000.

Kode Program 5.6 Deteksi kontur garis

```
1  //--------------------------------ROI ATAS-----------------------
   ------------------//
2
```

```
    findContours(roiFrame1.clone(),        contours,        hierarchy,
3   CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE, Point(0,0));
4   for (unsigned int A = 0; A < contours.size(); A++) {
5               float konturGaris = contourArea(contours[A]);
6               if (konturGaris > 1000 && konturGaris < 5000) {
7                   Moments mm = moments(contours[A], false);
8                   double moment10 = mm.m10;
9                   double moment01 = mm.m01;
10                  konturAtas = mm.m00;
11                  CoordA.x = int(moment10 / konturAtas);
12                  CoordA.y = int(moment01 / konturAtas);
                    Point2f centerAtas(CoordA.x, CoordA.y);  // lokasi
13  titik centroid Atas
                    circle(frame,centerAtas,5,Scalar(0,   0,   255),-
14  1,8,0);
15              }
16  }
17
    //--------------------------------ROI BAWAH-----------------------
18  -------------------//
    findContours(roiFrame2.clone(),        contours,        hierarchy,
19  CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE, Point(0,0));
20  for (unsigned int B = 0; B < contours.size(); B++) {
21              float konturGaris = contourArea(contours[B]);
22              if (konturGaris > 1000 && konturGaris < 5000) {
23                  Moments mm = moments(contours[B], false);
24                  double moment10 = mm.m10;
25                  double moment01 = mm.m01;
26                  konturBawah = mm.m00;
27                  CoordB.x = int(moment10 / konturBawah);
28                  CoordB.y = int(moment01 / konturBawah);
                    Point2f centerBawah(CoordB.x, CoordB.y+120);   //
29  lokasi titik centroid Bawah
                    circle(frame,centerBawah,5,Scalar(0,   0,   255),-
30  1,8,0);
31              }
    }
```

Untuk pembagian area pendeteksian seperti pada Kode Program 5.7 berikut ini.

Kode Program 5.7 Pembagian area deteksi

```
1   //----------------------------vertical-----------------------------
    ------//
2       line( frame, Point( 110,0 ), Point( 110,240), Scalar( 255, 0,
    0),  1, 8 );
3       line( frame, Point( 220,0 ), Point( 220,240), Scalar( 255, 0,
    0),  1, 8 );
4       //---------------------------horizontal-----------------------
    ----------//
5       line( frame, Point( 0,120 ), Point( 330,120), Scalar( 255, 0,
    0),  1, 8 );
6       //----------------------Centroid2Centroid--------------------
    ----------//
7       line( frame, Point(CoordA.x, CoordA.y), Point(CoordB.x,
    CoordB.y+120), Scalar(0, 0, 255),1,8);
```

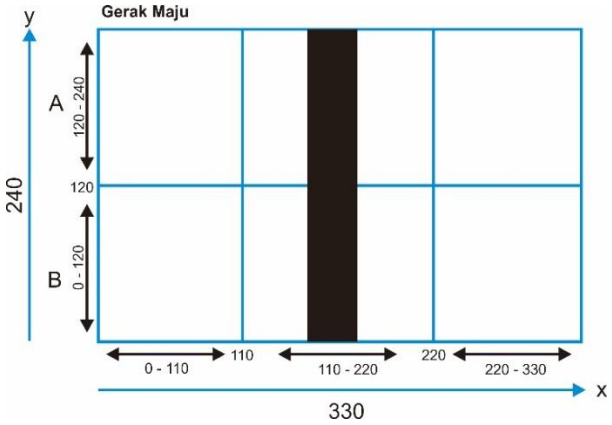
5.3 Pergerakan *Quadcopter*

5.3.1 Perancangan Pergerakan *Quadcopter*

Dalam perancangan gerakan *quadcopter* digunakan pembacaan pendeteksian garis sebagai *input* dari sistem ini, diperlukan data berupa letak garis pada *frame*. Berikut ini pergerakan *quadcopter* secara otomatis berdasarkan pendeteksian garis.

1. Gerak maju

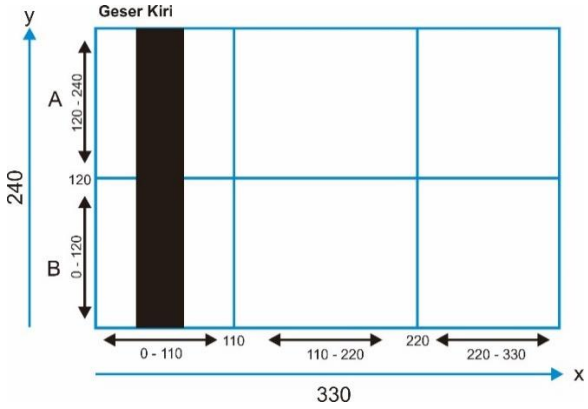
   Q*uadcopter* akan bergerak maju ketika garis yang terdeteksi berada pada bagian A dengan sumbu x ukuran 110-220 dan sumbu y dengan ukuran 120-240. Sedangkan bagian B dengan sumbu x pada ukuran 110-220 dan sumbu y dengan ukuran 0-120. Letak garis saat gerak maju dapat dilihat seperti pada Gambar 5.10.



Gambar 5.10 Area pendeteksian gerak maju

2. Geser Kiri

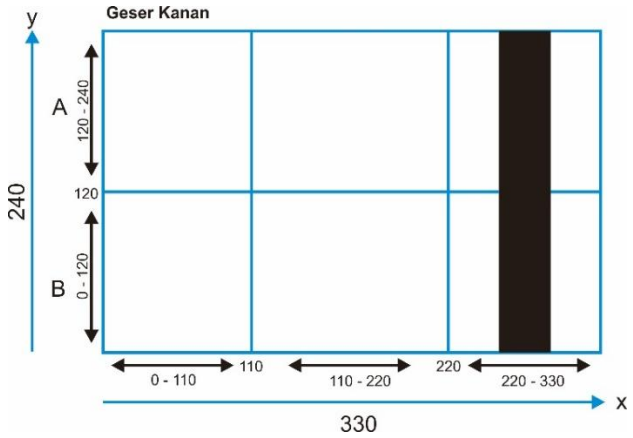
   Q*uadcopter* akan bergeser ke kiri ketika garis yang terdeteksi berada pada bagian A dengan sumbu x ukuran 0-110 dan sumbu y dengan ukuran 120-240. Sedangkan bagian B dengan sumbu x pada ukuran 0-110 dan sumbu y dengan ukuran 0-120. Letak garis saat bergeser ke kiri dapat dilihat seperti pada Gambar 5.11.



Gambar 5.11 Area pendeteksian geser kiri

3. Geser Kanan

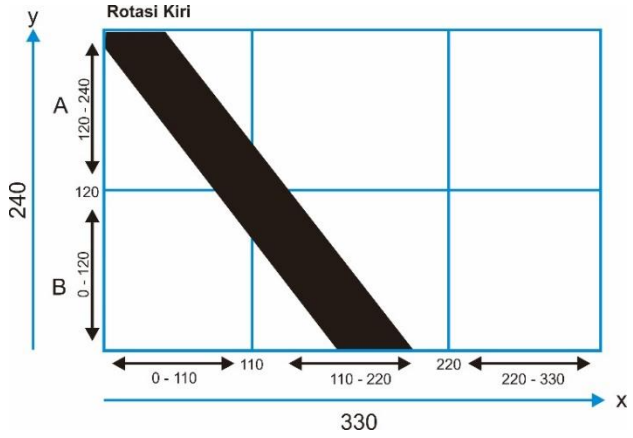
   Q*uadcopter* akan bergeser ke kanan ketika garis yang terdeteksi berada pada bagian A dengan sumbu x ukuran 220-320 dan sumbu y dengan ukuran 120-240. Sedangkan bagian B dengan sumbu x pada ukuran 220-320 dan sumbu y dengan ukuran 0-120. Letak garis saat bergeser ke kiri dapat dilihat seperti pada Gambar 5.12.



Gambar 5.12 Area pendeteksian geser kanan

4. Berputar ke Kiri

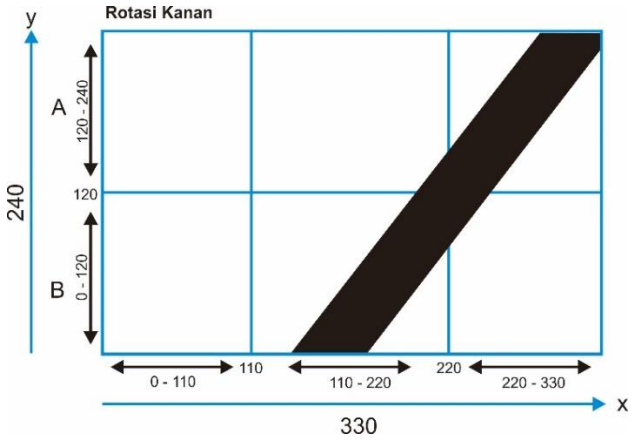
   Q*uadcopter* akan berputar ke kiri ketika garis yang terdeteksi berada pada bagian A dengan sumbu x ukuran 0-110 dan sumbu y dengan ukuran 120-240. Sedangkan bagian B dengan sumbu x pada ukuran 110-220 dan sumbu y dengan ukuran 0-120. Letak garis saat berputar ke kiri dapat dilihat seperti pada Gambar 5.13.



Gambar 5.13 Area pendeteksian rotasi kiri

5. Berputar ke Kanan

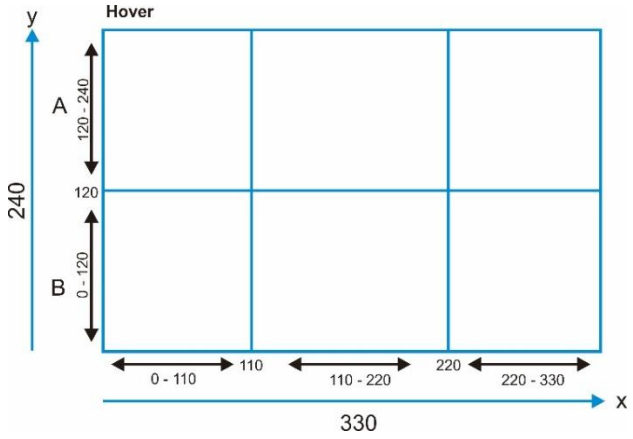
   Q*uadcopter* akan berputar ke kanan ketika garis yang terdeteksi berada pada bagian A dengan sumbu x ukuran 220-320 dan sumbu y dengan ukuran 120-240. Sedangkan bagian B dengan sumbu x pada ukuran 110-220 dan sumbu y dengan ukuran 0-120. Letak garis saat berputar ke kanan dapat dilihat seperti pada Gambar 5.14.



Gambar 5.14 Area pendeteksian rotasi kanan

6. *Hover*

   Q*uadcopter* akan *hover* atau tidak bergerak ketika tidak ada garis yang terdeteksi baik itu pada bagian A maupun bagian B. Letak garis saat *hover* dapat dilihat seperti pada Gambar 5.15.



Gambar 5.15 Area pendeteksian hover

Selain gerakan *quadcopter* secara otomatis mengikuti garis dengan mendeteksi letak, ada juga gerakan yang dilakukan secara manual seperti *takeoff* dan *landing.*

5.3.2 Implementasi Pergerakan *Quadcopter*

Setelah membuat perancangan pergerakan, selanjutnya dibuat kode program untuk masing-masing gerakan sesuai dengan letak garis yang terdeteksi. Adapun kode program untuk tiap gerakan sebagai berikut.

1. Gerak Maju

   Inisialisasi gerak maju, *quadcopter* bergerak maju secara linear pada sumbu x.

Kode Program 5.8 Inisialisasi gerak maju

```
1   //maju message----------------
2         maju.linear.x=0.1;
3         maju.linear.y=0.0;
```

```
4          maju.linear.z=0.0;
5          maju.angular.z=0.0;
```

Inisialisasi untuk menampilkan “Gerak Maju” di*terminal* dan memanggil fungsi maju.

Kode Program 5.9 Fungsi gerak maju

```
1   case 'w':
2          cout<<"Gerak Maju"<<endl;
3         pub_twist.publish(maju);
4         command=' ';
5         break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.10 Koordinat garis saat gerak maju

```
1   if (CoordB.x>110 && CoordB.x<220 && CoordA.x>110 && CoordA.x<220 )
    {
2            // 0 1 0
3            // 0 1 0
4            char Areas[20];
5            sprintf(Areas,"Gerak maju");
6            putText(frame,Areas,Point(10,40),  FONT_HERSHEY_PLAIN,  1,
    Scalar(255,0,0),2);
7            command = 'w';
8   }
```

2. Geser Kiri

Inisialisasi *source code* geser kiri, *quadcopter* bergeser ke kiri secara linear pada sumbu y.

Kode Program 5.11 Inisialisasi geser kiri

```
1   //geser kiri message
2                  geserki.linear.x=0.0;
3                  geserki.linear.y=+0.01;
4                  geserki.linear.z=0.0;
5                  geserki.angular.z=0.0;
```

Inisialisasi untuk menampilkan “Geser Kiri” di*terminal* dan memanggil fungsi geserki.

Kode Program 5.12 Fungsi geser kiri

```
1   case 'a':
2              cout<<"Geser Kiri"<<endl;
3              pub_twist.publish(geserki);
4              command = ' ';
5              break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.13 Koordinat garis saat geser kiri

```
1   else if (CoordB.x>0 && CoordB.x<110 && CoordA.x>0 && CoordA.x<110)
    {
2           // 1 0 0
3           // 1 0 0
4           char Areas[20];
5           sprintf(Areas,"Geser kiri");
6           putText(frame,Areas,Point(10,40), FONT_HERSHEY_PLAIN, 1,
    Scalar(255,0,0),2);
7           command = 'a';
8   }
```

3. Geser Kanan

   Inisialisasi *source code* geser kanan, *quadcopter* bergeser ke kiri secara linear pada sumbu y.

Kode Program 5.14 Inisialisasi geser kanan

```
1   //geser kanan message
2                   geserka.linear.x=0.0;
3                   geserka.linear.y=-0.01;
4                   geserka.linear.z=0.0;
5                   geserka.angular.z=0.0;
```

Inisialisasi untuk menampilkan “Geser Kanan” di*terminal* dan memanggil fungsi geserka.

Kode Program 5.15 Fungsi geser kanan

```
1   case 'd':
2               cout<<"Geser Kanan"<<endl;
3               pub_twist.publish(geserka);
4               command = ' ';
5               break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.16 Koordinat garis saat geser kanan

```
1   else if (CoordB.x>220 && CoordB.x<330 && CoordA.x>220 &&
    CoordA.x<330) {
2           // 0 0 1
3           // 0 0 1
4           char Areas[20];
5           sprintf(Areas,"Geser kanan");
6           putText(frame,Areas,Point(10,40), FONT_HERSHEY_PLAIN, 1,
    Scalar(255,0,0),2);
7           command = 'd';
8   }
```

4. Rotasi Kiri

   Inisialisasi *source code* rotasi kiri, *quadcopter* berputar ke kiri secara angular pada sumbu z.

Kode Program 5.17 Inisialisasi rotasi kiri

```
1  //rotasi kiri message
2                  rotasiki.linear.x=0.0;
3                  rotasiki.linear.y=0.0;
4                  rotasiki.linear.z=0.0;
5                  rotasiki.angular.z=+0.8;
```

Inisialisasi untuk menampilkan “Rotasi Kiri” di*terminal* dan memanggil fungsi rotasiki.

Kode Program 5.18 Fungsi rotasi kiri

```
1  case 'q':
2              cout<<"Rotasi Kiri"<<endl;
3              pub_twist.publish(rotasiki);
4              command = ' ';
5              break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.19 Koordinat garis saat rotasi kiri

```
1  else if (CoordB.x>110 && CoordB.x<220 && CoordA.x>0 && CoordA.x<120
   ) {
2           // 1 0 0
3           // 0 1 0
4           char Areas[20];
5           sprintf(Areas,"Rotasi kiri");
6           putText(frame,Areas,Point(10,40),  FONT_HERSHEY_PLAIN,  1,
   Scalar(255,0,0),2);
7           command = 'q';
8  }
```

5. Rotasi Kanan

Inisialisasi *source code* rotasi kanan, *quadcopter* berputar ke kanan secara angular pada sumbu z.

Kode Program 5.20 Inisialisasi rotasi kanan

```
1  //rotasi kanan message
2                  rotasika.linear.x=0.0;
3                  rotasika.linear.y=0.0;
4                  rotasika.linear.z=0.0;
5                   rotasika.angular.z=-0.8;
```

Inisialisasi untuk menampilkan “Rotasi Kanan” di*terminal* dan memanggil fungsi rotasika.

Kode Program 5.21 Fungsi rotasi kanan

```
1  case 'e':
2              cout<<"Rotasi Kanan"<<endl;
3              pub_twist.publish(rotasika);
4              command = ' ';
5              break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.22 Koordinat garis saat rotasi kanan

```
else if (CoordB.x>110 && CoordB.x<220 && CoordA.x>200 && CoordA.x<330 ) {
        // 0 0 1
        // 0 1 0
        char Areas[20];
        sprintf(Areas,"Rotasi kanan");
        putText(frame,Areas,Point(10,40), FONT_HERSHEY_PLAIN, 1, Scalar(255,0,0),2);
        command = 'e';
}
```

6. *Hover*

Inisialisasi *source code hover*, quadcopter tidak bergerak sehingga tidak terjadi perubahan pada program.

Kode Program 5.23 Inisialisasi hover

```
//hover message
                hover.linear.x=0.0;
                hover.linear.y=0.0;
                hover.linear.z=0.0;
                hover.angular.z=0.0;
```

Inisialisasi untuk menampilkan “Hovering” di*terminal* dan memanggil fungsi hover.

Kode Program 5.24 Fungsi hover

```
case 'h':
            cout<<"Hovering"<<endl;
            pub_twist.publish(hover);
            command = ' ';
            break;
```

Inisialisasi koordinat garis yang terdeteksi.

Kode Program 5.25 Koordinat garis saat hover

```
else if(CoordA.x==0 || CoordB.x==0){
        // 0 0 0
        // 0 0 0
        char Areas[20];
        sprintf(Areas,"Hover");
        putText(frame,Areas,Point(10,40), FONT_HERSHEY_PLAIN, 1, Scalar(255,0,0),2);
        command = 'h';
}
```

Berikut ini gerakan yang dilakukan secara manual dengan menekan tombol pada *keyboard* untuk mengerakkan *quadcopter* :

1. T*akeoff*

Inisialisasi fungsi *takeoff* dengan memanggil *library* Ardrone Autonomy.

Kode Program 5.26 Inisialisasi *takeoff*

```
1  ros::Publisher pub_empty_takeoff;
2  pub_empty_takeoff =
   node.advertise<std_msgs::Empty>("/ardrone/takeoff", 1);
```

Inisialisasi untuk menampilkan “Taking Off” di*terminal* dan memanggil fungsi *takeoff* dan *hover*.

Kode Program 5.27 Fungsi *takeoff*

```
1  case 't':
2              pub_empty_takeoff.publish(msg); //launches the drone
3              pub_twist.publish(hover); //drone is flat
4              cout<<"Taking Off"<<endl;
5              command=' ';
6              break;
```

2. *Landing*

Inisialisasi fungsi *landing* dengan memanggil *library* Ardrone Autonomy.

Kode Program 5.28 Inisialisasi *landing*

```
1  ros::Publisher pub_empty_takeoff;
2  pub_empty_takeoff =
   node.advertise<std_msgs::Empty>("/ardrone/takeoff", 1);
```

Inisialisasi untuk menampilkan “Landing” di*terminal* dan memanggil fungsi *landing* dan *hover*.

Kode Program 5.29 Fungsi *landing*

```
1  case 'l':
2              pub_twist.publish(hover); //drone is flat
3              pub_empty_land.publish(msg); //lands the drone
4              cout<<"Landing"<<endl;
5              exit(0);
6              command = ' ';
7              break;
```