Jurnal Teknik Elektro, Vol. 13, No. 1, Maret 2020, 32-37
ISSN 1411-870X
DOI: 10.9744/jte.11.1.1-6

SISTEM KENDALI PLATFORM DUCKIETOWN UNTUK MENJAGA JARAK AMAN 2 AUTONOMOUS CAR

Michael Emmanuel Victorious, Handry Khoswanto
Program Studi Teknik Elektro, Universitas Kristen Petra
Jl.Siwalankerto 121-131, Surabaya 60236, Indonesia
E-mail : michael.emmanuel0@gmail.com ; handry@petra.ac.id

Abstrak – **Penelitian ini memanfaatkan platform edukasi yang memiliki fokus pada pengembangan *autonomous car* yaitu Duckietown. Tujuan daripada penelitian ini adalah untuk mengembangkan platform Duckietown agar *autonomous car* (*Duckiebot*) dapat memiliki kemampuan untuk menjaga jarak aman. Platform Duckietown menggunakan *Robot Operating System* (ROS) sebagai kerangka dasar daripada pemrograman untuk *autonomous car* (*Duckiebot*). Pada penelitian ini, digunakan pola *circle grid* untuk mendeteksi adanya *autonomous car* lain serta melakukan pengukuran jarak dengan memanfaatkan OpenCV dalam *image processing*. Jarak yang dihasilkan dari hasil proses penangkapan gambar, digunakan dalam *proportional control* yang akan mengontrol besar kecepatan *autonomous car*. Beberapa pengujian yang dilakukan pada sistem adalah meliputi pengukuran akurasi pembacaan jarak, durasi pembacaan jarak, dan respon *proportional control*. Hasil daripada pengujian sistem menunjukkan, *autonomous car* dapat menjaga jarak aman terhadap *autonomous car* lain di depannya walau terdapat *error* pada pembacaan jarak serta didapati bahwa hasil respon *proportional control* dipengaruhi oleh durasi pembacaan jarak.**

***Kata Kunci – Autonomous Car, Circle Grid,* Duckietown, *Robot Operating System* (ROS), Raspberry Pi 3.**

I. PENDAHULUAN

Teknologi *autonomous car* menjadi salah satu hal yang sedang dikembangkan secara serius oleh banyak perusahaan besar seperti Tesla, Toyota, dan Google. *Autonomous car* adalah sebuah kendaraan yang dapat beroperasi dengan aman dan efektif tanpa perlu dikendalikan manusia. Kendaraan ini terdiri atas kumpulan sistem-sistem yang saling bekerja sama untuk memungkinkan kendaraan tersebut melintasi lingkungannya. Salah satu sistem yang paling penting pada *autonomous car* adalah sensor [1]. Salah satu platform *education prototype* daripada *autonomous car* adalah Duckietown. Platform Duckietown ini terdiri dari *autonomous car* dalam skala kecil yang bernama Duckiebot dan miniatur kota bernama Duckietown yang telah dilengkapi dengan jalan, lampu merah, dan rintangan [2]. Penelitian pada *autonomous car* untuk menjaga jarak sudah pernah dibuat oleh National Chiao Tung University dengan menggunakan platform Duckietown. Program yang dibuat, dapat mendeteksi gambar dengan pola *circle grid* pada *autonomous car* yang lain. Hal ini membuat *autonomous car* dapat mendeteksi keberadaan *autonomous car* lain dengan tepat. Namun, *autonomous car* masih tidak dapat mendeteksi jarak antara kedua *autonomous csar* sehingga tidak dapat menentukan besar jarak aman antar *autonomous car* [3]. Sehingga diperlukan pengembangan lebih lanjut lagi dalam meningkatkan kemampuan *autonomous car*, yaitu agar *autonomous car* dapat membaca jarak aman terhadap *autonomous car* lain di depannya serta mengontrol kecepatan *autonomous car* dengan menggunakan *proportional control*.

II. PERENCANAAN SISTEM

Seluruh sistem yang dirancang, menggunakan platform Duckietown sebagai dasar daripada perancangan sistem. Platform Duckietown digunakan sebagai dasar dalam melakukan desain arena Duckietown, desain *autonomous car*, dan perancangan program untuk menjaga jarak aman.

A. *Desain Arena Duckietown*

Gambar 1 merupakan desain arena yang dirancang berdasarkan aturan daripada platform Duckietown. Arena yang telah dirancang, memiliki ukuran panjang 3600 mm dan lebar 1000 mm. Radius putar balik pada arena yang telah dirancang adalah sebesar 180 derajat. Garis putih pada arena Duckietown ini memiliki ukuran lebar sebesar 50 mm sedangkan pada garis kuning memiliki lebar sebesar 25 mm dan panjang 101,6 mm (4 inch). Setiap garis kuning memiliki jarak sebesar 50,8 mm (2 inch). Arena ini dirancang untuk memiliki 2 jalur dan tidak ada halangan dari arah yang berlawanan. Lebar pada jalur kanan dan jalur kiri pada arena ini adalah sebesar 205 mm.

Gambar 1. Desain arena Duckietown

Arena Duckietown yang telah dibuat ditunjukkan pada gambar 2. Arena Duckietown yang telah dibuat menggunakan bahan dasar karet berwarna hitam yang memiliki ketebalan 3 mm. Penggunaan karet sebagai bahan dasar dari arena Duckietown ini dimaksudkan agar

mengurangi pantulan cahaya berlebih yang dapat mengganggu kinerja daripada kamera pada *autonomous car*. Karet sebagai bahan dasar kemudian dicat menggunakan cat karet untuk membentuk pola jalur. Pada bagian pinggir daripada arena Duckietown, dipasang pembatas dengan tinggi 260 mm yang telah dicat berwarna hitam dan menggunakan bahan dasar kayu lapis / tripleks. Penggunaan pembatas pada bagian pinggir daripada arena Duckietown adalah agar dapat mengurangi pantulan cahaya berlebih yang masuk pada kamera.

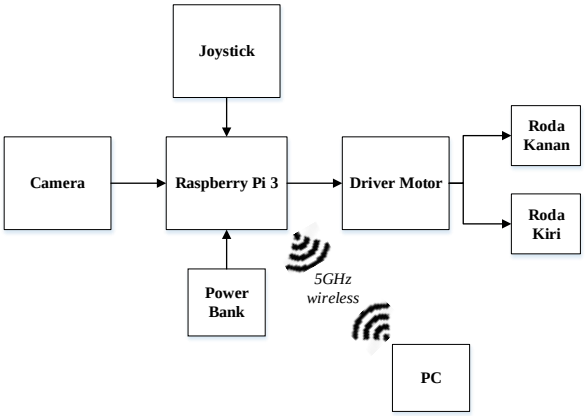
Gambar 2. Arena Duckietown

B. *Desain Autonomous Car*

Diagram koneksi *hardware* Duckiebot digambarkan pada gambar 3. *Hardware* yang digunakan pada Duckiebot adalah meliputi Raspberry Pi 3 Model B atau Raspberry Pi 3 Model B+, Adafruit DC & Stepper Motor Hat, *DC Motor Gearbox*, *USB joystick dongle*, Edimax 5GHz dongle, *fisheye camera* tipe G, dan *powerbank* dengan tegangan 5 volt yang dapat mengalirkan arus sebesar 2,4 ampere. Adafruit DC & Stepper Motor Hat dipasang diatas Raspberry Pi, kemudian dari modul Adafruit DC & Stepper Motor dihubungkan pada *DC Motor Gearbox*.

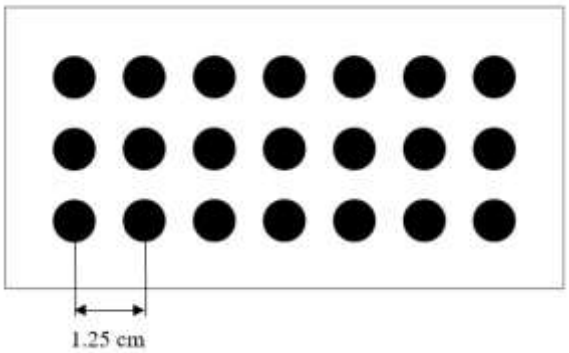
Koneksi antara Adafruit DC & Stepper Motor Hat dengan Raspberry Pi 3 menggunakan I2C. Dengan menggunakan I2C, maka Hat yang terpasang pada Raspberry Pi 3 akan memiliki *address* khusus. *Address* yang terdapat pada Hat ini adalah antara 0x60 sampai dengan 0x80 dengan total 32 *address* yang dapat digunakan. *Default address* pada Hat adalah 0x60. Dengan banyaknya *address* yang tersedia, maka dapat dilakukan *stacking* pada Hat. Pengubahan *address* pada Hat dapat dilakukan dengan melakukan *jumper* pada bagian *board i2c addr*. Terdapat *address* khusus yaitu 0x70 yang merupakan *address* "*all call*". Dinamakan "*all call*" karena semua Hat akan merespon pada *address* 0x70 tanpa memedulikan *jumper* pada bagian *board i2c addr*. *Address* yang digunakan pada Hat adalah 0x60, dimana Hat digunakan untuk mengontrol pin m1 dan m2.

Pada *USB port* Raspberry Pi dihubungkan Edimax 5GHz dongle dan *USB joystick dongle*. Untuk *fisheye camera* tipe G dihubungkan pada CSI *camera port* Raspberry Pi dengan menggunakan *ribbon cable* dengan 15 pin. *Powerbank* digunakan untuk mensuplai daya pada Raspberry Pi, digunakan *micro USB cable* yang kemudian dihubungkan pada *micro USB port* Raspberry Pi.



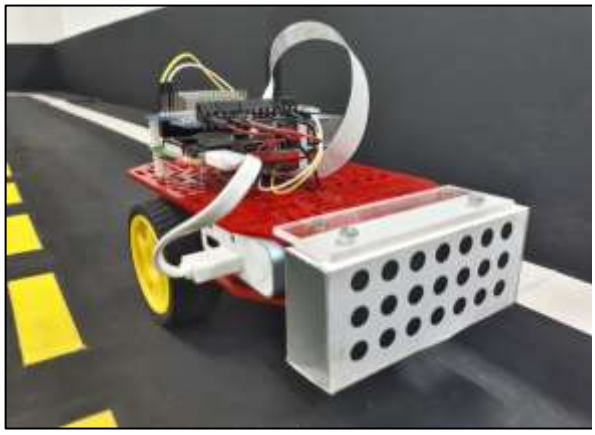
Gambar 3. Koneksi *hardware* Duckiebot

Digunakan pola *circle grid* yang memiliki jarak titik tengah antar lingkaran sebesar 1,25 cm. Pola *circle grid* ini membentuk sebuah persegi panjang dengan ukuran panjang sebesar 7 lingkaran dan lebar 3 lingkaran dimana semua lingkaran memiliki diameter yang sama yaitu 7,5 cm. Pola *circle grid* ini akan dipasang dibagian belakang *autonomous car* yang kemudian akan dideteksi oleh kamera pada *autonomous car* lain. Berikut merupakan pola *circle grid* pada gambar 4.



Gambar 4. *Circle Grid*
Sumber: [4]

Pola *circle grid* dipasang pada sebuah aluminium berbentuk balok. Aluminium berbentuk balok tersebut memiliki ukuran panjang sebesar 10,1 cm, lebar sebesar 2,2 cm, dan tinggi sebesar 4,4 cm. Kemudian aluminium berbentuk balok dipasangkan pada bagian belakang *autonomous car* dengan bantuan akrilik. Hasil pemasangan *circle grid* dapat dilihat pada gambar 5.

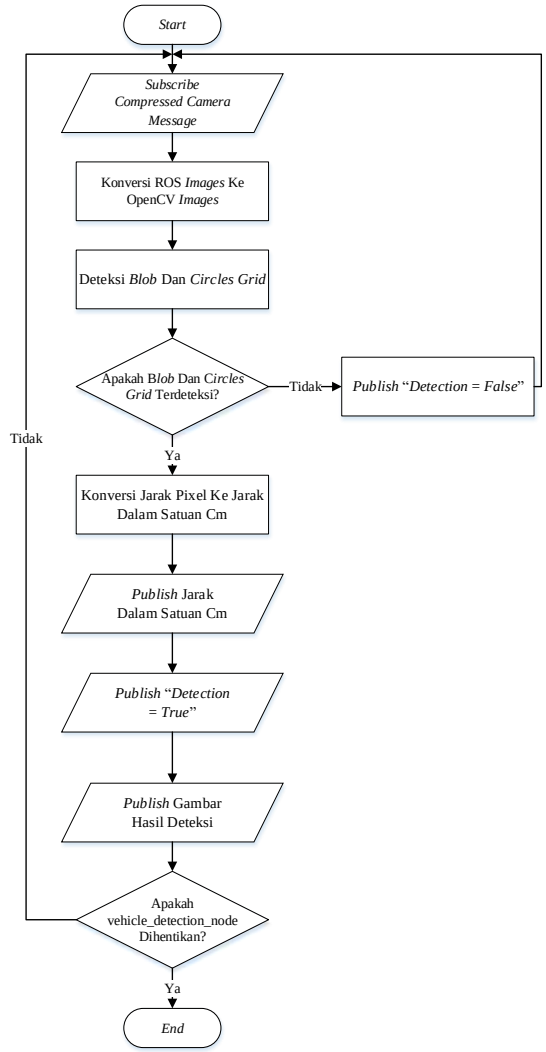


Gambar 5. *Circle Grid* pada *autonomous car*

C. *Program Jaga Jarak Aman*

Pada gambar 6 dijelaskan tentang proses kerja dari vehicle_detection_node. Proses dimulai dengan vehicle_detection_node melakukan *subscribe* pada *compressed camera message*. Setelah vehicle_detection_node menerima *compressed camera message* yang berisi *images* dari camera_node, maka vehicle_detection_node akan melakukan konversi *images*. Selanjutnya dilakukan pendeteksian pada *images* menggunakan *library* dari OpenCV yaitu pendeteksian *blob* menggunakan *SimpleBlobDetector* dan pendeteksian *circle grid* menggunakan *findCirclesGrid*.

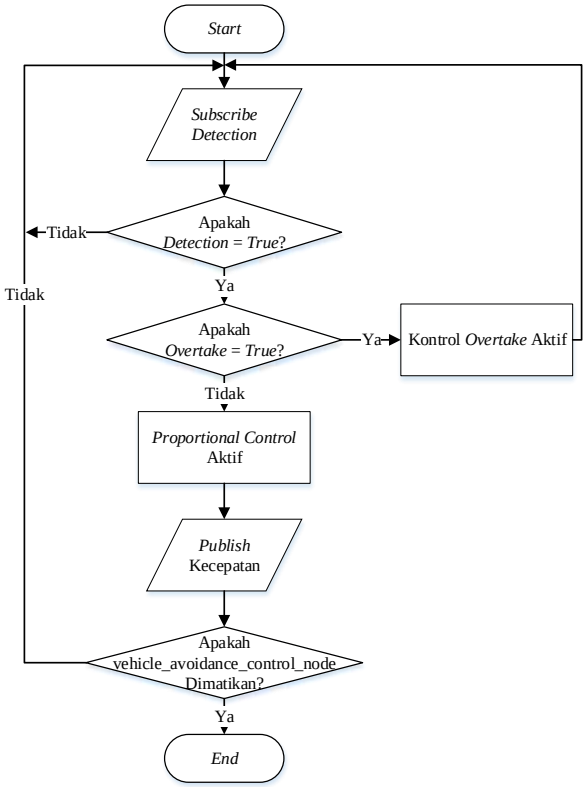
Apabila terdeteksi adanya *blob* dan *circle grid* pada *images* yang diproses, maka vehicle_detection_node akan melakukan konversi jarak *pixel* ke jarak dalam satuan cm dengan melakukan penghitungan menggunakan koordinat *pixel* yang terdeteksi. Vehicle_detection_node kemudian melakukan *publish* jarak hasil penghitungan, "*Detection = True*", dan *images* hasil deteksi. Jarak hasil penghitungan akan digunakan dalam *proportional control*, kemudian "*Detection = True*" menyatakan bahwa *autonomous car* mendeteksi adanya keberadaan *autonomous car* lain di depannya. *Images* hasil deteksi dapat dilihat dengan menggunakan *command* rqt_image_view pada *terminal* dengan tujuan untuk dapat melihat hasil dari pendeteksian pada *circle grid*. Pengulangan program pada vehicle_detection_node akan terus berjalan selama vehicle_detection_node tidak dihentikan.



Gambar 6. *Flowchart* vehicle_detection_node

Pada gambar 7 dijelaskan mengenai proses kontrol yang dilakukan oleh vehicle_avoidance_control_node sesudah dimodifikasi. Proses awal dimulai dengan melakukan *subscribe detection*. *Subscribe* ini dilakukan dengan tujuan agar vehicle_avoidance_control_node dapat mengetahui apakah ada *circle grid* yang terdeteksi oleh vehicle_detection_node. Saat vehicle_avoidance_control_ node tidak mendapatkan nilai *"Detection = True"* maka vehicle_avoidance_control_node akan kembali melakukan *subscribe detection*. Namun saat nilai yang didapatkan vehicle_avoidance_control_node adalah *"Detection = True"* maka proses akan masuk dalam kondisi berikutnya. Kondisi berikutnya adalah nilai dari *Overtake* merupakan *True* atau *False*. Saat vehicle_avoidance_control_node mendapatkan nilai *"Overtake = True"*, maka kontrol untuk melakukan *Overtake* akan aktif. Kontrol *Overtake* merupakan kontrol agar *autonomous car* mampu mendahului *autonomous car* yang terdeteksi berada di depannya. Setelah *autonomous car* menyelesaikan proses mendahului *autonomous car* di depannya maka *autonomous car* akan kembali melakukan *subscribe detection* kembali.

Saat vehicle_avoidance_control_node tidak mendapatkan nilai *"Overtake = True"* maka *proportional control* akan aktif. *Proportional control* digunakan untuk mengontrol kecepatan pada *autonomous car* sehingga jarak antara kedua *autonomous car* sesuai dengan *set point* yang telah ditentukan pada rumus *proportional control*. Hasil dari *proportional control* adalah besar kecepatan yang harus diberikan pada roda. Kemudian dilakukan *publish* untuk besar kecepatan dari *proportional control*. Pengulangan program vehicle_avoidance_control_node akan terus berlanjut hingga program vehicle_avoidance_control_node dimatikan.



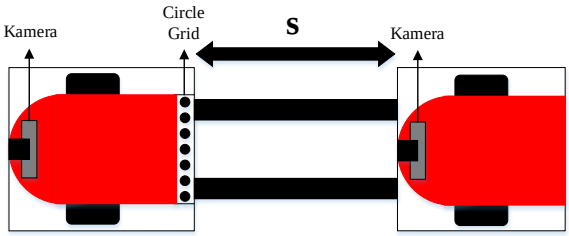
Gambar 7. *Flowchart* vehicle_avoidance_control_node

III. PENGUJIAN SISTEM

Dilakukan pengujian pada sistem yaitu meliputi pengujian pembacaan jarak pada *autonomous car*, pengujian durasi proses deteksi jarak, dan pengujian *proportional control* pada *autonomous car*.

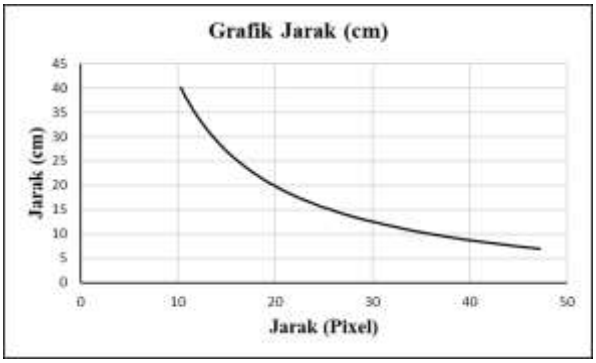
A. *Pengujian Pembacaan Jarak Pada Autonomous Car*

Pengujian ini dilakukan dengan tujuan untuk mengetahui tingkat akurasi konversi jarak dari persamaan jarak yang didapatkan dari regresi linear. Dilakukan pengambilan data jarak yang akan digunakan pada proses regresi linear. Metode pengambilan data dilakukan dengan cara meletakkan 2 *autonomous car* dalam posisi segaris pada jarak 7 cm hingga 40 cm, kemudian *autonomous car* akan melakukan pengambilan foto pada jarak 7 cm hingga 40 cm. Pengambilan foto dilakukan setiap penambahan jarak 0,5 cm. Data hasil dari foto yang diambil adalah jarak *pixel*. Ilustrasi pengambilan data dapat dilihat pada gambar 8.



Gambar 8. Metode pengambilan data jarak

Hasil pengambilan data ditampilkan pada gambar 9. Dari grafik hasil pengambilan data, dapat terlihat bahwa hasil daripada grafik adalah tidak linear dan cenderung turun. Karena hasil grafik yang cenderung turun dan tidak linear, maka dicari fungsi dasar daripada grafik tersebut dengan memasukkan data-data tersebut pada laman https://plot.ly/create/#/ sehingga memunculkan fungsi dasar dari grafik tersebut [5].



Gambar 9. Grafik Jarak (cm) terhadap Jarak (*Pixel*)

Fungsi dasar dari grafik pada gambar 9 kemudian diubah menjadi fungsi dasar persamaan garis lurus dengan tujuan agar mengurangi *error* pada hasil regresi linear.

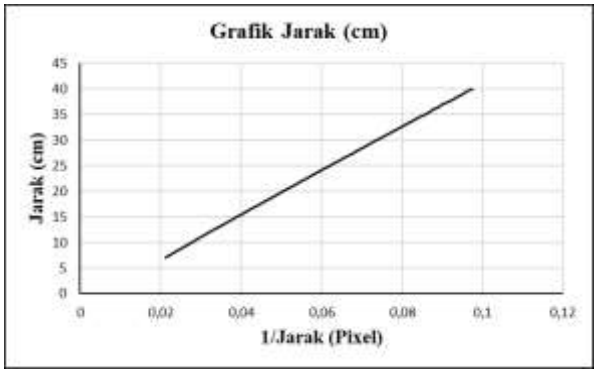
$$f(x) = y_0 + \frac{a}{x} \quad (4.1)$$

$$f(\frac{1}{x}) = y_0 + \frac{a}{\frac{1}{x}}$$

$$f(\frac{1}{x}) = y_0 + ax \quad (4.2)$$

Keterangan :
y_0 = konstanta
a = gradien

Dari penurunan fungsi dasar tersebut, didapatkan bahwa agar data yang didapatkan dapat membentuk persamaan garis lurus maka x diubah menjadi $\frac{1}{x}$. Nilai x yang akan digunakan pada regresi linear adalah data jarak (*pixel*). Apabila nilai x diubah menjadi $\frac{1}{x}$ maka grafik akan menjadi seperti pada gambar 10.



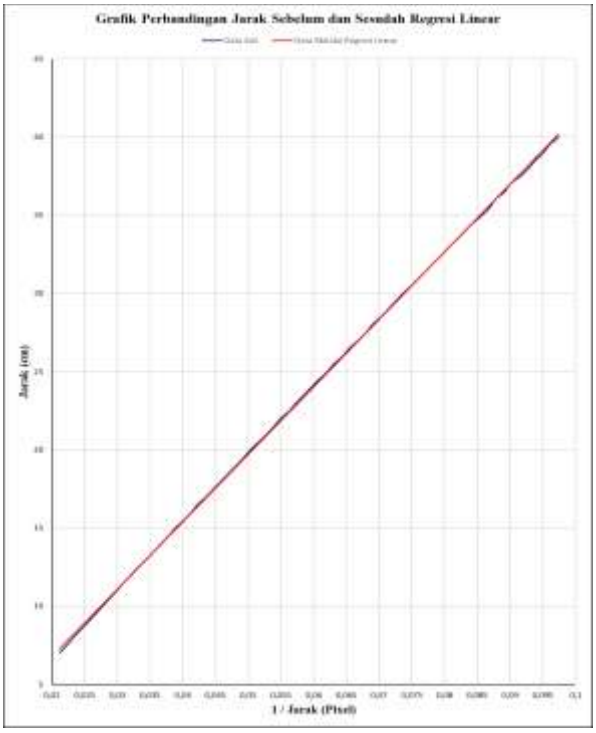
Gambar 10. Grafik Jarak (cm) terhadap 1/Jarak (*Pixel*)

Persamaan jarak yang didapatkan dari hasil regresi linear ditunjukkan pada persamaan dibawah.

y = (1 / x) * 431,8993525 - 1,884667361

Keterangan :
y = jarak dalam satuan cm
x = jarak dalam satuan *pixel*

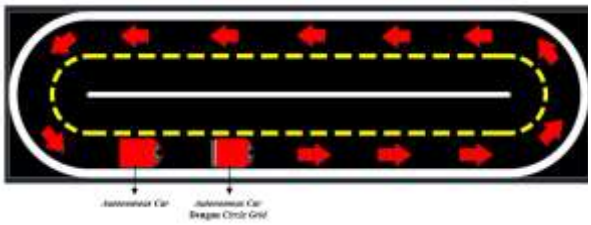
Pada gambar 11 merupakan grafik perbandingan jarak sebelum dan sesudah regresi linear. Pada grafik, data asli digambarkan dengan garis biru, sedangkan untuk data sesudah regresi linear digambarkan dengan garis merah. Dapat terlihat pada gambar 11 bahwa terjadi perubahan dari garis pada grafik, dimana garis biru pada grafik yang merupakan data asli tidak menghasilkan garis lurus. Sedangkan pada garis merah yang merupakan data sesudah regresi linear, mengubah garis biru yang tidak lurus menjadi lurus. Hal ini dapat terjadi karena proses regresi linear yang menghasilkan persamaan garis lurus.



Gambar 11. Grafik perbandingan jarak sebelum dan sesudah regresi linear

B. *Pengujian Durasi Proses Deteksi Jarak*

Pengujian ini bertujuan untuk mengetahui kemampuan *autonomous car* dalam menangkap dan memproses *circle grid* yang terdeteksi. Metode pengujian durasi proses deteksi jarak dilakukan dengan menjalankan 2 *autonomous car* pada arena Duckietown dimana *autonomous car* dengan *circle grid* akan melakukan *lane following* dan *autonomous car* di belakangnya akan melakukan *mode* menjaga jarak aman. Kondisi pengujian dilakukan sesuai dengan ilustrasi pada gambar 12. Setiap kali *autonomous car* mendeteksi adanya *circle grid*, maka data durasi deteksi jarak akan didapatkan. Data durasi deteksi jarak didapatkan melalui program yang telah dibuat pada *autonomous car*. *Library* yang digunakan adalah *rospy* dan *time*. Kode yang digunakan saat akan mulai mengambil waktu pada awal program deteksi adalah *rospy.Time.now()*, kode tersebut digunakan untuk mengambil waktu saat kode tersebut dijalankan. Waktu tersebut kemudian disimpan dalam variabel "start". Saat program deteksi selesai bekerja dalam 1 *cycle*, maka digunakan kode *(rospy.Time.now() – start).to_sec()* yang bertujuan untuk mengurangi waktu saat akhir program deteksi dengan waktu saat awal program deteksi. Hasil dari kode tersebut adalah durasi dari program deteksi dalam satuan detik.



Gambar 12. Metode pengujian durasi proses deteksi jarak

Pada tabel 1, rata-rata durasi pendeteksian adalah sebesar 0,628 detik. Data tersebut didapatkan dari pengambilan 20 data durasi proses deteksi jarak yang kemudian dimasukkan dalam penghitungan rata-rata. Dari data rata-rata durasi proses deteksi jarak ini, dapat diketahui juga berapa kali *autonomous car* dapat melakukan deteksi jarak saat berjalan dalam jalur lurus pada arena Duckietown. Panjang lintasan lurus arena Duckietown adalah sepanjang 238 cm. Kecepatan maksimal *autonomous car* adalah sebesar 20 cm/s. Dari data ini dapat dilakukan penghitungan sebagai berikut.

$s = v * t$
$s = 20 * 0{,}628$
$s = 12{,}56$ cm

Dari hasil penghitungan tersebut dapat diketahui bahwa a*utonomous car* akan melakukan pendeteksian setiap berjalan sejauh 12,56 cm. Dengan *autonomous car* berjalan dalam jalur yang memiliki panjang lintasan lurus sebesar 238 cm, maka penghitungan jumlah pendeteksian pada lintasan lurus adalah sebagai berikut.

$$Jumlah\ Pendeteksian = \frac{Total\ Panjang\ Arena}{Jarak\ 1\ kali\ deteksi}$$
$$Jumlah\ Pendeteksian = \frac{238}{12{,}56}$$
$Jumlah\ Pendeteksian = 18{,}95$ kali
$Jumlah\ Pendeteksian = 19$ kali

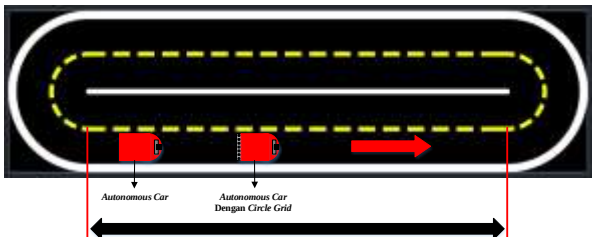
Melalui penghitungan di atas, dapat disimpulkan bahwa *autonomous car* dapat melakukan pendeteksian pada *circle grid* sebanyak 19 kali dalam kondisi *autonomous car* berjalan dalam jalur lurus pada arena Duckietown sejauh 238 cm dan dengan kecepatan konstan 20 cm/s.

Tabel 1. Tabel Hasil Pengujian Durasi Proses Deteksi Jarak

Pendeteksian Ke-	**Durasi (detik)**
1	0,613286019
2	0,636463881
3	0,633764028
4	0,629115104
5	0,522351026
6	0,742458105
7	0,706423997
8	0,639513969
9	0,688115119
10	0,571834087
11	0,64077878
12	0,685307026
13	0,647598982
14	0,559600115
15	0,512526989
16	0,668328047
17	0,676201105
18	0,638456821
19	0,634763956
20	0,514365912
Rata-rata	0,628062653

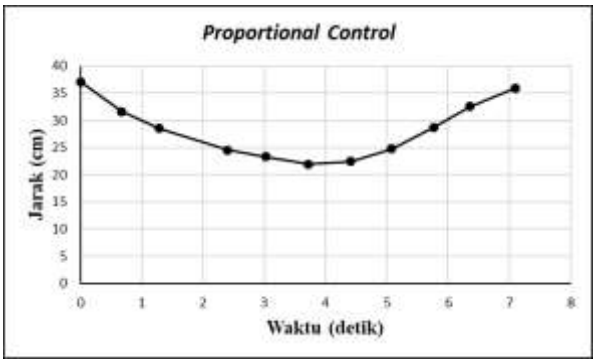
C. *Pengujian Proportional Control Pada Autonomous Car*

Pengujian ini dilakukan dengan tujuan untuk mengetahui respon kontrol daripada *proportional control* yang digunakan untuk mengontrol kecepatan pada *autonomous car*. Metode pengujian *proportional control* dilakukan dengan cara menjalankan *autonomous car* untuk melakukan jaga jarak dengan *autonomous car* di depannya pada jalur lurus arena Duckietown. Panjang jalur lurus pada arena Duckietown adalah sebesar 238 cm. *Autonomous car* akan menjaga jarak aman sebesar 30 cm dengan *autonomous car* di depannya. Pengambilan data dilakukan saat kedua *autonomous car* menjaga jarak aman pada jalur lurus arena Duckietown. Saat *autonomous car* melakukan jaga jarak dengan *autonomous car* di depannya, program menampilkan data jarak yang dibaca oleh *autonomous car* dan data kecepatan yang merupakan kondisi kecepatan daripada *autonomous car*.



Gambar 13. Metode pengujian *proportional control*

Pada gambar 14, merupakan grafik respon *proportional control* pada *autonomous car* saat berusaha untuk menjaga jarak aman. Saat jarak yang dibaca oleh *autonomous car* kurang dari *set point* yang telah ditentukan pada pada *proportional control autonomous car*, maka program *proportional control* akan melakukan penghitungan dan mengontrol kecepatan *autonomous car* sehingga jarak antara kedua *autonomous car* dapat dijaga.



Gambar 14. Grafik *Proportional Control*

Grafik *proportional control* yang terbentuk tidak dapat menunjukkan proses penstabilan jarak karena terbatasnya data yang dapat diambil. Mengacu pada pengujian durasi proses deteksi jarak, dimana jalur lurus arena Duckietown hanya sebesar 238 cm dan durasi deteksi jarak adalah sebesar 0,628 sehingga *autonomous car* hanya dapat mendeteksi maksimal 19 kali, pengambilan data pada pengujian *proportional control* ini hanya mendapat 11 data dari 11 kali pendeteksian.

Pada *autonomous car* hanya digunakan *proportional control* saja dan tidak ditambahkan *derivative control* karena tidak adanya pengaruh pada respon daripada *autonomous car*. Tujuan daripada *derivative control* adalah untuk menghilangkan osilasi, sehingga diperlukan banyak data untuk melihat respon daripada *derivative control*. Namun, karena jumlah data yang terbatas maka respon *derivative control* tidak dapat terlihat sehingga tidak ditambahkan pada sistem.

Kemampuan *autonomous car* dalam melakukan *lane following* dapat mempengaruhi banyaknya data yang terdeteksi oleh *autonomous car*. Selain itu, saat *autonomous car* melakukan deteksi dan bergerak maka fokus pada kamera untuk mendeteksi pola *circle grid* akan berkurang sehingga mempengaruhi kemampuan deteksi *autonomous car*.

IV. KESIMPULAN

- Dari hasil pengujian pembacaan jarak pada *autonomous car*, ditemukan adanya *error* pada pembacaan jarak dengan nilai *error* maksimal sebesar 3,794%.
- Dari hasil pengujian durasi proses deteksi jarak, ditemukan bahwa rata-rata durasi proses deteksi jarak pada *autonomous car* adalah sebesar 0,628 detik. Dari hasil penghitungan, pada jalur lurus arena Duckietown dengan panjang sebesar 238 cm, ketika *autonomous car* berjalan dengan kecepatan 20 cm/s maka dapat melakukan deteksi *circle grid* sebanyak 19 kali.
- Dari hasil pengujian *proportional control*, ditemukan bahwa *autonomous car* hanya dapat mendeteksi *circle grid* sebanyak 11 kali saat berjalan pada jalur lurus arena Duckietown dengan konstanta *gain* proporsional sebesar 0,5.

V. DAFTAR REFERENSI

[1] A. Wirjaputra, "*Mengungkap Teknologi Google Autonomous Car*," 2012. [Online]. Available: http://comp-eng.binus.ac.id/files/2012/06/Mengungkap-Teknologi-Google-Autonomous-Car-Andrew-W.pdf. [Accessed: 03-Nov-2018].

[2] L. Paull *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1497–1504, 2017.

[3] Duckietown-NCTU, "Duckietown-NCTU/Software," 2016. [Online]. Available: https://github.com/Duckietown-NCTU/Software/tree/master/catkin_ws/src/vehicle_detection. [Accessed: 09-Jun-2019].

[4] Duckietown, "Software/catkin_ws/src/50-misc-additional-functionality/vehicle_detection at master18," 2018. [Online]. Available: https://github.com/duckietown/Software/tree/master18/catkin_ws/src/50-misc-additional-functionality/vehicle_detection. [Accessed: 28-Jun-2019].

[5] plotly, "Online Graph Maker · Plotly Chart Studio," 2018. [Online]. Available: https://plot.ly/create/#/. [Accessed: 08-Jun-2019].