# Pendeteksian Gawang Menggunakan Algoritma *Ransac* pada *Platform Darwin-OP* Berbasis Peraturan KRSBI 2016

Uti Solichah, Nanik Suciati, Muhtadin
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
*e-mail*: nanik@if.its.ac.id

*Abstrak*— ***Dalam strategi sepak bola robot, robot dituntut untuk bergerak secara autonomous, untuk itu kemampuan dasar yang harus dimiliki setiap robot adalah dapat mengenali fitur penanda dalam lapangan, seperti bola, gawang dan garis lapangan. Kolaborasi pendeteksian gawang dengan informasi orientasi dapat membantu robot dalam memposisikan dirinya (x, y,θ), dan dapat membantu robot untuk memutuskan arah tendangan bola, apakah mengarah ke gawang lawan atau sebaliknya. Hough Transform merupakan metode pendeteksian gawang yang sudah dikembangkan oleh tim robot ITS, namun metode ini belum optimal dalam segi kecepatan dan ketepatan jarak. Algoritma Ransac merupakan Algoritma deteksi garis yang memproses semua point untuk dijadikan sebagai garis. Untuk meningkatkan performa robot dalam hal ketepatan jarak, pada Tugas Akhir ini Algoritma Ransac dipilih sebagai metode lain yang akan diterapkan pada pendeteksian gawang. Hasil uji coba menunjukkan bahwa robot dapat mendeteksi gawang dengan lingkungan pertandingannya. Rata-rata galat pengujian estimasi jarak robot pada gawang dengan Algoritma Ransac lebih kecil dibandingkan dengan Metode Hough Transform, dengan nilai rata-rata galat 0,03062 meter untuk Algoritma Ransac dan 0,1452 meter untuk Metode Hough Transform. Sedangkan pengujian estimasi waktu program menunjukkan bahwa Metode Hough Transform lebih cepat dibandingkan dengan Algoritma Ransac dengan mencapai rata-rata waktu eksekusi 0,0333 detik untuk Metode Hough Transform dan 0,0435 detik untuk Algoritma Ransac.***

***Kata Kunci*— *Algoritma Ransac, Autonomous, Gawang, Hough Transform, Pendeteksian Gawang, Algoritma Ransac.***

## I. PENDAHULUAN

DALAM strategi sepak bola robot, robot dituntut untuk bergerak secara *autonomous,* untuk itu kemampuan dasar yang harus dimiliki setiap robot adalah dapat mengenali fitur penanda dalam lapangan, seperti bola, gawang dan garis lapangan. Pengenalan fitur gawang ditambah dengan informasi orientasi dapat membantu robot dalam memposisikan dirinya *(x,y,θ)* [1]. Dengan mengetahui posisi dirinya, robot dapat memutuskan arah tendangan bola, apakah mengarah ke gawang lawan atau menjauhkan bola dari gawang sendiri [2].

Salah satu metode yang sudah dikembangkan dalam pendeteksian gawang adalah dengan menggunakan metode *Hough Transform* [3]. Pendeteksian gawang dengan menggunakan Metode *Hough Transform* telah sukses diaplikasikan pada Tim Robot Sepak Bola ITS, namun Metode ini belum optimal dalam segi kecepatan pendeteksian dan ketepatan jarak gawang dengan robot. Sehingga, dibutuhkan pendekatan metode lain untuk meningkatkan performa kemampuan pendeteksian gawang pada Tim Robot Sepak Bola ITS.

Algoritma *Ransac* merupakan Algoritma deteksi garis yang memproses semua *point* untuk dijadikan sebagai garis. Untuk meningkatkan performa robot dalam hal ketepatan jarak, pada Tugas Akhir ini Algoritma *Ransac* dipilih sebagai metode lain yang akan diterapkan pada pendeteksian gawang, Selain alasan diatas, beberapa penelitian dibawah ini juga mendasarkan Algoritma *Ransac* dipilih sebagai metode pendeteksian gawang pada Tugas Akhir, pertama oleh Stephen [4], yang menyebutkan bahwa pendekatan Algoritma *Ransac* lebih efektif daripada pendekatan *Hough Transform*. Kedua, penelitian yang dilakukan oleh Madison [5] yang mengaplikasikan Algoritma *Ransac* pada robot Darwin-OP untuk pendeteksian gawang. Diharapkan dengan pengaplikasian Algoritma *Ransac*, performa pendeteksian gawang yang dilakukan oleh Tim Robot Sepak Bola ITS akan lebih baik, dan lebih efektif dalam segi kecepatan pendeteksian dan akurasi jarak gawang dengan robot. Tahapan sistem pendeteksian gawang menggunakan Algoritma *Ransac* meliputi proses *prapocessing*, deteksi garis menggunakan Algoritma *Ransac*, dan terakhir *postprocessing.*

## II. PENDETEKSIAN GAWANG MENGGUNAKAN ALGORITMA *RANSAC*

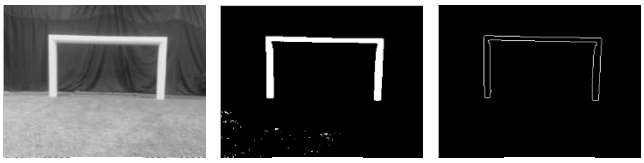
### *A. Preprocessing*

Setiap piksel pada citra masukan tidak bisa langsung dilakukan deteksi garis menggunakan Algoritma *Ransac*, ada beberapa tahapan yang harus dilakukan diawal, untuk mempersiapkan citra lebih mudah di proses di tahap selanjutnya, berikut adalah lima tahap yang harus dilakukan.

Tahap Pertama, yaitu mengubah citra masukan ke citra *grayscale* citra, Pengubahan citra *grayscale* memudahkan citra masukkan untuk di *thresholding* dimana objek yang diambil pada *thresholding* adalah warna putih dari gawang. Tahap kedua adalah *thresholding* citra, dimana pada *thresholding*

membantu citra masukan untuk memisahkan objek gawang yang dipilih dengan objek selain gawang.

Ketiga, reduksi *noise* citra, hasil keluaran citra dari proses *thresholding* terkadang tidak selalu ideal, untuk membantu citra mendapatkan hasil segmentasi yang ideal, citra hasil tadi dilanjutkan pada tahap reduksi *noise*. *Noise* yang ada pada citra dihilangkan menggunakan pendekatan metode *morphology opening*. Keempat, tahap *preprocessing* selanjutnya setelah reduksi *noise* citra adalah mendeteksi tepi citra gawang. Hal ini dilakukan untuk mengurangi garis yang akan terdeteksi oleh Algoritma *Ransac* Ilustrasi hasil citra *grayscale*, citra *thresholding* dan citra hasil deteksi tepi gawang ditunjukkan pada Gambar 1. Dan tahap *preprocessing* yang terakhir adalah transformasi citra ke *point*. Proses transformasi ini akan menghasilkan *point-point* pada suatu citra yang nantinya disimpan kemudian di proses selanjutnya oleh Algoritma *Ransac.*



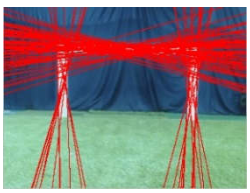
(a) (b) (c)

Gambar 1. Citra *Grayscale* (a), Citra *Thresholding* (b) dan Citra Deteksi Tepi (c)

### *B. Deteksi Garis Menggunakan Algoritma Ransac*

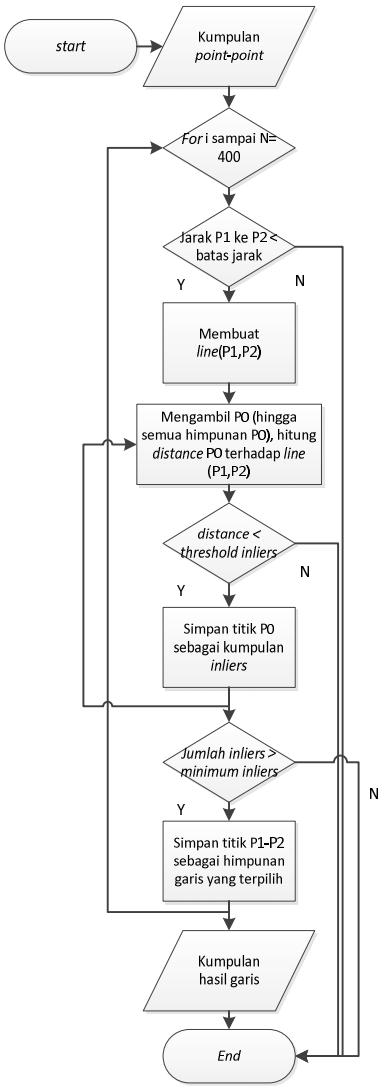
Hasil *binary* tepi citra yang sudah di transformasikan menjadi *point* kemudian di proses menggunakan Algoritma *Ransac* untuk didapatkan garis-garis yang terdeteksi. Himpunan *point* pada citra dilanjutkan dengan mengambil dua *point random*, kemudian, apabila jarak antara satu *point* dengan *point*-nya terlalu dekat, maka dua *point* random tadi tidak dipilih, Sebaliknya, jika dua *point* random diatas memenuhi jarak minimum antara dua *point,* maka selanjutnya adalah mengambil satu *point* kemudian dihitung jarak antara *point* ke garis tadi.

Apabila jarak point tersebut memenuhi *threshold* jarak *inliers*, maka *point* tadi termasuk *inliers* dari garis tersebut [6]. Setelah semua himpunan *point* sudah di cek, selanjutnya adalah mengecek apakah banyak nya *point inliers* tadi memenuhi minimum *inliers* sebuah garis, apabila tidak memenuhi, maka di hapus, jika memenuhi di masukkan ke dalam tampungan garis yang terdeteksi. Proses pengambilan dua *point* random hingga berhasil menjadi sebuah garis merupakan satu kesatuan proses, untuk mendapatkan banyak garis, maka dilakukan perulangan, hingga didapatkan banyak garis. Ilustrasi deteksi garis menggunakan Algoritma *Ransac* ditunjukkan pada Gambar 2.



Gambar 2. Citra Hasil Deteksi Garis Algoritma *Ransac*

Diagram alur dari proses deteksi garis Algoritma *Ransac* ditunjukkan pada Gambar 3.




Gambar 3. Diagram Alur Deteksi Garis Algoritma *Ransac*

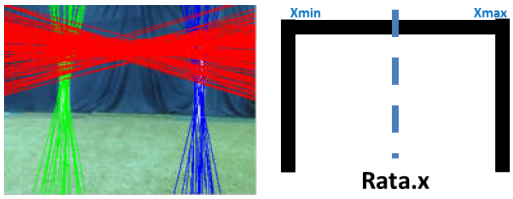
### *C. Postprocessing*

Deteksi garis oleh Algoritma *Ransac* menghasilkan kumpulan garis-garis. Untuk mendapatkan sebuah informasi mengenai objek gawang atau bukan, informasi tinggi tiang, dan juga informasi jarak dari sebuah garis yang terdeteksi, maka diperlukan tahapan *postprocessing* yang meliputi sebagai berikut:

1. *Clustering Line*

Mengetahui *ymin, ymax, xmin, xmax* dari *binary object* gawang adalah langkah pertama untuk menentukan *cluster* tiang. Penentuan *ymin, ymax, xmin, xmax* dilakukan perulangan dari citra koordinat *x* dan perulangan dari citra koordinat *x*. Setelah mendapatkan *ymin, ymax, xmin*, dan *xmax*. Selanjutnya adalah menghitung tengah-tengah koordinat *x* dari objek gawang. Tengah-tengah dari koordinat *x* atau rata *x* objek gawang digunakan sebagai acuan, apakah garis itu termasuk tiang kanan, tiang kiri atau tiang atas. Jika garis itu lebih besar dari tengah koordinat, maka garis tersebut

termasuk *cluster* tiang kanan. Jika garis itu lebih kecil dari tengah koordinat x, maka garis itu termasuk *cluster* tiang kiri, sisanya yang tidak termasuk keduanya, termasuk *cluster* tiang atas. Ilustrasi hasil cluster dari garis *clustering line* dan *ilustrasi cluster* bisa dilihat pada Gambar 4.

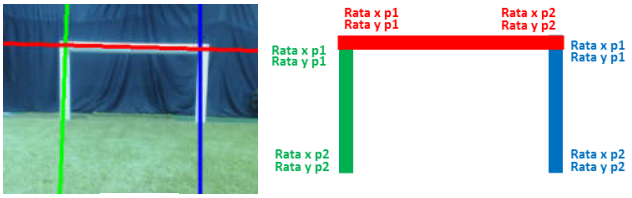



(a) (b)

Gambar 4. Citra Hasil *Clustering Line* (a) dan Ilustrasi *Cluster* (b)

2. Hitung Mean Tiang tiap *Cluster*

Hal yang pertama dilakukan adalah dengan cara menjumlahkan koordinat *X* dari $P_1$ tiang kanan kemudian dibagi banyaknya garis, begitu juga dengan menghitung mean koordinat *Y* dari $P_1$ dari menjumlahkan koordinat *Y* dari $P_1$ tiang kanan kemudian dibagi banyaknya garis. Setelah itu, menghitung *mean* tiang *cluster* kanan titik $P_2$ juga sama dengan seperti mencari mean *X* dari $P_1$, dan mean *Y* dari $P_1$. Rata-rata *X* dan mencari rata-rata Y dari titik $P_2$. Pencarian mean koordinat *X* dan *Y* dari $P_1$ dan $P_2$ dilakukan di tiap *cluster* kanan, *cluster* atas dan *cluster* kiri. Ilustrasi hasil hitung mean dan ilustrasi mean dapat dilihat pada Gambar 5.




(a) (b)

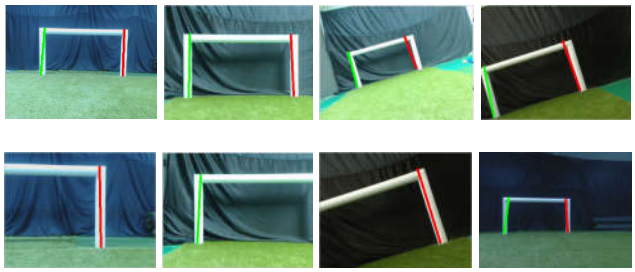
Gambar 5. Citra Hasil Hitung Mean (a) dan Ilustrasi Hitung Mean (b)

3. *Drawing Line*

Hasil yang didapatkan dari menentukkan *mean* tiang tiap *cluster* tidak dapat mengetahui tinggi piksel dari objek gawang yang terdeteksi, untuk itu diperlukan sebuah proses unuk menggambarkan *line* terbaik dari tiap-tiap *cluster*.Proses *drawing line* terbaik dimulai dari mengetahui *X* dari garis terbaik tersebut. Setelah mendapatkannya, maka di cek pada citra *thresholding* yang sudah ada, apakah garis terbaik itu pada keadaan nilai RGB = 255, jika iya termasuk warna putih kemudian disimpan dan dilakukan perulangan sampai citra hasil *thresholding* tidak memiliki warna RGB = 255. Ilustrasi berbagai hasil *drawing line* dapat dilihat pada Gambar 6.

4. Hitung Tinggi Tiang

Pada proses *drawing line,* didapatkan juga dua *point*, berupa batas atas dan batas bawah dari tiap tiang kanan dan juga tiang kiri. Dalam menghitung tinggi tiang, dari *best line* tersebut bisa dihitung dengan cara menghitung jarak antara dua titik. Persamaan untuk menghitung jarak antara dua titik tiang kanan menggunakan rumus menghitung menggunakan *eucledian distance*.



Gambar 6. Berbagai Citra Hasil *Drawing Line*

5. Hitung Jarak Tiang ke Robot

Proses terakhir dalam tahap *postprocessing* adalah menghitung jarak tiang ke Robot. Penghitungan estimasi jarak robot terhadap tiang gawang diperoleh berdasarkan tinggi tiang gawang dalam piksel dan jarak sebenarnya. Pada penelitian sebelumnya [7], persamaan ideal yang digunakan untuk mendapatkan jarak sebenarnya digunakan pendekatan regresi polynomial. Regeresi Polynomial orde ke 5 adalah orde yang paling cocok dari data yang diberikan. Sehingga, didapatkan persamaan ideal untuk perhitungan estimasi jarak robot dengan gawang dan ditunjukkan pada persamaan 1.

$$Jarak = (5{,}04 \times 10^{-10})X^5 - (2{,}94 \times 10^{-7})X^4 + (6{,}63 \times 10^{-5})X^3 - (6{,}60 \times 10^{-3})X^2 + 0{,}25948X + 2{,}0383 \quad (1)$$

## III. HASIL DAN EVALUASI

### A. *Pengujian Parameter*

1. Parameter *Thresholding*

Dari hasil uji yang dilakukan, pada tahap *thresholding,* parameter yang paling baik adalah dengan batas bawah 140 dan batas atas 255. Parameter yang baik pada setiap citra akan berubah, sangat bergantung dengan intensitas citra inputan. Untuk itu, jika metode ini diterapkan pada robot, diperlukan data kalibrasi warna objek untuk membantu segmentasi warna *thresholding* citra [1]. Pada Tabel 1 diperlihatkan beberapa perbandingan parameter batas atas pada citra Gambar 1.

Tabel 1. Parameter *Thresholding* Citra pada Gambar 1.

| No | Nama Gambar | Batas Bawah | Batas Atas | Keterangan |
|---|---|---|---|---|
| 1 | Gambar 1 | 20 | 255 | Terlalu banyak *noise* |
| 2 | Gambar 1 | 40 | 255 | Banyak *noise* |
| 3 | Gambar 1 | 60 | 255 | Banyak *noise* |
| 4 | Gambar 1 | 80 | 255 | Banyak *noise* |
| 7 | Gambar 1 | 100 | 255 | Banyak *noise* |
| 8 | Gambar 1 | 120 | 255 | Masih ada *noise* |
| 9 | Gambar 1 | 130 | 255 | *Noise* Citra sedikit |
| 10 | Gambar 1 | 140 | 255 | Tidak ada *noise* |
| 11 | Gambar 1 | 160 | 255 | Citra terdegradasi |
| 12 | Gambar 1 | 170 | 255 | Hanya ada tiang atas |
| 13 | Gambar 1 | 200 | 255 | *Thresholding* sedikit |

2. Parameter Perulangan *(N)*

Pada tahap deteksi garis menggunakan Algoritma *Ransac,* ada beberapa parameter yang di set, diantaranya paramater *N*. Parameter *N* digunakan untuk merepresentasikan jumlah perulangan untuk mendapatkan garis yang akan dibentuk sesuai dengan Algoritma *Ransac.* Pada Tabel 2 diperlihatkan beberapa perbandingan parameter *N*. Pemilihan parameter *N* sebesar 400 didasari oleh hasil keluaran tinggi tiang dan hasil *drawing line* yang tidak terlalu jauh, artinya dengan parameter *N* sebesar 400 sudah mewakili himpunan garis untuk proses deteksi garisnya.

Tabel 2. Parameter Perulangan *(N)* Minimum Garis

| No | Besarnya *N* | Keterangan |
|---|---|---|
| 1 | 100 | Terlalu sedikit garis yang terbuat |
| 2 | 200 | Garis yang terbuat pada tiang kanan dan tiang kiri terlalu sedikit |
| 3 | 300 | Garis yang terbuat belum ideal |
| 4 | 400 | Garis yang terbuat sudah ideal |
| 5 | 1000 | Terlalu banyak garis yang terbuat |

*3.* Parameter *Threshold Inliers*

Parameter pada deteksi garis yang di set adalah parameter *threshold inliers*, parameter ini merepresentasikan syarat jarak *point* $P_0$ yang dipilih dengan *line* $P_1$, $P_2$ yang sudah dibuat. Apabila jarak $p_0$ ke *line* memenuhi, maka *point* $P_0$ termasuk *inliers*, sedangkan jika tidak memenuhi, *point* $P_0$ tidak dimasukkan kedalam *inliers*. Pada Tabel 3 diperlihatkan beberapa perbandingan parameter *threshold inliers*. Dimana, dengan parameter *threshold inliers* 2, garis yang dihasilkan masih terlalu sedikit, sedangkan dengan parameter *threshold inliers* sebesar 5, garis yang dihasilkan ideal. Untuk itu, *threshold inliers* terbaik yang diambil adalah dengan nilai 5.

Tabel 3. Parameter Minimum *Threshold Inliers*

| No | *Threshold inliers* | Keterangan |
|---|---|---|
| 1 | 2 | Sedikit garis yang terbentuk |
| 2 | 5 | Garis yang terbentuk ideal |
| 3 | 10 | Terlalu banyak garis yang terbentuk |

4. Parameter Batas Jarak

parameter selanjutnya yang di set adalah parameter *batas jarak*. *Batas jarak* pada Algoritma *Ransac* merepresentasikan jarak minimum yang digunakan sebagai persyaratan jarak antara *point* $P_1$ dan *point* $P_2$. Parameter batas jarak terbaik adalah 40, ketika jarak *point* $P_1$ dengan *point* $P_2$ lebih dari 40, maka tidak akan dibuat *line* $P_1$, $P_2$. Sedangkan, jika jaraknya memenuhi, maka *point* $P_1$ dan $P_2$ akan di proses menjadi *line.* Pada Tabel 4 diperlihatkan beberapa perbandingan parameter batas jarak yang sudah dilakukan.

Tabel 4. Parameter Minimum Batas Jarak $P_1$ dengan $P_2$

| No | Batas jarak | Keterangan |
|---|---|---|
| 1 | 10 | Terlalu banyak garis yang dibuat |
| 2 | 20 | Terlalu banyak garis yang dibuat |
| 3 | 40 | Garis yang dibuat ideal |
| 4 | 70 | Sedikit garis yang dibuat |
| 5 | 100 | Terlalu sedikit garis yang dibuat |

*5.* Paramater Minimum *Inliers*

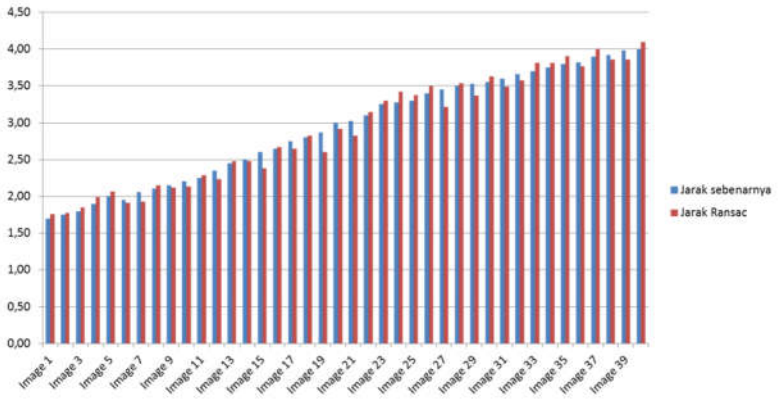
Parameter terakhir yang diatur pada Algoritma *Ransac* adalah *minimum inliers. Minimum inliers* merepresentasikan jumlah *minimum inliers point* yang terbentuk untuk menjadi satu garis. Apabila jumlah *inliers* yang terkumpul untuk membuat garis kurang, maka garis dari *point* $P_1$ dan $P_2$ tidak akan dibuat dan ditampung pada variabel *best line*. Sebaliknya, jika sebuah garis memenuhi *minimum inliers*-nya, maka garis dari *point* $P_1$ dan $P_2$ dimasukkan pada tampungan *best line*. Pada Tabel 5 diperlihatkan beberapa perbandingan parameter *minimum inliers.* Dimana, dengan parameter *minimum inliers* = 10, garis yang dibuat terlalu banyak, sedangkan dengan *minimum inliers* sebesar 85, garis yang terbuat ideal.

Tabel 5. Parameter Minimum *Inliers* Garis

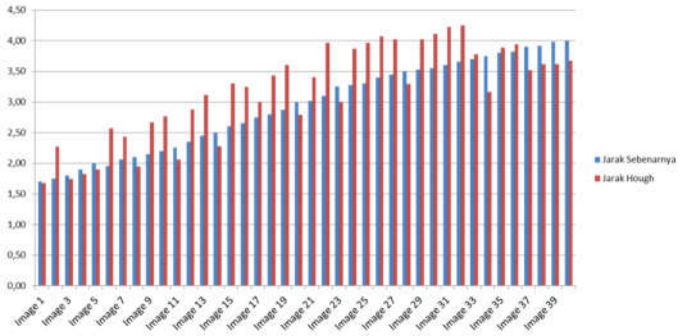
| No | *Minimum inliers* | Keterangan |
|---|---|---|
| 1 | 20 | Terlalu banyak garis yang terbuat |
| 2 | 30 | Banyak garis yang terbuat |
| 3 | 40 | Banyak garis yang terbuat |
| 4 | 60 | Cukup banyak garis yang terbuat |
| 5 | 85 | Garis yang terbuat ideal |
| 6 | 100 | Garis yang terbuat sedikit |
| 7 | 200 | Garis terbuat hanya di tiang atas saja |
| 8 | 1000 | Tidak ada garis yang terbuat |

*B. Pengujian Ketepatan Jarak*

Dari hasil uji yang dilakukan, diperlihatkan ada 40 dataset citra yang masing-masing diproses yang satu menggunakan Algoritma *Ransac* yang satu menggunakan metode *Hough Transform*. Hal yang bisa dianalisa adalah jarak yang didapat dari tiap-tiap metode. Pada pengujian ini, robot ditempatkan di posisi yang sama, dengan keadaan yang sama, kemudian sama-sama di proses dengan metode masing-masing. Dengan skenario uji seperti itu, bisa dibandingkan galat jarak gawang yang terdeteksi antar metode. Jarak gawang didapat dari tinggi piksel yang sudah dijelaskan di bab tiga buku ini. Berdasarkan rata-rata galat yang didapat. Gambar 6 memperlihatkan jarak yang terdeteksi menggunakan Algoritma *Ransac* dan jarak sebenarnya. Gambar 7 memperlihatkan jarak yang terdeteksi menggunakan Metode *Hough Transform* dan jarak sebenarnya




Gambar 6. Perbandingan Jarak Sebenarnya dengan Jarak yang dideteksi Algoritma *Ransac*

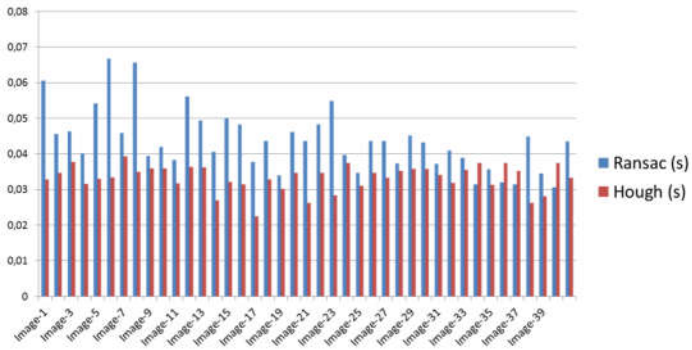



Gambar 7. Perbandingan Jarak Sebenarnya dengan Jarak yang dideteksi Metode *Hough Transform*

Berdasarkan rata-rata galat yang didapat, Algoritma *Ransac* memiliki keakuratan jarak robot dengan gawang yang lebih baik daripada Metode *Hough Transform*, dengan rata-rata galat jarak 0,03062 meter untuk Algoritma *Ransac* dan rata-rata galat jarak sebesar 0,1452 meter untuk Metode *Hough Transform.*

### *C. Pengujian Kecepatan Waktu Eksekusi Program*

Dari hasil uji yang dilakukan, pada Gambar 8. diperlihatkan perbedaan waktu eksekusi yang dilakukan pada sekian dataset, dimana masing-masing dataset diproses dengan Algoritma *Ransac* dan Metode *Hough Transform*. Batasan program perbandingan hanya meliputi *preprocessing* sampai deteksi garis saja. Berdasarkan hasil rata-rata waktu eksekusi yang didapat pada setiap metode. *Hough Transform* lebih cepat dalam memproses algoritmanya untuk pendeteksian garis gawang dibandingkan dengan metode Algoritma *Ransac*, dengan rata-rata waktu eksekusi program 0,0333 detik untuk *Hough Transform* dan rata-rata waktu eksekusi program 0,0435 detik untuk Algoritma *Ransac.*




Gambar 8. Perbandingan Waktu Eksekusi Program antara Algoritma *Ransac* dan Metode *Hough Transform*

## IV. KESIMPULAN

Kesimpulan berdasarkan hasil uji coba parameter didapatkan bahwa pada Tabel 1, parameter *thresholding* terbaik untuk Gambar 1 adalah 140, namun itu berlaku untuk satu citra saja. Parameter *thresholding* setiap inputan bisa berbeda, dan bergantung pada kondisi citra masukan. Berdasarkan Tabel 2, parameter perulangan *(N)* terbaik adalah 400. Berdasarkan Tabel 3, parameter *batas jarak* terbaik adalah 40. Berdasarkan Tabel 4, parameter *threshold inliers* terbaik adalah 5. Berdasarkan Tabel 5, parameter *minimum inliers* terbaik adalah 85. Berdasarkan Gambar 6 dan 7 dapat diambil kesimpulan bahwa rata-rata galat pengujian estimasi jarak robot pada gawang dengan Algoritma *Ransac* lebih kecil dibandingkan dengan Metode *Hough Transform*, dengan nilai rata-rata galat 0,03062 meter untuk Algoritma *Ransac* dan 0,1452 meter untuk Metode *Hough Transform.* Dan, berdasarkan Gambar 8, dapat diambil kesimpulan bahwa pengujian estimasi waktu program Metode *Hough Transform* lebih cepat dibandingkan dengan Algoritma *Ransac* dengan mencapai rata-rata waktu eksekusi 0,0333 detik untuk Metode *Hough Transform* dan 0,0435 detik untuk Algoritma *Ransac.*

## UCAPAN TERIMA KASIH

Penulis U.S. mengucapkan terima kasih sebesar-besarnya kepada Tim Robot Institut Teknologi Sepuluh Nopember, khususnya kepada Tim Robot Sepak Bola ITS, ICHIRO yang telah membantu dan memfasilitasi penulis dalam pengujian penilitian yang menggunakan Robot Darwin-OP.

## DAFTAR PUSTAKA

[1] Budiono, I. (2015). Kerjasama dalam permainan sepak bola robot pada platform Darwin-OP berbasis peraturan KRSBI 2015. Surabaya.
[2] Jose M. Canas, D. P. (2009). Visual Goal Detection for the RoboCup Standard Platform League. X Workshop De Agentes Fiscos.
[3] Pallares, A. M. (2009). Goal Detection for Soccer-playing Robots Based on Hough Transform. Universitat Rovira I Virgili.
[4] Stephen Se, D. L. (2002). Global Localization using Distinctive Visual Feature. IEEE Intl. Conference on Intellegent Robots and System.
[5] Madison Flannery, S. F. (2014). RANSAC: Identification of Higher-Order Geomteric Features and Applications in Humanoid Robot Soccer. Proceedings of Australian Conference on Robotic and Automation.
[6] Martin A. Fischler, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. SRI International.
[7] Muntaha, M. A. (2013). Penentuan posisi robot humanoid dalam lapangan sepak bola menggunakan metode triangulasi dan particle filter. Surabaya: Institut Teknologi Sepuluh Nopember.