TUGAS AKHIR – TM184835

SIMULASI *DEEP LEARNING* PADA *AUTONOMOUS VEHICLE* MENGGUNAKAN NVIDIA JETBOT UNTUK MENJAGA JALUR KENDARAAN PADA LINTASAN DAN BERHENTI JIKA TERDETEKSI RINTANGAN

NAUFAL NABIL PRAMONO
NRP. 02111740000056

Dosen Pembimbing
Alief Wikarta S.T., MSc.Eng., Ph.D
NIP. 198202102006041002

Program Studi S-1 Teknik Mesin
Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember
Surabaya
2022

“Halaman ini sengaja dikosongkan.”

TUGAS AKHIR – TM184835

SIMULASI *DEEP LEARNING* PADA *AUTONOMOUS VEHICLE* MENGGUNAKAN NVIDIA JETBOT UNTUK MENJAGA JALUR KENDARAAN PADA LINTASAN DAN BERHENTI JIKA TERDETEKSI RINTANGAN

NAUFAL NABIL PRAMONO

NRP. 02111740000056

Dosen Pembimbing

Alief Wikarta, S.T. MSc.Eng., Ph.D

NIP. 198202102006041002

Program Studi S-1 Teknik Mesin

Departemen Teknik Mesin

Fakultas Teknologi Industri dan Rekayasa Sistem

Institut Teknologi Sepuluh Nopember

Surabaya

2022

“Halaman ini sengaja dikosongkan.”

FINAL PROJECT – TM184835

DEEP LEARNING SIMULATION ON AUTONOMOUS VEHICLE USING NVIDIA JETBOT TO MAINTAIN VEHICLE ON THE TRACKS AND STOP WHEN AN OBSTACLE IS DETECTED

NAUFAL NABIL PRAMONO

NRP. 02111740000056

Supervisor

Alief Wikarta, S.T. MSc.Eng., Ph.D

NIP. 198202102006041002

Undergraduate Study Program of Mechanical Engineering

Department of Mechanical Engineering

Faculty of Industrial Technology and Systems Engineering

Sepuluh Nopember Institute of Technology

Surabaya

2022

“Halaman ini sengaja dikosongkan.”

LEMBAR PENGESAHAN

SIMULASI *DEEP LEARNING* PADA *AUTONOMOUS VEHICLE* MENGGUNAKAN NVIDIA JETBOT UNTUK MENJAGA JALUR KENDARAAN PADA LINTASAN DAN BERHENTI JIKA TERDETEKSI RINTANGAN

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
Program Studi S-1 Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember

Oleh : **Naufal Nabil Pramono**
NRP. 02111740000056

Disetujui oleh Tim Penguji Tugas Akhir:

1. Alief Wikarta, ST., MSc. Eng., Ph.D. — Pembimbing
   NIP. 198202102006041002
2. Mohammad Khoirul Effendi, S.T., M.Sc.Eng., PhD. — Penguji
   NIP. 198204142010121001
3. Ir. Julendra B. Ariatedja, M.T. — Penguji
   NIP. 196807061999031004
4. Dr. Eng. Yohanes, S.T., M.Sc. — Penguji
   NIP. 198006272012121003

SURABAYA
Juli, 2022

“Halaman ini sengaja dikosongkan.”

APPROVAL SHEET

DEEP LEARNING SIMULATION ON AUTONOMOUS VEHICLE USING NVIDIA JETBOT TO MAINTAIN VEHICLE ON THE TRACKS AND STOP WHEN AN OBSTACLE IS DETECTED

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a degree Bachelor of Engineering at
Undergraduate Study Program of Mechanical Engineering
Department of Mechanical Engineering
Faculty of Industrial Technology and Systems Engineering
Sepuluh Nopember Institute of Technology

By: **Naufal Nabil Pramono**
NRP. 02111740000056

Approved by Final Project Examiner Team:

1. Alief Wikarta, ST., MSc. Eng., Ph.D. — Advisor
   NIP. 198202102006041002
2. Mohammad Khoirul Effendi, S.T., M.Sc.Eng., PhD. — Examiner
   NIP. 198204142010121001
3. Ir. Julendra B. Ariatedja, M.T. — Examiner
   NIP. 196807061999031004
4. Dr.Eng. Yohanes, S.T., M.Sc. — Examiner
   NIP. 198006272012121003

SURABAYA
July, 2022

“Halaman ini sengaja dikosongkan.”

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama / NRP	: Naufal Nabil Pramono / 02111740000056
Departemen	: Departemen Teknik Mesin
Dosen Pembimbing / NIP	: Alief Wikarta S.T, M.Sc. Eng, Ph.D / 198202102006041002

Dengan ini menyatakan bahwa Tugas Akhir dengan judul **"SIMULASI *DEEP LEARNING* PADA *AUTONOMOUS VEHICLE* MENGGUNAKAN NVIDIA JETBOT UNTUK MENJAGA JALUR KENDARAAN PADA LINTASAN DAN BERHENTI JIKA TERDETEKSI RINTANGAN"** adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataaan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 2 Agustus 2022

Mengetahui
Dosen Pembimbing,

Alief Wikarta ST., MSc.Eng. Ph.D
NIP. 198202102006041002

Mahasiswa,

Naufal Nabil Pramono
NRP. 02111740000056

“Halaman ini sengaja dikosongkan.”

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP	: Naufal Nabil Pramono / 021117400000056
Department	: Mechanical Engineering Department
Advisor / NIP	: Alief Wikarta S.T, M.Sc. Eng, Ph.D / 198202102006041002

Hereby declare that Final Project with the title of **"DEEP LEARNING SIMULATION ON AUTONOMOUS VEHICLE USING NVIDIA JETBOT TO MAINTAIN VEHICLE ON THE TRACKS AND STOP WHEN AN OBSTACLE IS DETECTED"** is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, August 2nd 2022

Acknowledged
Advisor,

Alief Wikarta ST., MSc.Eng., Ph.D
NIP. 198202102006041002

Student,

Naufal Nabil Pramono
NRP. 02111740000056

“Halaman ini sengaja dikosongkan.”

SIMULASI PENGGUNAAN *DEEP LEARNING* PADA *AUTONOMOUS VEHICLE* DENGAN NVIDIA JETBOT UNTUK MENJAGA JALUR KENDARAAN DAN BERHENTI JIKA TERDETEKSI RINTANGAN

Nama Mahasiswa : Naufal Nabil Pramono

NRP : 02111740000056

Departemen : Teknik Mesin FTIRS-ITS

Dosen Pembimbing : Alief Wikarta, S.T. MSc.Eng., Ph.D

ABSTRAK

Kecelakaan lalu lintas jalan raya yang menempati urutan ke-9 dengan rasio sebesar 2,2% dan akan diprediksikan meningkat hingga 3,6% pada tahun 2030. Kecelakaan Lalu Lintas tersebut menempati rasio sebesar 61% yang diakibatkan oleh faktor manusia, 30% diakibatkan oleh prasarana dan lingkungan, dan faktor kendaraan sebesar 9%. Adapun tujuan dilakukannya penelitian ini, yaitu simulasi Autonomous Vehicle yang dapat mengenali, memprediksi, serta dapat memutuskan suatu keputusan seperti adanya kendaraan lain yang tiba-tiba kehilangan kendali dan berada dekat dengan kendaraan yang sedang dikemudikan. Alat yang digunakan adalah NVIDIA JetBot dengan kecerdasan buatan berarsitektur CNN. NVIDIA JetBot ini akan dihadapkan dengan beberapa kondisi variabel, yaitu pengidentifikasi pola garis lintasan yang mengelilingi area, pengindentifikasian rintangan berupa objek, pengaturan kecepatan dan Kondisi Cahaya.

Tahapan pada penelitian ini terdiri dari menginput data dan mengatur kecepatan dari JetBot kemudian Collecting Data dengan sebanyak 50, 75, 100, dan 125 data dengan menggunakan Convolutional Neural Network dengan Arsitektur ResNet18 dan juga kecepatannya dimulai dari 0.16 dengan pertambahan 0.04 sampai 0.24. Setelah data dikumpulkan, maka Training Data akan dilakukan pada kondisi area yang telah dibentuk, serta kondisi cahaya kecil dan besar. Kemudian akan disimulasikan dengan kecepatan yang telah diberikan dan juga jumlah data yang telah dilatih. Setelah itu semua simulasi akan dicoba pada kondisi pencahayaan gelap dan terang. Pengolahan data didapat setelah semua simulasi selesai dilaksanakan dan kemudian analisis hasil akhir dilakukan untuk menentukan "Algoritma" terbaik dengan batasan yang ada yang dapat dilakukan oleh JetBot.

Hasil performa Nvidia Jetbot dalam melakukan fungsi Collision Avoidance dan Road Following pada pengujian di kondisi pencahayaan terang dengan dataset pada kondisi terang dengan dengan kecepatan berturut-turut sebesar 0.16, 0.20, dan 0.24, untuk 100 data gambar yaitu 76%, 56%, 12%. Untuk 150 data gambar yaitu 80%, 64%, 16%. Untuk 200 data gambar yaitu 84%, 72%, 20%. Untuk 250 Data gambar yaitu 92%, 84%, 24%. Pada pengujian di kondisi pencahayaan gelap dengan dataset pada kondisi terang dengan kecepatan yang sama pengujian sebelumnya, untuk 100 data gambar didapatkan keberhasilan 52%, 36%, 12%. Untuk 150 data gambar didapatkan keberhasilan 60%, 44%, 12%. Untuk 200 data gambar adalah 68%, 56%, 16%. Untuk 250 data gambar didapatkan 72%, 60%, 20%. Dan pada pengujian di kondisi pencahayaan gelap dengan dataset pada kondisi gelap dengan kecepatan yang sama dengan dua pengujian sebelumnya. Untuk 100 gambar dapat keberhasilan sebesar 20%, 16%, 8%. Untuk 150 gambar didapat keberhasilan sebesar 40%, 28%, 12%. Untuk 200 gambar didapatkan sebesar 48%, 32%, 12%. Dan untuk 250 gambar tingkat keberhasilannya adalah 64%, 40%, 16%.

Kata kunci: Autonomous Vehicle, Deep Learning, Convolutional Neural Network

“Halaman ini sengaja dikosongkan.”

DEEP LEARNING SIMULATION ON AUTONOMOUS VEHICLE USING NVIDIA JETBOT TO MAINTAIN VEHICLE ON THE TRACKS AND STOP WHEN AN OBSTACLE IS DETECTED

Nama Mahasiswa : Naufal Nabil Pramono

NRP : 02111740000056

Departemen : Teknik Mesin FTIRS-ITS

Dosen Pembimbing : Alief Wikarta, S.T. MSc.Eng., Ph.D

ABSTRACT

Road traffic accidents are ranked 9th with a ratio of 2.2% and will be predicted to increase to 3.6% in 2030. These traffic accidents occupy a ratio of 61% caused by human factors, 30% caused by infrastructure and environment, and the vehicle factor by 9%. The purpose of this research is to simulate an Autonomous Vehicle that can recognize, predict, and can make a decision, such as another vehicle suddenly losing control and being close to the vehicle being driven. The tool used is NVIDIA JetBot with CNN architecture artificial intelligence. NVIDIA JetBot will be faced with several variable conditions, namely identifying the pattern of the trajectory that surrounds the area, identifying obstacles in the form of objects, speed settings and lighting conditions.

The stages in this research consist of inputting data and adjusting the speed of the JetBot then Collecting Data with as much as 50, 75, 100, and 125 data using Convolutional Neural Network with ResNet Architecture and also the speed starts from 0.16 with 0.04 increments until 0.24. After the data is collected, the Data Training will be carried out on the condition of the area that has been formed, as well as small and large light conditions. Then it will be simulated with the given speed and the amount of data that has been trained. After that, all simulations will be tried in dark and bright lighting conditions. Data processing is obtained after all simulations are completed and then the final result analysis is carried out to determine the best "Algorithm" with existing limitations that can be done by JetBot.

Nvidia Jetbot performance results in performing Collision Avoidance and Road Following functions on tests in bright lighting conditions with datasets in bright conditions with speeds of 0.16, 0.20, and 0.24, for 100 image data with success rate of 76%, 56%, 12% . For 150 image data with success rate of 80%, 64%, 16%. For 200 image data with success rate of 84%, 72%, 20%. For 250 image data with success rate of 92%, 84%, 24%. In the test in dark lighting conditions with a dataset in bright conditions with the same speed as the previous test, for 100 image data the success was 52%, 36%, 12%. For 150 image data, the success was 60%, 44%, 12%. For 200 image data the success was 68%, 56%, 16%. For 250 image data success rate is 72%, 60%, 20%. And in testing in dark lighting conditions with a dataset in dark conditions at the same speed as the two previous tests. For 100 images data success rate is 20%, 16%, 8%. For 150 images data success rate is 40%, 28%, 12%. For 200 images data the success rate is 48%, 32%, 12%. And for 250 images the success rate is 64%, 40%, 16%.

Keyword: Autonomous Vehicle, Deep Learning, Convolutional Neural Network

“Halaman ini sengaja dikosongkan.”

KATA PENGANTAR

Puji dan syukur saya ucapkan kehadirat Tuhan Yang Maha Esa atas segala limpahan nikmat dan karuniaNya sehingga peneliti dapat menyelesaikan penulisan tugas akhir ini.

Tugas akhir ini berjudul "Simulasi *Deep Learning* pada *Autonomous Vehicle* Menggunakan Nvidia Jetbot untuk Menjaga Jalur Kendaraan pada Lintasan dan Berhenti jika Terdeteksi Rintangan" Penyusunan Tugas Akhir ini selain merupakan salah satu persyaratan yang harus dipenuhi untuk menyelesaikan pendidikan Tingkat Sarjana pada Fakultas Teknologi Industri dan Rekayasa Sistem, Departemen Teknik Mesin, Institut Teknologi Sepuluh Nopember. Pada kesempatan ini ijinkan penulis untuk mengucapkan terima kasih dan rasa hormat atas segala bantuan yang telah diberikan kepada penulis sehingga dapat menyelesaikan Laporan Tugas Akhir ini, yaitu:

1. Bapak Alief Wikarta, S.T, M.Sc. Eng, Ph.D., selaku Dosen Pembimbing tugas akhir saya yang telah membimbing, memberi masukan sehingga penulis dapat lebih menyempurnakan Laporan Tugas Akhir ini.
2. Bapak Mohammad Khoirul Effendi, S.T, M.Sc., Eng, Ph.D. selaku Dosen Penguji tugas akhir saya yang telah membimbing, memberi masukan sehingga penulis dapat lebih menyempurnakan Laporan Tugas Akhir ini.
3. Bapak Ir. Julendra B. Ariatedja, M.T., selaku Dosen Penguji tugas akhir saya yang telah membimbing, memberi masukan sehingga penulis dapat lebih menyempurnakan Laporan Tugas Akhir ini.
4. Ibu Dr. Eng., Yohanes, S.T., M.Sc., selaku Dosen Penguji tugas akhir saya yang telah membimbing, memberi masukan sehingga penulis dapat lebih menyempurnakan Laporan Tugas Akhir ini..
5. Seluruh Dosen, Staf, dan Karyawan Departemen Teknik Mesin FTI-RS ITS atas jasa-jasanya selama penulis menuntut ilmu.
6. Jerry, dan Mas Yoyok dalam penyusunan dan detik-detik terakhir yudisium
7. Mama, Papa, Atha, dan Mayfa atas dukungan dan doanya di setiap hal penyusunan Tugas Akhir ini.
8. Artina Syifa Ramadhanti yang selalu menemani dan membantu penulis
9. Semua pihak yang telah banyak memberikan bantuan yang tidak dapat penulis sebutkan satu persatu sehingga mengantarkan penulis untuk menyelesaikan Laporan Tugas Akhir ini.

Saya menyadari masih banyak kekurangan dalam penulisan tugas akhir ini. Oleh karenanya, berbagai kritik, saran, atau masukan yang membangun atas tulisan ini sangat saya apresiasi.

Surabaya, 2 Agustus 2022

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

“Halaman ini sengaja dikosongkan.”

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam kehidupan sehari-hari, kendaraan menjadi hal yang penting terutama dalam membantu akomodasi kegiatan manusia. Dengan kendaraan, akomodasi lebih efisien dalam waktu dan juga tenaga. Akan tetapi, dibalik semua kegiatan tersebut tidak dapat dihindari bahwa kecelakaan dalam penggunaan kendaraan tidak terbilang sedikit. Kecelakaan yang terjadi pada laporan *Global Status Report on Road Safety* yang dilakukan oleh WHO diketahui terjadi sebanyak 1,27 Juta Kecelakan setiap tahunnya dan akan terus bertambah seiring banyaknya pengguna kendaraan khususnya kendaraan bermotor. Pada laporan tersebut juga diketahui, bahwa pada tahun 2004 salah satu penyebab utama pada kematian di dunia yaitu kecelakaan lalu lintas jalan raya yang menempati urutan ke-9 dengan rasio sebesar 2,2% dan akan diprediksikan meningkat hingga 3,6% pada tahun 2030. Di Indonesia sendiri, berdasarkan Laporan Badan Pusat Statistik (BPS) dikatakan bahwa pada tahun 2018 sebesar 109.215 Kecelakaan Lalu Lintas, dimana sebesar 29.472 Korban Meninggal akibat Kecelakaan Lalu Lintas. Penyebab Kecelakaan Lalu Lintas terbesar dikarenakan tidak disiplin dalam berlalu lintas, rendahnya kesadaran, dan minimnya pengetahuan akan keselamatan berkendara. Kecelakaan Lalu Lintas tersebut menempati rasio sebesar 61% yang diakibatkan oleh faktor manusia, 30% diakibatkan oleh prasarana dan lingkungan, dan faktor kendaraan sebesar 9%.

Dari pernyataan yang didapat sebelumnya, sudah dapat disimpulkan bahwa angka peningkatan kecelakaan terbesar disebabkan oleh faktor manusia yang mengendalikan kendaraan bermotor. Oleh karena itu, diperlukan teknologi yang disematkan pada kendaraan bermotor yang dapat membantu manusia dalam pengendaliannya. Salah satunya yang menjadi solusi bagi hal tersebut adalah kecerdasan buatan yang dapat menggantikan peran manusia sebagai pengemudi atau dapat disebut sebagai *Autonomous Vehicle*, yang mempunyai peran untuk meminimalisir kecelakaan karena kelemahan manusia salah satu contohnya yaitu lelah dalam berkendara yang dapat mengakibatkan kontrol pengendara keluar dari jalur lintasan serta penghindaran dengan kendaraan lain sehingga mengakibatkan kecelakaan. Selain itu, *Autonomous Vehicle* juga dapat membantu pengendara dalam efisiensi berlalu lintas dikarenakan mempunyai variabel yang dapat mengetahui dan menjaga kendaraan tetap dalam lintasan sehingga kecelakaan dapat dihindari serta dapat menghindari adanya rintangan. Akan tetapi, dalam membuat *Autonomous Vehicle* diperlukan pemahaman kondisi pada kendaraan dan kemampuan dalam mengambil keputusan seperti halnya manusia dalam mengemudi kendaraan. Kondisi sekitar seperti halnya jalan berkelok, adanya jalur lain pada simpangan pertemuan, garis lintasan, hingga kendaraan lain yang tiba-tiba berpindah jalur ataupun kendaraan lain yang mengalami kehilangan kendali harus dapat diketahui, diprediksi, dan diantisipasi dengan presisi yang tinggi sehingga tidak merugikan penggunanya yang dapat berakibat fatal. Dari hal tersebut, maka diperlukan penyesuaian oleh sistem *Autonomous Vehicle* terhadap kondisi dan kemampuan dalam analisis lingkungan sekitarnya.

Penelitian mengenai *Autonomous Vehicle* sudah dilakukan sejak tahun 1920 tetapi penelitian yang menjanjikan dilakukan pada tahun 1988 dimana penelitian ini dilakukan oleh *DARPA* (*Defense Advanced Research Projects Agency*) dalam pengembangan *Autonomous Land Driven* (ALV) di Amerika Serikat yang memanfaatkan teknologi baru yang dikembangkan oleh Universitas Maryland, Universitas Carnegie Mellon, Institut Penelitian Lingkungan Michigan, Martin Marietta dan SRI Internasional. Proyek ALV mencapai demonstrasi mengikuti jalan pertama yang menggunakan LIDAR (*Light Radar*), *Computer Vision*, dan kontrol robotik otonom untuk mengarahkan kendaraan robot dengan kecepatan

hingga 19 mil per jam (31 km / jam). Pada tahun 1987, HRL Laboratories (sebelumnya Hughes Research Labs) mendemonstrasikan peta *off-road* pertama dan navigasi otonom berbasis sensor pada ALV. Kendaraan ini menempuh jarak lebih dari 2.000 kaki (610 m) dengan kecepatan 1,9 mil per jam (3,1 km / jam) di medan yang kompleks dengan lereng curam, jurang, bebatuan besar, dan vegetasi. Pada tahun 1989, *Carnegie Mellon University* telah mempelopori penggunaan jaringan saraf untuk mengarahkan dan sebaliknya mengontrol kendaraan otonom, membentuk dasar dari strategi kontrol kontemporer.

Pada tahun 2020, terdapat penelitian mengenai *Self-Driving Car* dengan *Deep Learning* dengan perangkat NVIDIA JetBot Nano oleh Shinji Kawakura dan Ryosuka Shibasaki dalam pengantaran barang pada kegiatan agrikultur dengan fungsi pencarian objek dan prediksi penghindaran. Dalam penelitiannya digunakan kecerdasan buatan dengan arsitektur *Convolutional Neural Network* (CNN), didapatkan hingga 56% dan 60% dari dua lintasan yang mempunyai variasi yang berbeda.

Adapun tujuan dilakukannya penelitian ini, yaitu simulasi *Autonomous Vehicle* yang dapat mengenali, memprediksi, serta dapat memutuskan suatu keputusan untuk berada di area lintasan yang tidak keluar dari jalur lintasan dan dapat menghindari ataupun berhenti jika ada rintangan. Alat yang digunakan adalah NVIDIA JetBot dengan kecerdasan buatan berasitektur CNN yaitu ResNet18. NVIDIA JetBot ini akan dihadapkan dengan beberapa kondisi variabel, yaitu pengenalan bentuk area lingkungan sekitar, identifikasi pola garis lintasan yang mengelilingi area, pengindentifikasian rintangan, dan pengaturan kontrol *Proportional-Integral-Derivative* (PID), dan Kondisi Cahaya. NVIDIA JetBot dikatakan dapat berhasil ketika dapat menghindari rintangan yang bergerak dengan baik dan prediksi objek bergerak tanpa ada kesalahan.

1.2 Rumusan Masalah

Dalam penyusunan tugas akhir ini, dirumuskan beberapa permasalahan yaitu sebagai berikut:

1. Berapa minimal jumlah data input yang dibutuhkan terhadap kemampuan NVIDIA JetBot dalam fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan?
2. Bagaimana pengaruh pencahayaan lintasan terhadap kemampuan NVIDIA JetBot dalam fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan?
3. Bagaimana pengaruh kecepatan terhadap kemampuan NVIDIA JetBot dalam fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan?

1.3 Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Mengetahui minimal jumlah data input terhadap kemampuan NVIDIA JetBot fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan.
2. Mengetahui pengaruh pencahayaan lintasan terhadap kemampuan NVIDIA JetBot dalam fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan
3. Mengetahui pengaruh kecepatan terhadap kemampuan NVIDIA JetBot dalam fungsi menjaga posisi pada jalur lintasan dan berhenti jika terdeteksi rintangan

1.4 Batasan Masalah

Batasan masalah yang digunakan pada penelitian ini adalah sebagai berikut:

1. Alat yang digunakan adalah NVIDIA *Autonomous Machine* tipe Waveshare JetBot AI Kit
2. Arsitektur kecerdasan buatan yang digunakan adalah *Convolutional Neural Network* dengan arsitektur ResNet18
3. Pemrograman menggunakan platform JupyterLab
4. Penggunaan *Epoch* setiap jenis fungsi yang digunakan adalah 30
5. Ukuran *Batch* yang digunakan adalah 8
6. Tingkat akurasi minimal adalah 60%
7. Komposisi Train-Test-Validation adalah 70%-20%-10%, 75%-15%-10%, dan 80%-10%-10%

1.5 Manfaat Penelitian

Dari penelitian ini, diharapkan dapat terciptanya suatu miniatur *Autonomous Vehicle* yang dapat beradaptasi dengan kondisi jalanan aktual. Penelitian ini juga diharapkan mampu diterapkan pada garis lintasan dan fungsi mendektsi sebenarnya, memprediksi hingga mengambil keputusan dalam penerapannya di kondisi aktual.

“Halaman ini sengaja dikosongkan.”

BAB II
DASAR TEORI

2.1 Autonomous Vehicle

Autonomous Vehicle atau bisa disebut dengan kendaraan otonom adalah kendaraan yang mampu merasakan lingkungannya dan bergerak dengan aman dengan sedikit atau tanpa masukan manusia. Pengendara tidak perlu melakukan kendali pada kendaraan. Kendaraan Otonom menggabungkan berbagai sensor untuk melihat sekelilingnya, seperti radar, lidar, sonar, GPS, odometri, dan unit pengukuran inersia. Sistem kontrol lanjutan menafsirkan informasi sensorik untuk mengidentifikasi jalur navigasi yang sesuai, serta rintangan dan rambu yang relevan.

2.2 Pertimbangan Terminologi dan Keamanan

Kendaraan modern menyediakan fitur-fitur seperti menjaga mobil tetap dalam jalurnya, kontrol kecepatan atau pengereman darurat. Fitur-fitur itu sendiri baru dianggap sebagai teknologi bantuan pengemudi karena masih memerlukan kontrol pengemudi manusia sementara kendaraan otomatis sepenuhnya dapat mengemudi sendiri tanpa masukan dari pengemudi manusia.

Menurut Fortune, beberapa nama teknologi kendaraan baru seperti *AutonoDrive*, *PilotAssist*, *Full-Self Driving* atau *DrivePilot*. Mungkin membingungkan pengemudi, yang mungkin percaya tidak ada masukan pengemudi yang diharapkan padahal sebenarnya pengemudi perlu tetap terlibat dalam tugas mengemudi. Menurut BBC, kebingungan antara konsep-konsep itu menyebabkan kematian.

Untuk alasan ini, beberapa organisasi seperti AAA (*American Automobile Association*) mencoba menyediakan konvensi penamaan standar untuk fitur seperti ALKS yang bertujuan untuk memiliki kapasitas untuk mengelola tugas mengemudi, tetapi belum disetujui untuk menjadi kendaraan otomatis di negara mana pun. *Association of British Insurers* (ABI) menganggap penggunaan kata otonom dalam pemasaran mobil modern berbahaya karena iklan mobil membuat pengendara berpikir 'otonom' dan 'autopilot' berarti kendaraan dapat mengemudi sendiri ketika mereka masih mengandalkan pengemudi untuk memastikan keselamatan. Teknologi sendiri masih belum mampu menggerakkan mobil.

Beberapa pembuat mobil menyarankan atau mengklaim kendaraan dapat mengemudi sendiri ketika mereka tidak dapat mengatur beberapa situasi mengemudi. Meskipun disebut Full Self-Driving, Tesla menyatakan bahwa penawarannya tidak boleh dianggap sebagai sistem mengemudi yang sepenuhnya otonom. Hal ini membuat pengemudi berisiko menjadi terlalu percaya diri, mengambil perilaku mengemudi yang terganggu, yang menyebabkan kecelakaan. Sementara di Inggris Raya, mobil yang dapat mengemudi sendiri sepenuhnya hanya merupakan mobil yang terdaftar dalam daftar tertentu. Ada juga usulan untuk mengadopsi pengetahuan keselamatan otomasi penerbangan ke dalam diskusi tentang penerapan aman kendaraan otonom, karena pengalaman yang telah diperoleh selama beberapa dekade oleh sektor penerbangan tentang topik keselamatan.

2.3 Otonom, Otomatis, dan Kooperatif

Otonom berarti mengatur sendiri. Banyak proyek bersejarah yang berkaitan dengan otomasi kendaraan telah diotomatiskan (dibuat otomatis) tergantung pada ketergantungan yang tinggi pada alat bantu buatan di lingkungan mereka, seperti strip magnetik. Kontrol otonom menyiratkan kinerja yang memuaskan di bawah ketidakpastian lingkungan yang signifikan, dan kemampuan untuk mengkompensasi kegagalan sistem tanpa intervensi eksternal.

Salah satu pendekatannya adalah dengan menerapkan jaringan komunikasi baik di sekitar (untuk menghindari tabrakan) dan jauh (untuk manajemen kemacetan). Pengaruh luar seperti itu dalam proses pengambilan keputusan mengurangi otonomi kendaraan individu, namun tetap tidak membutuhkan campur tangan manusia.

Istilah "otonom" dipilih "karena itu adalah istilah yang saat ini digunakan lebih luas (dan dengan demikian lebih akrab bagi masyarakat umum). Namun, istilah yang terakhir bisa dibilang lebih akurat. 'Otomatis' berkonotasi dengan kontrol atau operasi oleh sebuah mesin, sementara 'otonom' berkonotasi bertindak sendiri atau mandiri. Sebagian besar konsep kendaraan (yang saat ini kami ketahui) memiliki seseorang di kursi pengemudi, menggunakan koneksi komunikasi ke *Cloud* atau kendaraan lain, dan tidak secara mandiri memilih baik tujuan atau rute untuk mencapainya. Dengan demikian, istilah 'otomatis' akan menggambarkan konsep kendaraan ini dengan lebih akurat.

Pada 2017, sebagian besar proyek komersial berfokus pada kendaraan otomatis yang tidak berkomunikasi dengan kendaraan lain atau dengan rezim manajemen yang melingkupi. EuroNCAP mendefinisikan otonom dalam "Pengereman Darurat Otonom" sebagai: "sistem bertindak secara independen dari pengemudi untuk menghindari atau mengurangi kecelakaan." yang menyiratkan sistem otonom bukanlah pengemudi.

Di Eropa, kata otomatis dan otonom juga dapat digunakan bersama. Misalnya, Peraturan (UE) 2019/2144 Parlemen Eropa dan Dewan 27 November 2019 tentang persyaratan persetujuan jenis untuk kendaraan bermotor mendefinisikan "kendaraan otomatis" dan "kendaraan otomatis penuh" berdasarkan otonomnya kapasitas:

- "kendaraan otomatis" (*Automated Vehicle*) berarti kendaraan bermotor yang dirancang dan dibangun untuk bergerak secara otonom untuk jangka waktu tertentu tanpa pengawasan pengemudi yang berkelanjutan tetapi dalam hal mana intervensi pengemudi masih diharapkan atau diperlukan;
- "kendaraan otomatis penuh" (*Autonomous Vehicle*) berarti kendaraan bermotor yang telah dirancang dan dibangun untuk bergerak secara otonom tanpa pengawasan pengemudi;

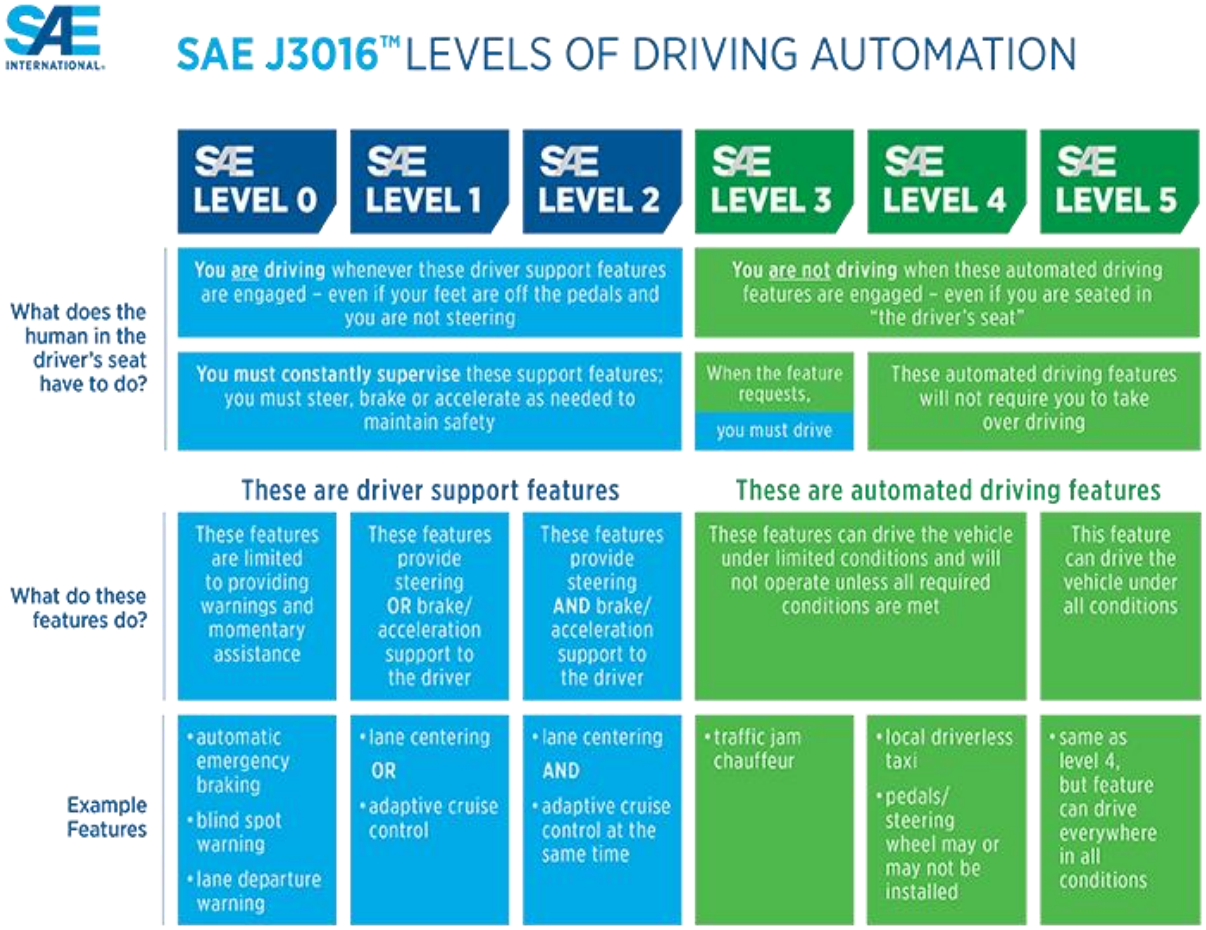
Dalam bahasa Inggris British, kata otomatis sendiri mungkin memiliki beberapa arti, seperti dalam kalimat: "Ditemukan bahwa sistem pemeliharaan jalur otomatis hanya dapat memenuhi dua dari dua belas prinsip yang diperlukan untuk menjamin keselamatan, selanjutnya dikatakan tidak bisa, oleh karena itu, digolongkan sebagai 'mengemudi otomatis', sebaliknya ia mengklaim bahwa teknologinya harus digolongkan sebagai 'mengemudi dengan bantuan'. Hukum Inggris menafsirkan arti "kendaraan otomatis" berdasarkan bagian interpretasi yang terkait dengan kendaraan "mengemudi sendiri" dan kendaraan yang diasuransikan.

2.4 Klasifikasi Sistem Otonom

Sebuah sistem klasifikasi dengan enam tingkat, mulai dari manual penuh hingga sistem otonom penuh diterbitkan pada tahun 2014 oleh SAE (*Society of Automotive Engineers*) International, sebuah badan standardisasi otomotif, sebagai J3016, Taksonomi dan Definisi untuk Istilah yang Terkait dengan Sistem Penggerak Otomatis Kendaraan Bermotor di Jalan Raya. Klasifikasi ini didasarkan pada jumlah intervensi dan perhatian pengemudi yang diperlukan, daripada kemampuan kendaraan, meskipun keduanya terkait secara longgar. Di Amerika Serikat pada tahun 2013, *National Highway Traffic Safety Administration* (NHTSA) merilis sistem klasifikasi formal, tetapi mengabaikannya untuk mendukung standar SAE pada tahun 2016. Juga pada tahun 2016, SAE memperbarui klasifikasinya, yang disebut J3016_201609.

Dengan tujuan menyediakan terminologi umum untuk mengemudi otomatis, standar baru SAE International J3016: Taksonomi dan Definisi untuk Istilah-istilah yang Berkaitan dengan Sistem Penggerak Otomatis Kendaraan Bermotor Di Jalan, memberikan sistem klasifikasi yang selaras dan definisi pendukung bahwa:

- Mengidentifikasi enam tingkat otomatisasi mengemudi dari "tanpa otomatisasi" ke "otomatisasi penuh".
- Mendasarkan definisi dan level pada aspek fungsional teknologi.
- Jelaskan perbedaan kategoris untuk kemajuan langkah-bijaksana melalui tingkat.
- Konsisten dengan praktik industri saat ini.
- Menghilangkan kebingungan dan berguna di berbagai disiplin ilmu (teknik, hukum, media, dan wacana publik).
- Mendidik komunitas yang lebih luas dengan menjelaskan untuk setiap tingkat peran apa (jika ada) pengemudi dalam melakukan tugas mengemudi dinamis saat sistem otomasi mengemudi dijalankan.



Gambar 2.1 SAE *Level of Automation*

Dikeluarkan Januari 2014, SAE international J3016 memberikan taksonomi dan definisi umum untuk mengemudi otomatis untuk menyederhanakan komunikasi dan memfasilitasi kolaborasi dalam domain teknis dan kebijakan. Ini mendefinisikan lebih dari selusin istilah kunci, termasuk itu dicetak miring di bawah, dan memberikan deskripsi lengkap serta contoh untuk setiap level.

Enam tingkat laporan tentang otomatisasi penggerak, dari tanpa otomatisasi hingga otomatisasi penuh. Perbedaan utama adalah antara level 2, di mana pengemudi manusia melakukan bagian dari tugas mengemudi dinamis, dan level 3, di mana sistem penggerak otomatis melakukan seluruh dinamika tugas mengemudi.

Tingkatan ini lebih deskriptif daripada normatif dan teknis daripada hukum. Mereka menyiratkan tidak ada urutan pengenalan pasar tertentu. Elemen menunjukkan kemampuan sistem minimum daripada maksimum untuk setiap level. Kendaraan tertentu mungkin memiliki banyak cara mengemudi fitur otomatisasi sedemikian rupa sehingga dapat beroperasi pada level yang berbeda tergantung pada fitur yang digunakan.

Sistem mengacu pada sistem bantuan pengemudi, kombinasi sistem bantuan pengemudi, atau sistem penggerak otomatis. Dikecualikan adalah peringatan dan sistem intervensi sesaat, yang tidak mengotomatiskan bagian mana pun dari tugas penggerak dinamis secara berkelanjutan dan oleh karena itu melakukannya tidak mengubah peran pengemudi manusia dalam melakukan tugas mengemudi dinamis.

Definisi utama di J3016 meliputi (antara lain):

Tugas mengemudi dinamis meliputi operasional (kemudi, pengereman, percepatan, pemantauan kendaraan dan jalan raya) dan taktis (menanggapi peristiwa, menentukan kapan harus mengubah jalur, berbelok, menggunakan sinyal, dll.) Aspek tugas mengemudi, tetapi tidak strategis (menentukan tujuan dan titik arah) dari tugas mengemudi.

Mode mengemudi adalah jenis skenario mengemudi dengan karakteristik persyaratan tugas mengemudi dinamis (misalnya, penggabungan jalan tol, kecepatan tinggi jelajah, kemacetan lalu lintas kecepatan rendah, operasi kampus tertutup, dll.).

Permintaan untuk campur tangan adalah pemberitahuan oleh sistem penggerak otomatis kepada pengemudi manusia bahwa ia harus segera memulai atau melanjutkan kinerja tugas mengemudi dinamis.

2.5 Tingkat Otomatisasi Mengemudi

Dalam definisi tingkat otomasi SAE, "mode mengemudi" berarti "jenis skenario mengemudi dengan karakteristik persyaratan tugas mengemudi dinamis (misalnya, penggabungan jalan tol, jelajah kecepatan tinggi, kemacetan lalu lintas kecepatan rendah, operasi kampus tertutup, dll.)"

Level 0: Sistem otomatis mengeluarkan peringatan dan mungkin melakukan intervensi sesaat tetapi tidak memiliki kontrol kendaraan yang berkelanjutan.

Level 1 ("Tetap dalam kontrol manusia"): Pengemudi dan sistem otomatis berbagi kendali kendaraan. Contohnya adalah sistem di mana pengemudi mengontrol kemudi dan sistem otomatis mengontrol tenaga mesin untuk mempertahankan kecepatan yang ditetapkan (Cruise Control) atau tenaga mesin dan rem untuk mempertahankan dan memvariasikan kecepatan (*Adaptive Cruise Control* atau ACC); dan Bantuan Parkir, di mana kemudi diotomatiskan sementara kecepatan dikendalikan secara manual. Pengemudi harus siap untuk mengambil kendali penuh kapan saja. *Lane Keeping Assistance* (LKA) *Type II* adalah contoh lebih lanjut dari self-driving Level 1. Pengereman darurat otomatis yang memperingatkan pengemudi akan kecelakaan dan memungkinkan kapasitas pengereman penuh juga merupakan fitur Level 1, menurut majalah *Autopilot Review*.

Level 2 ("lepas tangan"): Sistem otomatis mengambil kendali penuh atas kendaraan: akselerasi, pengereman, dan kemudi. Pengemudi harus memantau perjalanan dan bersiap untuk

segera turun tangan jika sistem otomatis gagal merespons dengan benar. Singkatan "lepas tangan" tidak dimaksudkan untuk diartikan secara harfiah - kontak antara tangan dan roda sering kali wajib selama SAE 2 mengemudi, untuk memastikan bahwa pengemudi siap untuk turun tangan. Mata pengemudi mungkin dipantau oleh kamera untuk memastikan bahwa pengemudi tetap memperhatikan lalu lintas. Contoh umum adalah *cruise control* adaptif yang juga memanfaatkan teknologi bantuan penjaga jalur sehingga pengemudi cukup memantau kendaraan, seperti "Super-Cruise" di Cadillac CT6 oleh General Motors.

Level 3 ("*eyes off*"): Pengemudi dapat dengan aman mengalihkan perhatian mereka dari tugas mengemudi, mis. pengemudi dapat mengirim pesan teks atau menonton film. Kendaraan akan menangani situasi yang memerlukan respons segera, seperti pengereman darurat. Pengemudi harus tetap bersiap untuk campur tangan dalam waktu yang terbatas, ditentukan oleh pabrikan, ketika diminta oleh kendaraan untuk melakukannya. Anda dapat menganggap sistem otomatis sebagai co-driver yang akan memberitahu Anda secara tertib saat tiba giliran Anda untuk mengemudi. Contohnya adalah Pengemudi Kemacetan Lalu Lintas, contoh lain adalah mobil yang memenuhi peraturan Sistem Penjaga Jalur Otomatis (ALKS) internasional.

Level 4 ("mind off"): Sebagai level 3, tetapi tidak ada perhatian pengemudi yang diperlukan untuk keselamatan, mis. pengemudi dapat dengan aman tidur atau meninggalkan kursi pengemudi. Namun, mengemudi sendiri hanya didukung di area spasial terbatas (geofence) atau dalam keadaan khusus. Di luar area atau keadaan ini, kendaraan harus dapat membatalkan perjalanan dengan aman, mis. memperlambat dan memarkir mobil, jika pengemudi tidak mengambil kendali kembali. Contohnya adalah taksi robotik atau layanan pengiriman robotik yang mencakup lokasi tertentu di suatu daerah, pada waktu dan jumlah tertentu.

Level 5 ("setir opsional"): Tidak diperlukan campur tangan manusia sama sekali. Contohnya adalah kendaraan robot yang bekerja di semua jenis permukaan, di seluruh dunia, sepanjang tahun, dalam segala kondisi cuaca.

Dalam definisi resmi SAE di bawah ini, transisi penting adalah dari SAE Level 2 ke SAE Level 3 di mana pengemudi manusia tidak lagi diharapkan untuk memantau lingkungan secara terus menerus. Di SAE 3, pengemudi manusia masih memiliki tanggung jawab untuk campur tangan ketika diminta melakukannya oleh sistem otomatis. Di SAE 4, pengemudi manusia selalu dibebaskan dari tanggung jawab itu dan di SAE 5 sistem otomatis tidak perlu meminta intervensi.

Tingkat Otomasi SAE telah dikritik karena fokus teknologinya. Telah diperdebatkan bahwa struktur level menunjukkan bahwa otomatisasi meningkat secara linear dan bahwa lebih banyak otomatisasi lebih baik, yang mungkin tidak selalu demikian. [78] Tingkat SAE juga tidak memperhitungkan perubahan yang mungkin diperlukan pada infrastruktur [79] dan perilaku pengguna jalan.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Gambar 2.2 Definisi Level dari Otomatis mengemudi

2.6 Traditional Programming dan Machine Learning

Pemrograman komputer tradisional telah ada selama lebih dari satu abad, dengan program komputer pertama yang diketahui berasal dari pertengahan tahun 1800-an. Pemrograman Tradisional mengacu pada program yang dibuat secara manual yang menggunakan data masukan dan berjalan di komputer untuk menghasilkan keluaran.

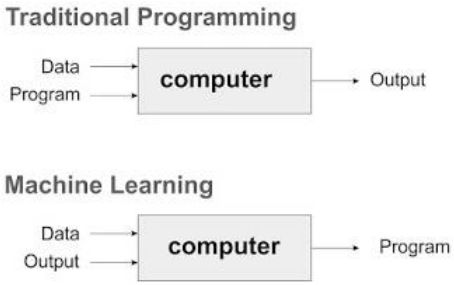
Tetapi selama beberapa dekade sekarang, jenis pemrograman tingkat lanjut telah merevolusi bisnis, terutama di bidang kecerdasan dan analitik yang disematkan. Dalam Machine Learning, juga dikenal sebagai augmented analytics, data input dan output dimasukkan ke algoritme untuk membuat program. Ini menghasilkan wawasan yang kuat yang dapat digunakan untuk memprediksi hasil di masa depan.

2.6.1 Traditional Programming

Pemrograman tradisional adalah proses manual artinya seseorang (programmer) menciptakan program. Tetapi tanpa siapa pun yang memprogram logika, seseorang harus merumuskan atau membuat kode aturan secara manual.

2.6.2 Machine Learning

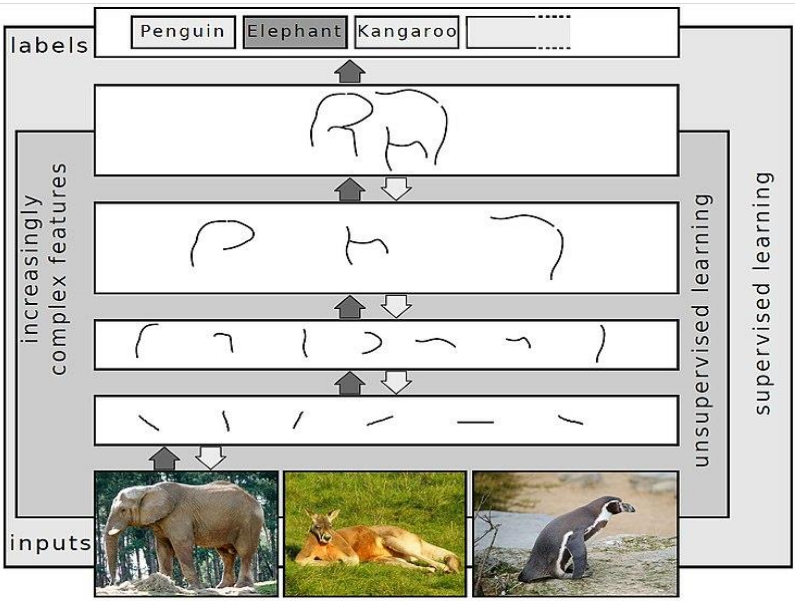
Tidak seperti pemrograman tradisional, pembelajaran mesin adalah proses otomatis. Ini dapat meningkatkan nilai analitik tersemat Anda di banyak area, termasuk persiapan data, antarmuka bahasa alami, deteksi pencilan otomatis, rekomendasi, serta deteksi kausalitas dan signifikansi. Semua fitur ini membantu mempercepat wawasan pengguna dan mengurangi bias keputusan.



Gambar 2.3 Perbedaan pemrograman tradisional dan *Machine Learning*

2.7 Deep Learning

Deep Learning atau juga dikenal sebagai pembelajaran terstruktur mendalam adalah bagian dari keluarga metode *Artificial Neural Networks* yang lebih luas berdasarkan *Representation Learning*. Pembelajaran dapat diawasi (*Supervised*), semi-supervisi (*Semi-supervised*), atau tanpa supervisi (*Unsupervised*). Pembelajaran mendalam (*Deep Learning*) adalah kelas algoritma pembelajaran mesin yang menggunakan beberapa lapisan untuk mengekstrak fitur tingkat yang lebih tinggi secara progresif dari masukan mentah. Misalnya, dalam pemrosesan gambar, lapisan bawah dapat mengidentifikasi tepi, sedangkan lapisan yang lebih tinggi dapat mengidentifikasi konsep yang relevan dengan manusia seperti angka atau huruf atau wajah.



Gambar 2.4 Pemrosesan Gambar melalui identifikasi konsep oleh *Deep Learning*

Dalam pembelajaran mendalam, setiap tingkat belajar mengubah data masukannya menjadi representasi yang sedikit lebih abstrak dan komposit. Dalam aplikasi pengenalan gambar, input mentah mungkin berupa matriks piksel; lapisan representasi pertama dapat mengabstraksi piksel dan menyandikan tepi; lapisan kedua dapat menyusun dan menyandikan pengaturan tepi; lapisan ketiga mungkin menyandikan hidung dan mata; dan lapisan keempat mungkin mengenali bahwa gambar tersebut mengandung wajah. Yang penting, proses pembelajaran dalam dapat mempelajari fitur untuk secara optimal tempat di mana tingkat sendiri. Tentu saja, ini tidak sepenuhnya menghilangkan kebutuhan akan hand-tuning; misalnya, berbagai jumlah lapisan dan ukuran lapisan dapat memberikan derajat abstraksi yang berbeda.

Kata "mendalam" dalam "pembelajaran mendalam" mengacu pada jumlah lapisan tempat data diubah. Lebih tepatnya, sistem pembelajaran mendalam memiliki kedalaman Credit Assignment Path (CAP) yang substansial. CAP adalah rantai transformasi dari input ke output. CAP menggambarkan hubungan kausal yang potensial antara input dan output.

Untuk jaringan syaraf maju umpan, kedalaman CAP sama dengan jaringan dan merupakan jumlah lapisan tersembunyi ditambah satu (karena lapisan keluaran juga diberi parameter). Untuk jaringan neural berulang, di mana sinyal dapat merambat melalui lapisan lebih dari sekali, kedalaman CAP berpotensi tidak terbatas.

Tidak ada ambang batas kedalaman yang disepakati secara universal membagi pembelajaran dangkal dari pembelajaran dalam, tetapi sebagian besar peneliti setuju bahwa pembelajaran dalam melibatkan kedalaman CAP lebih tinggi dari 2. CAP kedalaman 2 telah terbukti menjadi aproksimator universal dalam arti dapat meniru fungsi apa pun. Selain itu, lebih banyak lapisan tidak menambah kemampuan perkiraan fungsi jaringan.

Model dalam (CAP> 2) mampu mengekstrak fitur yang lebih baik daripada model dangkal dan karenanya, lapisan tambahan membantu mempelajari fitur secara efektif. Arsitektur pembelajaran mendalam dapat dibangun dengan metode lapis demi lapis yang besar. Pembelajaran mendalam membantu menguraikan abstraksi ini dan memilih fitur mana yang meningkatkan kinerja.

Untuk tugas pembelajaran yang diawasi, metode pembelajaran mendalam menghilangkan rekayasa fitur, dengan menerjemahkan data ke dalam representasi perantara yang kompak yang mirip dengan komponen utama, dan mendapatkan struktur berlapis yang menghilangkan redundansi dalam representasi.

Algoritma pembelajaran mendalam dapat diterapkan pada tugas-tugas pembelajaran tanpa pengawasan. Ini merupakan keuntungan penting karena data yang tidak berlabel lebih banyak daripada data yang berlabel. Contoh struktur dalam yang dapat dilatih dengan cara tanpa pengawasan adalah kompresor riwayat saraf dan jaringan keyakinan dalam.

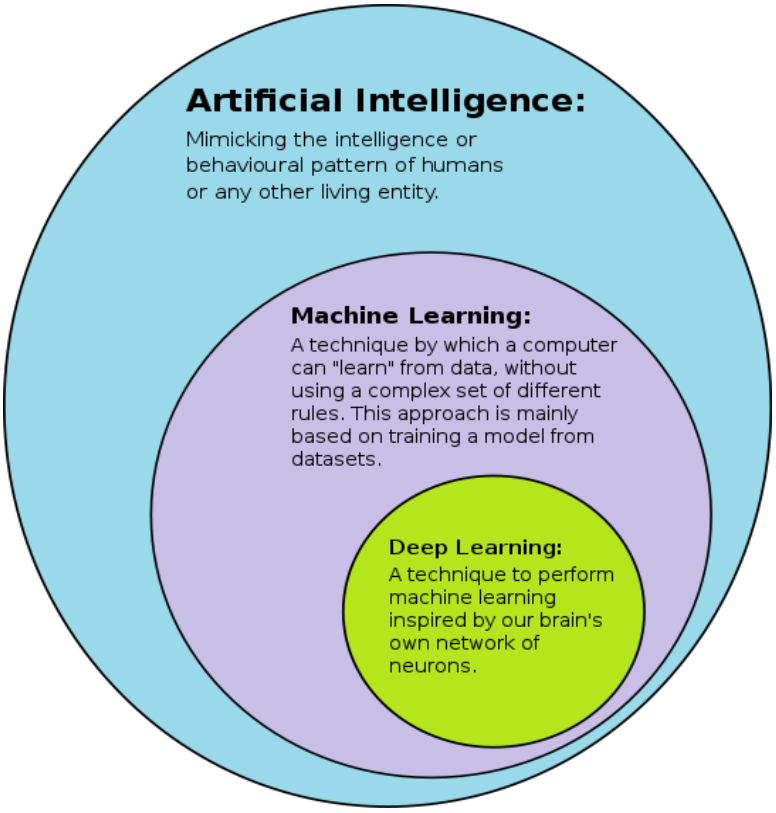
Deep Neural Network umumnya ditafsirkan dalam istilah teorema pendekatan universal atau inferensi probabilistik. Teorema aproksimasi universal klasik menyangkut kapasitas jaringan saraf maju umpan dengan satu lapisan tersembunyi berukuran terbatas untuk memperkirakan fungsi kontinu. Pada tahun 1989, bukti pertama diterbitkan oleh George Cybenko untuk fungsi aktivasi sigmoid dan digeneralisasikan untuk mengumpan maju arsitektur multi-layer pada tahun 1991 oleh Kurt Hornik.

Pekerjaan terbaru juga menunjukkan bahwa pendekatan universal juga berlaku untuk fungsi aktivasi tidak terbatas seperti unit linier yang diperbaiki. Teorema aproksimasi universal untuk jaringan neural dalam menyangkut kapasitas jaringan dengan lebar terbatas tetapi kedalamannya dibiarkan tumbuh. Lu dkk. membuktikan bahwa jika lebar jaringan neural dalam dengan aktivasi ULT benar-benar lebih besar dari dimensi masukan, maka jaringan dapat mendekati fungsi integral Lebesgue ; Jika lebarnya lebih kecil atau sama dengan dimensi input, maka deep neural network bukan merupakan aproksimeter universal.

The probabilistic interpretation atau diperoleh dari bidang pembelajaran mesin. Ini menampilkan inferensi, serta konsep optimasi pelatihan dan pengujian, masing-masing terkait dengan pemasangan dan generalisasi.

Lebih khusus lagi, interpretasi probabilistik menganggap nonlinier aktivasi sebagai fungsi distribusi kumulatif. Interpretasi probabilistik menyebabkan pengenalan dari putus sekolah sebagai regularizer di jaringan saraf. Interpretasi probabilistik diperkenalkan oleh para

peneliti termasuk Hopfield, Widrow dan Narendra dan dipopulerkan dalam survei seperti yang dilakukan oleh Bishop.



Gambar 2.5 Diagram Venn hubungan antara *Deep Learning*, *Machine learning*, dan *Artificial Learning*

2.8 Neural Networks

2.8.1 Artificial Neural Networks (Jaringan Saraf Tiruan)

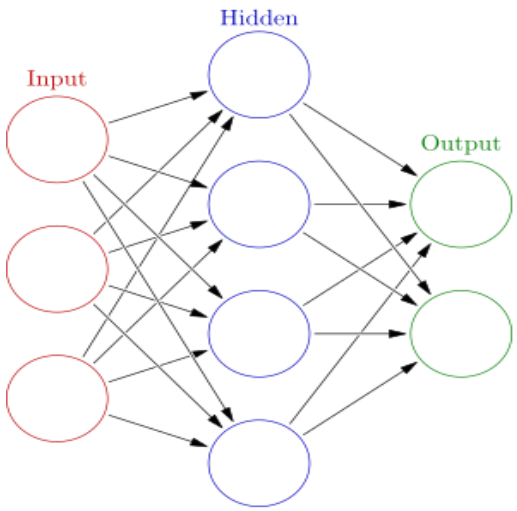
Artificial Neural Networks (ANN), biasanya hanya disebut jaringan saraf, adalah sistem komputasi yang secara samar-samar diilhami oleh jaringan saraf biologis yang membentuk otak hewan.

ANN didasarkan pada kumpulan unit atau node yang terhubung yang disebut neuron buatan, yang secara longgar memodelkan neuron di otak biologis. Setiap koneksi, seperti sinapsis di otak biologis, dapat mengirimkan sinyal ke neuron lain. Neuron buatan yang menerima sinyal kemudian memprosesnya dan dapat memberi sinyal pada neuron yang terhubung dengannya. "Sinyal" pada suatu koneksi adalah bilangan real, dan output dari setiap neuron dihitung oleh beberapa fungsi non-linier dari jumlah inputnya. Koneksi tersebut disebut *edge*. Neuron dan tepi biasanya memiliki bobot yang menyesuaikan saat pembelajaran berlangsung. Bobot menambah atau mengurangi kekuatan sinyal pada sambungan. Neuron mungkin memiliki ambang sedemikian sehingga sinyal dikirim hanya jika sinyal agregat melintasi ambang itu. Biasanya, neuron dikumpulkan menjadi lapisan. Lapisan yang berbeda dapat melakukan transformasi yang berbeda pada masukannya. Sinyal bergerak dari lapisan pertama (lapisan masukan), ke lapisan terakhir (lapisan keluaran), kemungkinan setelah melintasi lapisan tersebut beberapa kali.

Artificial Neural Networks dilatih dengan memproses contoh, yang masing-masing berisi "masukan" dan "hasil" yang diketahui, membentuk asosiasi pembobotan probabilitas

antara keduanya, yang disimpan dalam struktur data jaringan itu sendiri. Pelatihan jaringan saraf dari contoh yang diberikan biasanya dilakukan dengan menentukan perbedaan antara keluaran jaringan yang diproses (seringkali berupa prediksi) dan keluaran target. Ini adalah kesalahannya. Jaringan kemudian menyesuaikan pengaitan berbobotnya sesuai dengan aturan pembelajaran dan menggunakan nilai kesalahan ini. Penyesuaian yang berurutan akan menyebabkan jaringan saraf menghasilkan keluaran yang semakin mirip dengan keluaran target. Setelah cukup banyak penyesuaian ini, pelatihan dapat dihentikan berdasarkan kriteria tertentu. Ini dikenal sebagai pembelajaran yang diawasi.

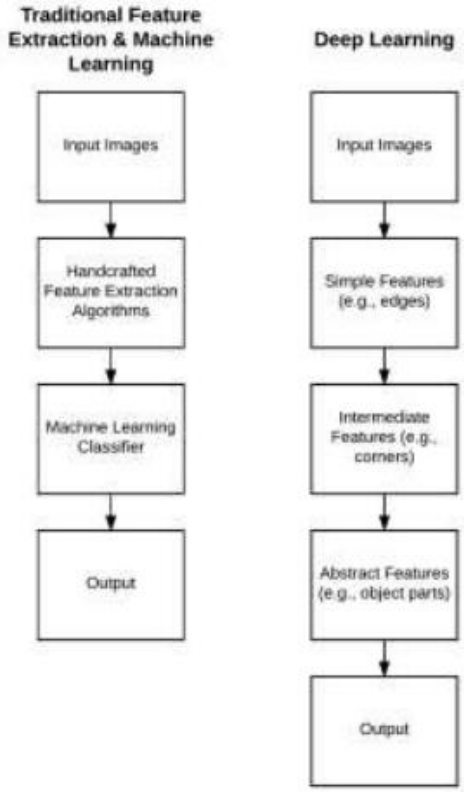
Sistem seperti itu "belajar" untuk melakukan tugas dengan mempertimbangkan contoh, umumnya tanpa diprogram dengan aturan khusus tugas. Misalnya, dalam pengenalan gambar, mereka mungkin belajar mengidentifikasi gambar yang berisi kucing dengan menganalisis contoh gambar yang telah diberi label secara manual sebagai "kucing" atau "tanpa kucing" dan menggunakan hasilnya untuk mengidentifikasi kucing di gambar lain. Mereka melakukan ini tanpa sepengetahuan kucing sebelumnya, misalnya, bahwa mereka memiliki bulu, ekor, kumis, dan wajah seperti kucing. Sebaliknya, mereka secara otomatis menghasilkan karakteristik pengenal dari contoh yang mereka proses.



Gambar 2.6 Konsep Artificial Neural Network yang diwakilkan oleh sekelompok node yang saling berhubungan

ANN terdiri dari neuron buatan yang secara konseptual diturunkan dari neuron biologis. Setiap neuron buatan memiliki masukan dan menghasilkan satu keluaran yang dapat dikirim ke banyak neuron lainnya. Input dapat berupa nilai fitur sampel data eksternal, seperti gambar atau dokumen, atau dapat berupa output neuron lain. Keluaran neuron keluaran akhir dari jaringan saraf menyelesaikan tugas, seperti mengenali objek dalam gambar.

Untuk menemukan keluaran neuron, pertama kita ambil jumlah bobot semua masukan, ditimbang dengan bobot koneksi dari masukan ke neuron. Kami menambahkan istilah bias ke jumlah ini. Jumlah tertimbang ini terkadang disebut aktivasi. Jumlah tertimbang ini kemudian dilewatkan melalui fungsi aktivasi (biasanya nonlinier) untuk menghasilkan keluaran. Input awal adalah data eksternal, seperti gambar dan dokumen. Keluaran akhir menyelesaikan tugas, seperti mengenali objek dalam gambar.



Gambar 2.7 Proses pengolahan data berupa input gambar oleh metode tradisional, dan deep learning

Fungsi propagasi menghitung masukan ke neuron dari keluaran neuron pendahulunya dan koneksinya sebagai jumlah tertimbang. Istilah bias dapat ditambahkan ke hasil propagasi.

Neuron biasanya diatur menjadi beberapa lapisan, terutama dalam pembelajaran yang mendalam. Neuron dari satu lapisan hanya terhubung ke neuron dari lapisan sebelumnya dan segera setelahnya. Lapisan yang menerima data eksternal adalah lapisan masukan. Lapisan yang menghasilkan hasil akhir adalah lapisan keluaran. Di antara mereka ada nol atau lebih lapisan tersembunyi. Lapisan tunggal dan jaringan tidak berlapis juga digunakan. Di antara dua lapisan, beberapa pola koneksi dimungkinkan. Mereka dapat terhubung sepenuhnya, dengan setiap neuron dalam satu lapisan terhubung ke setiap neuron di lapisan berikutnya. Mereka dapat dikumpulkan, dimana sekelompok neuron dalam satu lapisan terhubung ke satu neuron di lapisan berikutnya, sehingga mengurangi jumlah neuron di lapisan itu. Neuron dengan hanya koneksi seperti itu membentuk grafik asiklik terarah dan dikenal sebagai jaringan *feedforward*. Atau, jaringan yang memungkinkan koneksi antar neuron di lapisan yang sama atau sebelumnya dikenal sebagai jaringan berulang (RNN).

2.8.2 Deep Neural Networks (Jaringan Saraf Mendalam)

Deep Neural Networks (DNN) adalah *Artificial Neural Networks* (ANN) dengan beberapa lapisan antara lapisan masukan dan keluaran. Ada berbagai jenis jaringan saraf tetapi selalu terdiri dari komponen yang sama: neuron, sinapsis, bobot, bias, dan fungsi. Komponen ini berfungsi mirip dengan otak manusia dan dapat dilatih seperti algoritma ML lainnya. Misalnya, DNN yang dilatih untuk mengenali ras anjing akan memeriksa gambar yang diberikan dan menghitung probabilitas bahwa anjing dalam gambar tersebut adalah ras tertentu. Pengguna dapat meninjau hasil dan memilih probabilitas mana yang harus ditampilkan jaringan (di atas ambang tertentu, dll.) Dan mengembalikan label yang diusulkan. Setiap manipulasi

matematika dianggap sebagai lapisan, dan DNN kompleks memiliki banyak lapisan, oleh karena itu dinamai jaringan "dalam". DNN dapat memodelkan hubungan non-linier yang kompleks. Arsitektur DNN menghasilkan model komposisi yang objeknya diekspresikan sebagai komposisi primitif berlapis. Lapisan ekstra memungkinkan komposisi fitur dari lapisan bawah, berpotensi memodelkan data kompleks dengan unit yang lebih sedikit daripada jaringan dangkal yang berkinerja serupa. Misalnya, terbukti bahwa polinomial multivariasi renggang secara eksponensial lebih mudah untuk diperkirakan dengan DNN dibandingkan dengan jaringan dangkal. Arsitektur dalam mencakup banyak varian dari beberapa pendekatan dasar. Setiap arsitektur telah menemukan kesuksesan dalam domain tertentu. Tidak selalu mungkin untuk membandingkan kinerja beberapa arsitektur, kecuali mereka telah dievaluasi pada kumpulan data yang sama. DNN biasanya merupakan jaringan feedforward di mana data mengalir dari lapisan masukan ke lapisan keluaran tanpa mengulang kembali.

Pada awalnya, DNN membuat peta neuron virtual dan memberikan nilai numerik acak, atau "bobot", ke koneksi di antara mereka. Bobot dan masukan dikalikan dan menghasilkan keluaran antara 0 dan 1. Jika jaringan tidak secara akurat mengenali pola tertentu, algoritma akan menyesuaikan bobot. Dengan cara itu, algoritma dapat membuat parameter tertentu lebih berpengaruh, hingga ia menentukan manipulasi matematika yang benar untuk memproses data sepenuhnya. *Recurrent Neural networks* (RNN), dimana data dapat mengalir ke segala arah, digunakan untuk aplikasi seperti pemodelan bahasa. Memori jangka pendek sangat efektif untuk penggunaan ini. *Convolutional Neural Networks* (CNN) digunakan dalam visi komputer. CNN juga telah diterapkan pada pemodelan akustik untuk *Automatic Speech Recognition* (ASR).

2.8.3 Recurrent Neural Networks (Jaringan Saraf Berulang)

Recurrent Neural Networks (RNN) adalah kelas dari jaringan saraf tiruan di mana koneksi antara node membentuk grafik diarahkan sepanjang urutan temporal. Ini memungkinkannya untuk menunjukkan perilaku dinamis temporal. Berasal dari jaringan neural feedforward, RNN dapat menggunakan status internalnya (memori) untuk memproses urutan input dengan panjang variabel. Ini membuatnya dapat diterapkan untuk tugas-tugas seperti pengenalan tulisan tangan yang tidak tersegmentasi dan terhubung atau pengenalan ucapan. Istilah "jaringan saraf berulang" digunakan tanpa pandang bulu untuk merujuk pada dua kelas jaringan yang luas dengan struktur umum yang serupa, di mana yang satu adalah impuls hingga dan yang lainnya adalah impuls tak hingga. Kedua kelas jaringan menunjukkan perilaku dinamis temporal. Jaringan berulang impuls terbatas adalah grafik asiklik terarah yang dapat dibuka dan diganti dengan jaringan saraf maju yang ketat, sedangkan jaringan berulang impuls tak terbatas adalah grafik siklik terarah yang tidak dapat dibuka gulungannya. Jaringan berulang impuls terbatas dan impuls tak terbatas dapat memiliki status simpanan tambahan, dan penyimpanan dapat dikontrol langsung oleh jaringan saraf. Penyimpanan juga dapat diganti dengan jaringan atau grafik lain, jika itu menggabungkan penundaan waktu atau memiliki *loop* umpan balik. Status terkontrol seperti itu disebut sebagai status gated atau gated memory, dan merupakan bagian dari *Long-Short Term Memory* (LSTM) dan unit berulang yang terjaga keamanannya. Ini juga disebut Feedback Neural Network (FNN).

2.8.4 Convolutional Neural Networks (Jaringan Saraf Konvolusional)

Dalam pembelajaran mendalam, jaringan saraf konvolusional (CNN, atau ConvNet) adalah kelas jaringan saraf dalam, yang paling umum diterapkan untuk menganalisis citra visual. Mereka juga dikenal sebagai *Shift Invariant* atau *Space Invariant Artificial Neural Networks* (SIANN), berdasarkan arsitektur bobot-bersama dari kernel konvolusi atau filter yang

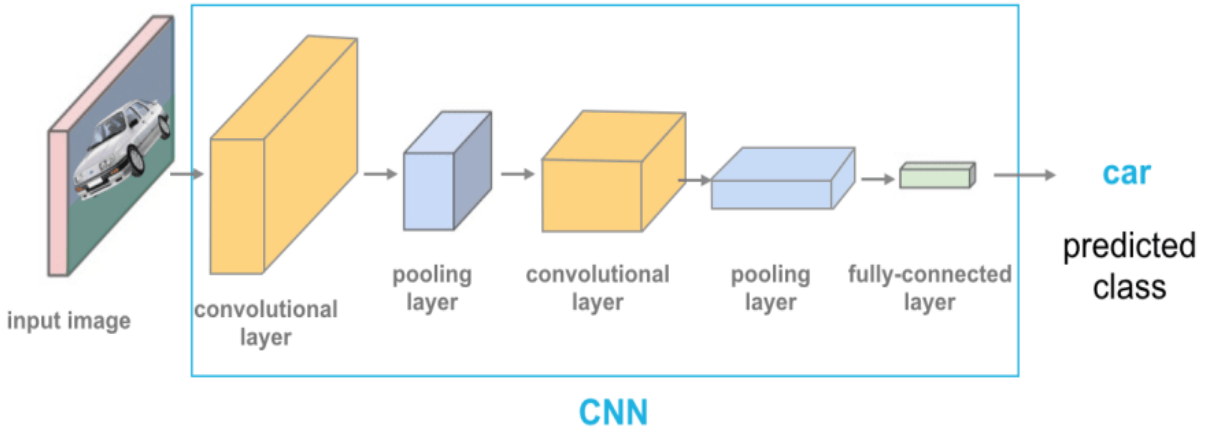
meluncur di sepanjang fitur masukan dan memberikan respons ekuivarian terjemahan yang dikenal sebagai peta fitur.

CNN adalah versi yang diatur dari multilayer perceptron. Perceptron multilayer biasanya berarti jaringan yang terhubung sepenuhnya, yaitu setiap neuron dalam satu lapisan terhubung ke semua neuron di lapisan berikutnya. "Konektivitas penuh" dari jaringan ini membuatnya rentan terhadap *overfitting* data. Cara khas regularisasi, atau pencegahan overfitting, meliputi: menghukum parameter selama pelatihan (seperti penurunan bobot) atau pemangkasan konektivitas (koneksi yang dilewati, putus, dll.) CNN mengambil pendekatan berbeda terhadap regularisasi: CNN memanfaatkan pola hierarki dalam data dan merakit pola yang semakin kompleks menggunakan pola yang lebih kecil dan lebih sederhana yang diembos dalam filternya. Oleh karena itu, pada skala konektivitas dan kompleksitas,

CNN berada pada titik ekstrem yang lebih rendah. Jaringan konvolusional terinspirasi oleh proses biologis di mana pola konektivitas antara neuron menyerupai organisasi korteks visual hewan. Neuron kortikal individu merespons rangsangan hanya di wilayah terbatas bidang visual yang dikenal sebagai bidang reseptif . Bidang reseptif dari neuron yang berbeda sebagian tumpang tindih sehingga menutupi seluruh bidang visual.

CNN menggunakan pra-pemrosesan yang relatif sedikit dibandingkan dengan algoritma klasifikasi gambar lainnya . Ini berarti bahwa jaringan belajar untuk mengoptimalkan filter (atau kernel) melalui pembelajaran otomatis, sedangkan dalam algoritma tradisional, filter ini direkayasa dengan tangan. Kemandirian dari pengetahuan sebelumnya dan campur tangan manusia dalam ekstraksi fitur merupakan keuntungan utama.

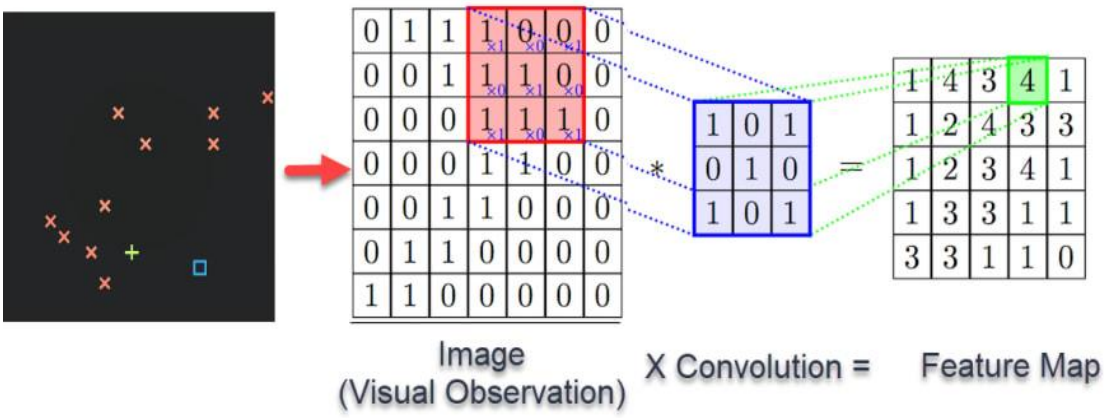
Nama "jaringan saraf konvolusional" menunjukkan bahwa jaringan tersebut menggunakan operasi matematika yang disebut konvolusi . Jaringan konvolusional adalah jenis jaringan saraf khusus yang menggunakan konvolusi sebagai pengganti perkalian matriks umum di setidaknya salah satu lapisannya. Berikut akan dijelaskan arsitektur CNN:



Gambar 2.8 Struktur Arsitektur *Convolutional Neural Networks*

a. Operasi Convolutional Neural Networks

Dalam proses ini, Pengurangan ukuran gambar dengan meneruskan gambar masukan melalui Detektor fitur / Filter / Kernel untuk mengubahnya menjadi Peta Fitur / fitur Konvolusi / Peta Aktivasi. Ini membantu menghilangkan detail yang tidak perlu dari gambar. Kita dapat membuat banyak peta fitur (mendeteksi fitur tertentu dari gambar) untuk mendapatkan lapisan konvolusi pertama kita. Melibatkan perkalian bijak dari filter konvolusional dengan potongan matriks masukan dan akhirnya penjumlahan semua nilai dalam matriks yang dihasilkan.



Gambar 2.9 Operasi *Convolution* pada sebuah matriks gambar

ReLU adalah fungsi aktivasi yang paling umum digunakan di dunia. Saat menerapkan konvolusi, ada risiko kita mungkin membuat sesuatu yang linier dan di sana kita perlu memutus linieritas. Satuan Rectified Linear dapat digambarkan dengan fungsi f (x) = max (x, 0). Kami menerapkan penyearah untuk meningkatkan non-linearitas pada gambar / CNN kami. Penyearah hanya menyimpan nilai non-negatif dari suatu gambar.

Dalam CNN, masukan adalah tensor dengan bentuk: (jumlah masukan) x (tinggi masukan) x (lebar masukan) x (saluran masukan). Setelah melewati lapisan konvolusional, gambar menjadi abstrak ke peta fitur, juga disebut peta aktivasi, dengan bentuk: (jumlah input) x (tinggi peta fitur) x (lebar peta fitur) x (saluran peta fitur). Lapisan konvolusional dalam CNN umumnya memiliki atribut berikut: Filter / kernel konvolusional yang ditentukan oleh lebar dan tinggi (hyper-parameter). Jumlah saluran masukan dan saluran keluaran (hyper-parameter). Saluran masukan satu lapisan harus sama dengan jumlah saluran keluaran (juga disebut kedalaman) masukannya. Hyperparameter tambahan dari operasi konvolusi, seperti: padding, stride, dan dilation. Lapisan konvolusional menggabungkan masukan dan meneruskan hasilnya ke lapisan berikutnya. Ini mirip dengan respons neuron di korteks visual terhadap rangsangan tertentu. Setiap neuron konvolusional memproses data hanya untuk bidang reseptifnya. Meskipun jaringan neural feedforward yang terhubung sepenuhnya dapat digunakan untuk mempelajari fitur dan mengklasifikasikan data, arsitektur ini umumnya tidak praktis untuk input yang lebih besar seperti gambar resolusi tinggi. Ini akan membutuhkan jumlah neuron yang sangat tinggi, bahkan dalam arsitektur yang dangkal, karena ukuran masukan gambar yang besar, di mana setiap piksel merupakan fitur masukan yang relevan. Misalnya, lapisan yang sepenuhnya terhubung untuk gambar (kecil) berukuran 100 x 100 masing - masing memiliki 10.000 bobot neuron di lapisan kedua. Sebaliknya, konvolusi mengurangi jumlah parameter bebas, memungkinkan jaringan menjadi lebih dalam. Misalnya, berapapun ukuran gambarnya, menggunakan wilayah petak 5 x 5, masing-masing dengan bobot yang sama, hanya memerlukan 25 parameter yang dapat dipelajari. Menggunakan bobot yang diatur pada parameter yang lebih sedikit menghindari gradien yang hilang dan masalah gradien yang meledak yang terlihat selama propagasi mundur di jaringan saraf tradisional. Selain itu, jaringan saraf konvolusional ideal untuk data dengan topologi seperti kisi (seperti gambar) karena hubungan spasial antara fitur terpisah diperhitungkan selama konvolusi dan / atau penggabungan.

b. Pooling

Ini membantu mengurangi ukuran spasial dari fitur berbelit-belit yang pada gilirannya membantu mengurangi daya komputasi yang diperlukan untuk memproses data. Di sini kami

dapat mempertahankan fitur dominan, sehingga membantu dalam proses melatih model secara efektif. Mengubah Peta Fitur menjadi Peta Fitur yang Dikumpulkan. Pooling dibagi menjadi 2 jenis:

- *Max Pooling* - Mengembalikan nilai maksimal dari porsi gambar yang dicakup oleh kernel.
- Pengumpulan Rata-rata - Mengembalikan rata-rata semua nilai dari bagian gambar yang dicakup oleh kernel.

Jaringan konvolusional dapat mencakup lapisan penyatuan lokal dan / atau global bersama dengan lapisan konvolusional tradisional. Lapisan penggabungan mengurangi dimensi data dengan menggabungkan keluaran dari gugus neuron di satu lapisan menjadi satu neuron di lapisan berikutnya. Penggabungan lokal menggabungkan kelompok kecil, ukuran ubin seperti 2 x 2 biasanya digunakan. Penyatuan global bekerja pada semua neuron dari peta fitur. Ada dua jenis penggabungan umum yang populer digunakan: maks dan rata-rata. Max pooling menggunakan nilai maksimum dari setiap cluster lokal neuron di peta fitur, sedangkan pooling rata-rata mengambil nilai rata-rata.

c. Flattening

Melibatkan mengubah peta fitur yang digabungkan menjadi vektor Kolom satu dimensi.

d. Full Connection

Output yang diratakan diumpankan ke jaringan saraf *feedforward* dengan backpropagation diterapkan ke setiap iterasi. Selama serangkaian zaman, model tersebut mampu mengidentifikasi fitur yang mendominasi dan fitur tingkat rendah dalam gambar dan mengklasifikasikannya menggunakan teknik Klasifikasi Softmax (Ini membawa nilai keluaran antara 0 dan 1). Lapisan yang terhubung sepenuhnya menghubungkan setiap neuron dalam satu lapisan ke setiap neuron di lapisan lain . Ini sama dengan jaringan saraf tiruan perceptron multi-layer tradisional (MLP). Matriks yang diratakan melewati lapisan yang terhubung sepenuhnya untuk mengklasifikasikan gambar.

e. Receptive Field

Dalam jaringan saraf, setiap neuron menerima masukan dari beberapa lokasi di lapisan sebelumnya. Dalam lapisan konvolusional, setiap neuron menerima masukan hanya dari wilayah terbatas pada lapisan sebelumnya yang disebut bidang reseptif neuron. Biasanya luasnya persegi (misalnya 5 kali 5 neuron). Sedangkan, dalam lapisan yang terhubung, bidang reseptifnya adalah seluruh lapisan sebelumnya. Jadi, di setiap lapisan konvolusional, setiap neuron mengambil masukan dari area masukan yang lebih besar dari lapisan sebelumnya. Ini karena menerapkan konvolusi berulang-ulang, yang memperhitungkan nilai piksel, dan piksel sekitarnya. Saat menggunakan dilatasi lapisan, jumlah piksel dalam bidang reseptif tetap konstan, tetapi bidang tersebut lebih diisi karena dimensinya bertambah saat gabungan beberapa lapisan.

f. Weight (Bobot)

Setiap neuron dalam jaringan saraf menghitung nilai keluaran dengan menerapkan fungsi tertentu ke nilai masukan yang diterima dari bidang reseptif di lapisan sebelumnya. Fungsi yang diterapkan ke nilai input ditentukan oleh vektor bobot dan bias (biasanya bilangan real). Pembelajaran terdiri dari penyesuaian berulang-ulang bias dan bobot ini. Vektor bobot dan bias disebut filter dan mewakili fitur tertentu dari input (misalnya, bentuk tertentu). Ciri khas CNN adalah banyak neuron dapat berbagi filter yang sama. Ini mengurangi jejak memori

karena bias tunggal dan satu vektor bobot digunakan di semua bidang reseptif yang berbagi filter itu, berbeda dengan setiap bidang reseptif yang memiliki bias dan pembobotan vektornya sendiri.

2.9 Paradigma Pembelajaran Neural Networks

Tiga paradigma pembelajaran utama adalah pembelajaran yang diawasi, pembelajaran tanpa pengawasan, dan pembelajaran penguatan. Masing-masing sesuai dengan tugas pembelajaran tertentu.

2.9.1 Supervised Learning (Pembelajaran yang diawasi)

Pembelajaran yang diawasi menggunakan satu set input berpasangan dan output yang diinginkan. Tugas belajar adalah menghasilkan keluaran yang diinginkan untuk setiap masukan. Dalam hal ini, fungsi biaya terkait dengan menghilangkan pemotongan yang salah. Biaya yang umum digunakan adalah kesalahan kuadrat rata-rata, yang mencoba meminimalkan kesalahan kuadrat rata-rata antara keluaran jaringan dan keluaran yang diinginkan. Tugas yang cocok untuk pembelajaran yang diawasi adalah pengenalan pola (juga dikenal sebagai klasifikasi) dan regresi (juga dikenal sebagai pendekatan fungsi). Pembelajaran yang diawasi juga berlaku untuk data sekuensial (misalnya, untuk tulisan tangan, ucapan dan pengenalan gerakan). Ini dapat dianggap sebagai pembelajaran dengan "guru", dalam bentuk fungsi yang memberikan umpan balik secara terus menerus tentang kualitas solusi yang diperoleh selama ini.

2.9.2 Unsupervised Learning (Pembelajaran tanpa diawasi)

Dalam pembelajaran tanpa pengawasan, data masukan diberikan bersama dengan fungsi biaya, beberapa fungsi data {\ displaystyle \ textstyle x} \ textstyle x dan keluaran jaringan. Fungsi biaya bergantung pada tugas (domain model) dan asumsi apriori apa pun (properti implisit model, parameternya, dan variabel yang diamati). Sebagai contoh sederhana, perhatikan modelnya f (x) = a, dimana a adalah konstanta dan biaya C = E [(xf (x)) ^ 2]. Meminimalkan biaya ini menghasilkan nilai a yang sama dengan rata-rata data. Fungsi biaya bisa jauh lebih rumit. Bentuknya bergantung pada aplikasinya: misalnya, dalam kompresi dapat dikaitkan dengan informasi timbal balik antara x dan f (x), sedangkan dalam pemodelan statistik, ini dapat dikaitkan dengan probabilitas posterior model yang diberikan data (perhatikan bahwa dalam kedua contoh tersebut jumlah tersebut akan dimaksimalkan daripada diminimalkan). Tugas yang termasuk dalam paradigma pembelajaran tanpa pengawasan dalam masalah estimasi umum; aplikasi termasuk pengelompokan, estimasi distribusi statistik, kompresi dan pemfilteran.

2.9.3 Reinforcement Learning (Pembelajaran Penguatan)

Dalam aplikasi seperti bermain video game, seorang aktor mengambil serangkaian tindakan, menerima respons yang umumnya tidak dapat diprediksi dari lingkungan setelah masing-masing tindakan. Tujuannya adalah untuk memenangkan permainan, yaitu menghasilkan respons paling positif (biaya terendah). Dalam pembelajaran penguatan, tujuannya adalah untuk membobotkan jaringan (menyusun kebijakan) untuk melakukan tindakan yang meminimalkan biaya jangka panjang (kumulatif yang diharapkan). Pada setiap titik waktu agen melakukan suatu tindakan dan lingkungan menghasilkan pengamatan dan biaya seketika, menurut beberapa aturan (biasanya tidak diketahui). Aturan dan biaya jangka panjang biasanya hanya bisa diperkirakan. Pada titik mana pun, agen memutuskan apakah akan mengeksplorasi tindakan baru untuk mengungkap biaya mereka atau mengeksploitasi pembelajaran sebelumnya untuk melanjutkan lebih cepat.

Secara formal lingkungan dimodelkan sebagai proses keputusan Markov (*Markov Decision Process*) dengan status s1, ..., sn ∈ S dan tindakan a1, ..., am ∈ A. Karena transisi keadaan tidak diketahui, maka digunakan distribusi probabilitas: distribusi biaya sesaat P (c_t | s_t), distribusi observasi P (x_t | s_t) dan distribusi transisi P (s_t + 1 | s_ t, a_t), sedangkan kebijakan didefinisikan sebagai distribusi bersyarat atas tindakan yang diberikan pengamatan. Secara bersama-sama, keduanya mendefinisikan rantai Markov (MC). Tujuannya adalah untuk menemukan MC berbiaya rendah.

ANN berfungsi sebagai komponen pembelajaran dalam aplikasi tersebut. Pemrograman dinamis digabungkan dengan ANN (memberikan pemrograman neurodinamik) telah diterapkan pada masalah seperti yang terlibat dalam perutean kendaraan, video game, manajemen sumber daya alam dan obat-obatan karena kemampuan ANN untuk mengurangi hilangnya akurasi bahkan ketika mengurangi kepadatan grid diskritisasi untuk mendekati solusi masalah kontrol secara numerik. Tugas yang termasuk dalam paradigma pembelajaran penguatan adalah masalah kontrol, permainan, dan tugas pengambilan keputusan berurutan lainnya.

2.9.4 Self-Learning (Pembelajaran Mandiri)

Pembelajaran mandiri dalam jaringan saraf diperkenalkan pada tahun 1982 bersama dengan jaringan saraf yang mampu belajar mandiri bernama *Crossbar Adaptive Array* (CAA). Ini adalah sistem dengan hanya satu masukan, situasi, dan hanya satu keluaran, tindakan (atau perilaku) a. Ini tidak memiliki masukan nasihat eksternal maupun masukan penguatan eksternal dari lingkungan. CAA menghitung, secara garis besar, keputusan tentang tindakan dan emosi (perasaan) tentang situasi yang dihadapi. Sistem ini didorong oleh interaksi antara kognisi dan emosi. Mengingat matriks memori W = ||w (a, s)||, algoritme pembelajaran mandiri palang di setiap iterasi melakukan komputasi berikut:

Nilai backpropagated (penguatan sekunder) adalah emosi terhadap situasi konsekuensi. CAA ada dalam dua lingkungan, satu lingkungan perilaku di mana ia berperilaku, dan yang lainnya adalah lingkungan genetik, di mana awalnya dan hanya sekali menerima emosi awal yang akan dihadapi situasi di lingkungan perilaku. Setelah menerima vektor genom (vektor spesies) dari lingkungan genetik, CAA akan mempelajari perilaku pencarian tujuan, dalam lingkungan perilaku yang berisi situasi yang diinginkan dan yang tidak diinginkan.

```
In situation s perform action a;
Receive consequence situation s';
Compute emotion of being in consequence situation v(s');
Update crossbar memory w'(a,s) = w(a,s) + v(s').
```

Gambar 2.10 Algoritma Pembelajaran Mandiri

2.10 Spatial Arrangement (Pengaturan Ruang Memori)

Tiga hyperparameter mengontrol ukuran volume keluaran lapisan konvolusional: kedalaman, langkah , dan ukuran bantalan.

2.10.1 Kedalaman Volume (Depth of Volume)

Kedalaman volume keluaran mengontrol jumlah neuron di lapisan yang terhubung ke wilayah yang sama dari volume masukan. Neuron-neuron ini belajar untuk mengaktifkan fitur-fitur berbeda pada input. Misalnya, jika lapisan konvolusional pertama mengambil gambar mentah sebagai masukan, maka neuron yang berbeda di sepanjang dimensi kedalaman dapat aktif dengan adanya berbagai tepi yang berorientasi, atau gumpalan warna.

2.10.2 Kontrol Proses (Stride Controls)

Langkah mengontrol bagaimana kolom kedalaman di sekitar lebar dan tinggi dialokasikan. Jika langkahnya 1, maka kami memindahkan filter satu piksel dalam satu waktu. Hal ini menyebabkan bidang reseptif yang sangat tumpang tindih antara kolom, dan volume keluaran yang besar. Untuk bilangan bulat S > 0, langkah S berarti filter diterjemahkan unit S pada satu waktu per keluaran. Dalam praktiknya, $S \geq 3$ jarang terjadi. Langkah yang lebih besar berarti lebih kecil tumpang tindih bidang reseptif dan dimensi spasial volume keluaran yang lebih kecil.

2.10.3 Kontrol Lapisan (Padding Control)

Terkadang, akan lebih mudah untuk mengisi input dengan nol (atau nilai lain, seperti rata-rata kawasan) di perbatasan volume input. Ukuran padding ini adalah hyperparameter ketiga. Padding memberikan kontrol ukuran spasial volume keluaran. Secara khusus, terkadang diinginkan untuk mempertahankan ukuran spasial volume input secara tepat, ini biasanya disebut sebagai padding yang "sama".

Ukuran spasial dari volume keluaran adalah fungsi dari ukuran volume masukan W, ukuran bidang kernel K dari neuron lapisan konvolusional, langkah S, dan jumlah bantalan nol P di perbatasan. Maka, jumlah neuron yang "cocok" dalam volume tertentu adalah:

$$\frac{W - K + 2P}{S} + 1.$$

Gambar 2.11 Rumus dalam menggunakan *Spasial Volume*

Jika angka ini bukan bilangan bulat, maka langkahnya salah dan neuron tidak dapat disusun agar sesuai dengan volume input secara simetris. Secara umum, pengaturan zero padding menjadi P = (K-1) / 2 saat langkahnya adalah S = 1 memastikan bahwa volume input dan volume output akan memiliki ukuran spasial yang sama. Namun, tidak selalu sepenuhnya perlu menggunakan semua neuron dari lapisan sebelumnya. Misalnya, perancang jaringan neural mungkin memutuskan untuk menggunakan hanya sebagian dari bantalan.

2.11 Parameter Sharing (Parameter yang berbagi)

Skema berbagi parameter digunakan dalam lapisan konvolusional untuk mengontrol jumlah parameter bebas. Ini bergantung pada asumsi bahwa jika fitur tambalan berguna untuk menghitung pada beberapa posisi spasial, fitur tambalan juga harus berguna untuk menghitung di posisi lain. Menunjukkan potongan kedalaman 2 dimensi sebagai potongan kedalaman, neuron di setiap potongan kedalaman dibatasi untuk menggunakan bobot dan bias yang sama.

Karena semua neuron dalam satu irisan kedalaman memiliki parameter yang sama, lintasan maju di setiap irisan kedalaman lapisan konvolusional dapat dihitung sebagai konvolusi bobot neuron dengan volume masukan. [Nb 2] Oleh karena itu, hal ini umum untuk merujuk ke kumpulan bobot sebagai filter (atau kernel), yang digabungkan dengan input. Hasil dari konvolusi ini adalah peta aktivasi, dan sekumpulan peta aktivasi untuk setiap filter yang berbeda ditumpuk bersama sepanjang dimensi kedalaman untuk menghasilkan volume keluaran. Berbagi parameter berkontribusi pada invariansi terjemahan dari arsitektur CNN.

Terkadang, asumsi berbagi parameter mungkin tidak masuk akal. Hal ini terutama terjadi ketika gambar masukan ke CNN memiliki beberapa struktur terpusat tertentu; untuk itu kami mengharapkan fitur yang sangat berbeda untuk dipelajari di lokasi spasial yang berbeda. Salah satu contoh praktisnya adalah ketika input adalah wajah yang telah dipusatkan pada

gambar: kita mungkin mengharapkan fitur khusus mata atau rambut yang berbeda dipelajari di bagian gambar yang berbeda. Dalam hal ini, umum untuk melonggarkan skema berbagi parameter, dan sebagai gantinya cukup memanggil lapisan sebagai "lapisan yang terhubung secara lokal".

2.12 Pooling Layer (Lapisan Penggabungan)

Ada beberapa fungsi non-linier untuk mengimplementasikan penggabungan, di mana penggabungan maksimal adalah yang paling umum. Ini mempartisi gambar masukan menjadi satu set persegi panjang dan, untuk setiap sub-wilayah tersebut, menghasilkan keluaran maksimum.

Secara intuitif, lokasi pasti dari suatu fitur kurang penting dibandingkan dengan lokasi kasarnya dibandingkan dengan fitur lainnya. Ini adalah ide di balik penggunaan *pooling* dalam jaringan neural konvolusional. Lapisan penyatuan berfungsi untuk secara progresif mengurangi ukuran spasial representasi, untuk mengurangi jumlah parameter, jejak memori dan jumlah komputasi dalam jaringan, dan karenanya juga mengontrol overfitting.

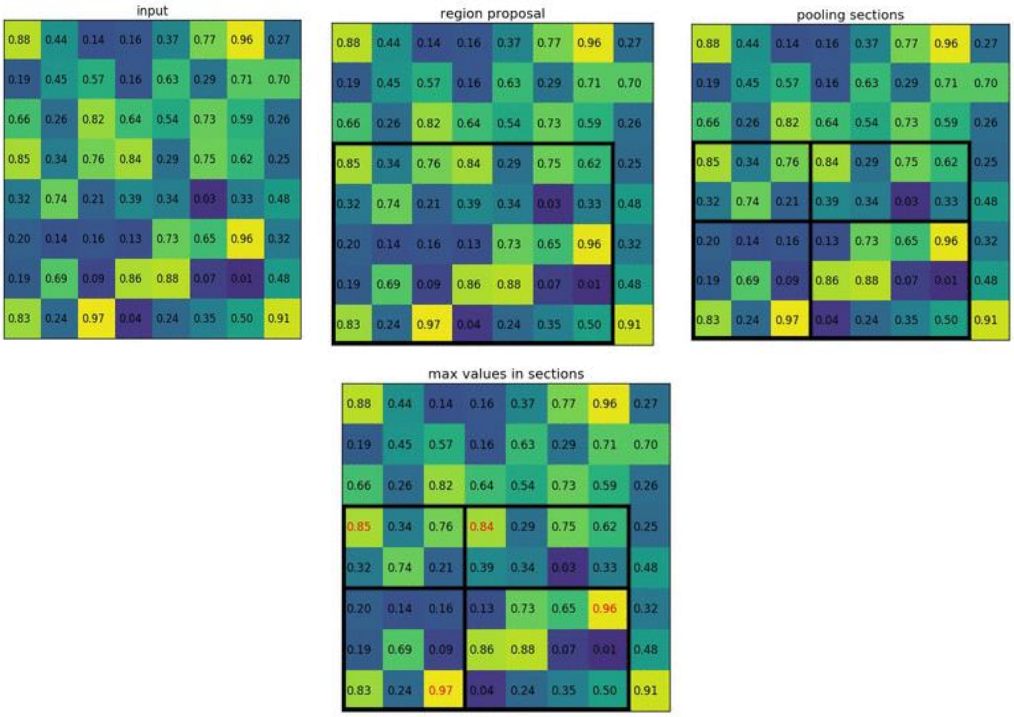
Ini dikenal sebagai down-sampling. Hal ini umum untuk secara berkala menyisipkan lapisan penyatuan antara lapisan konvolusional yang berurutan (masing-masing biasanya diikuti oleh fungsi aktivasi, seperti lapisan ULT) dalam arsitektur CNN. Saat penggabungan layer berkontribusi pada invariansi terjemahan lokal, mereka tidak menyediakan invariansi terjemahan global dalam CNN, kecuali sebuah bentuk penggabungan global digunakan. [4] [60] Lapisan penggabungan biasanya beroperasi secara independen pada setiap kedalaman, atau potongan, dari input dan mengubah ukurannya secara spasial.

Bentuk penggabungan maksimal yang paling umum adalah lapisan dengan filter berukuran 2x2, diterapkan dengan langkah 2, yang mengambil subsampel setiap irisan kedalaman dalam masukan sebesar 2 sepanjang lebar dan tinggi, membuang 75% aktivasi:

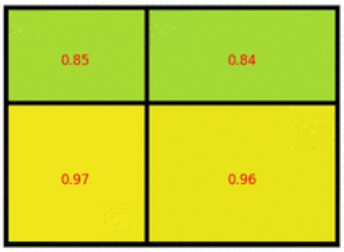
$$f_{X,Y}(S) = \max_{a,b=0}^{1} S_{2X+a,2Y+b}.$$

Gambar 2.12 Rumus dari *Max Pooling*

Dalam hal ini, setiap operasi maksimal lebih dari 4 angka. Dimensi kedalaman tetap tidak berubah (ini juga berlaku untuk bentuk penggabungan lainnya). Selain penggabungan maksimal, unit penggabungan dapat menggunakan fungsi lain, seperti penggabungan rata-rata atau penggabungan $\ell 2$-norm. Penyatuan rata-rata sering digunakan secara historis tetapi baru-baru ini tidak disukai dibandingkan dengan penyatuan maksimal, yang umumnya berkinerja lebih baik dalam praktiknya. Karena efek pengurangan spasial yang cepat dari ukuran representasi, Ada kecenderungan baru-baru ini untuk menggunakan filter yang lebih kecil atau membuang lapisan penyatuan sama sekali. Pooling "Region of Interest" (juga dikenal sebagai pooling RoI) adalah varian dari pooling maks, di mana ukuran output ditetapkan dan persegi panjang input adalah parameternya. Pooling adalah komponen penting dari jaringan saraf konvolusional untuk deteksi objek berdasarkan arsitektur Fast R-CNN.



(a)



(b)

Gambar 2.13 Proses *ROI Pooling* dengan *section* (a) dan output (b)

2.13 ReLU Layer (Lapisan Unit Linier Searah)

ReLU adalah singkatan dari *Rectified Linear Unit*, yang menerapkan fungsi aktivasi non-saturasi f (x) = max (0, x). Ini secara efektif menghilangkan nilai negatif dari peta aktivasi dengan mengaturnya ke nol. Ini memperkenalkan non linier ke fungsi keputusan dan di seluruh jaringan tanpa mempengaruhi bidang reseptif dari lapisan konvolusi.

Fungsi lain juga dapat digunakan untuk meningkatkan nonlinier, misalnya tangen hiperbolik jenuh f (x) = tanh (x), f (x) = | tanh (x) |, dan fungsi sigmoid sigma (x) = (1+) e ^ {- x}) ^ {- 1}. ReLU seringkali lebih disukai daripada fungsi lain karena melatih jaringan saraf beberapa kali lebih cepat tanpa mengurangi akurasi generalisasi.

2.13.1 ReLU Layer Varian Linier

1. Leaky ReLU

Leaky ReLU memungkinkan gradien positif kecil saat unit tidak aktif.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

Gambar 2.14 Rumus Fungsi dari Leaky ReLU

2. Parametric ReLU

Parametric ReLUs (PReLUs) mengambil ide dengan membuat koefisien kebocoran menjadi parameter yang dipelajari bersama dengan parameter jaringan saraf lainnya.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{otherwise.} \end{cases}$$

Note that for a ≤ 1, this is equivalent to

$$f(x) = \max(x, ax)$$

and thus has a relation to "maxout" networks

Gambar 2.15 Rumus Fungsi dari Parametric ReLU

2.13.2 ReLU Layer Varian Non-Linier

1. Gaussian Error Linear Unit (GELU)

GELU adalah perkiraan halus untuk penyearah. Ini memiliki "benjolan" non-monotonik saat x <0, dan berfungsi sebagai aktivasi default untuk model seperti BERT (*Bidirectional Encoder Representations from Transformers*.

$$f(x) = x \cdot \Phi(x), \quad f(x) = x \cdot \Phi(x),$$

Gambar 2.16 Rumus Fungsi dari GELU

2. Sigmoid Linear Unit (SiLU)

SiLU (Satuan Linear Sigmoid) adalah pendekatan halus lainnya.

$$f(x) = x \cdot \text{sigmoid}(x)$$

Gambar 2.17 Rumus Fungsi dari SiLU

3. Softplus

Perkiraan halus untuk penyearah adalah fungsi analitik.

$$f(x) = \ln(1 + e^x),$$

Gambar 2.18 Rumus Fungsi Analitik

Yang disebut fungsi *softplus* atau SmoothReLU. Untuk negatif besar x sekitar e ^ x jadi tepat di atas 0, sedangkan untuk positif besar x sekitar x + e ^ -x jadi tepat di atas x. Parameter ketajaman k dapat disertakan:

$$f(x) = \frac{\ln(1 + e^{kx})}{k}$$

Gambar 2.19 Penambahan parameter k ke Fungsi Analitik

Turunan dari *softplus* adalah fungsi logistik. Mulai dari versi parametrik:

$$f'(x) = \frac{e^{kx}}{1 + e^{kx}} = \frac{1}{1 + e^{-kx}}$$

Gambar 2.20 Versi Parametrik Fungsi Logistik

Fungsi sigmoid logistik adalah perkiraan halus dari turunan penyearah, fungsi langkah *Heaviside*. Generalisasi multivariabel *softplus* variabel tunggal adalah *LogSumExp* dengan argumen pertama disetel ke nol:

$$\mathrm{LSE}(x_1, \ldots, x_n) = \log(e^{x_1} + \cdots + e^{x_n}),$$

Gambar 2.21 Fungsi LogSumExp

Dan gradiennya adalah softmax; softmax dengan argumen pertama disetel ke nol adalah generalisasi multivariabel dari fungsi logistik. Baik *LogSumExp* dan *softmax* digunakan dalam pembelajaran mesin.

4. Exponential Linear Units (ELU)

Unit linier eksponensial mencoba membuat aktivasi rata-rata mendekati nol, yang mempercepat pembelajaran. ELU dapat memperoleh akurasi klasifikasi yang lebih tinggi daripada ReLU.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ a(e^x - 1) & \text{otherwise,} \end{cases}$$

where a is a hyper-parameter to be tuned, and $a \geq 0$ is a constraint.

Gambar 2.22 Fungsi ELU

ELU dapat dilihat sebagai versi smoothed dari ReLU bergeser (SReLU), yang memiliki bentuk f (x) = max (-a, x) dengan interpretasi yang sama dari a.

2.14 Image Recognition (Pengenalan melalui Gambar)

CNN sering digunakan dalam sistem pengenalan gambar. Pada tahun 2012 tingkat kesalahan 0,23% pada database MNIST dilaporkan. Makalah lain tentang penggunaan CNN untuk klasifikasi gambar melaporkan bahwa proses pembelajaran "sangat cepat"; dalam makalah yang sama, hasil publikasi terbaik pada tahun 2011 dicapai dalam database MNIST dan database NORB. Selanjutnya, CNN serupa bernama AlexNet memenangkan ImageNet Large Scale Visual Recognition Challenge 2012.

Ketika diterapkan pada pengenalan wajah, CNN mencapai penurunan tingkat kesalahan yang besar. Makalah lain melaporkan tingkat pengenalan 97,6% pada "5.600 gambar diam lebih dari 10 subjek". CNN digunakan untuk menilai kualitas video secara obyektif setelah pelatihan manual; sistem yang dihasilkan memiliki kesalahan akar kuadrat rata-rata yang sangat rendah.

ImageNet Large Scale Visual Recognition Challenge adalah tolok ukur dalam klasifikasi dan deteksi objek, dengan jutaan gambar dan ratusan kelas objek. Dalam ILSVRC 2014, tantangan pengenalan visual berskala besar, hampir setiap tim berperingkat tinggi menggunakan CNN sebagai kerangka dasar mereka. Pemenang GoogLeNet (Pembuat *DeepDream*) meningkatkan rata-rata presisi deteksi objek menjadi 0,439329, dan mengurangi

kesalahan klasifikasi menjadi 0,06656, hasil terbaik hingga saat ini. Jaringannya menerapkan lebih dari 30 lapisan. Kinerja jaringan saraf konvolusional pada tes ImageNet mendekati kinerja manusia. Algoritma terbaik masih berjuang dengan objek yang kecil atau tipis, seperti semut kecil di tangkai bunga atau orang yang memegang pena bulu di tangan mereka. Mereka juga mengalami masalah dengan gambar yang telah terdistorsi dengan filter, fenomena yang semakin umum terjadi pada kamera digital modern. Sebaliknya, citra semacam itu jarang mengganggu manusia. Namun, manusia cenderung bermasalah dengan masalah lain. Misalnya, mereka tidak pandai mengklasifikasikan objek ke dalam kategori berbutir halus seperti jenis anjing atau spesies burung tertentu, sedangkan jaringan saraf konvolusional menangani hal ini.

Pada tahun 2015, CNN berlapis banyak mendemonstrasikan kemampuan untuk melihat wajah dari berbagai sudut, termasuk terbalik, bahkan ketika tertutup sebagian, dengan kinerja yang kompetitif. Jaringan tersebut dilatih pada database 200.000 gambar yang mencakup wajah di berbagai sudut dan orientasi, serta 20 juta gambar tanpa wajah. Mereka menggunakan kumpulan 128 gambar lebih dari 50.000 iterasi.

2.15 Video Analysis (Analisis Video)

Dibandingkan dengan domain data gambar, hanya ada sedikit pekerjaan untuk menerapkan CNN ke klasifikasi video. Video lebih kompleks daripada gambar karena memiliki dimensi lain (temporal). Namun, beberapa ekstensi CNN ke dalam domain video telah dieksplorasi. Salah satu pendekatannya adalah memperlakukan ruang dan waktu sebagai dimensi yang setara dari input dan melakukan konvolusi baik dalam ruang maupun waktu. Cara lain adalah dengan menggabungkan fitur dari dua jaringan saraf konvolusional, satu untuk spasial dan satu lagi untuk aliran temporal. Unit berulang memori jangka pendek (LSTM) biasanya digabungkan setelah CNN untuk memperhitungkan dependensi antar-frame atau antar-klip. Skema pembelajaran yang tidak diawasi untuk melatih fitur-fitur spatio-temporal telah diperkenalkan, berdasarkan *Convolutional Gated Restricted Boltzmann Machines* dan *Independent Subspace Analysis*.

2.16 PID Controller

PID (*Proportional-Integral-Derivative*) adalah merupakan kontroler mekanisme umpan balik yang biasanya dipakai pada sistem kontrol industri. Sebuah kontroler PID secara kontinu menghitung nilai kesalahan sebagai beda antara setpoint yang diinginkan dan variabel proses terukur. Kontroler mencoba untuk meminimalkan nilai kesalahan setiap waktu dengan penyetelan variabel kontrol, seperti posisi keran kontrol, damper, atau daya pada elemen pemanas, ke nilai baru. Secara spesifik, kontroler PID terdiri dari tiga parameter, Kd diferensial, Ki integral, dan Kp proporsional.

Kombinasi persamaan kontroler sinyal, proporsional, integral, dan diferensial dirumuskan sebagai berikut:

- Kp digunakan untuk merefleksikan error secara proporsional dan menghasilkan sinyal kontrol secepatnya,
- Ki digunakan untuk menghilangkan kesalahan statis, dan
- Kd digunakan untuk mencerminkan tren perubahan kesalahan dan menghasilkan sinyal koreksi awal yang efektif ke sistem.

Tabel 1.1 Efek menaikkan salah satu parameter (atas-bawah)

Parameter	**Rise Time**	**Overshoot**	**Settling**

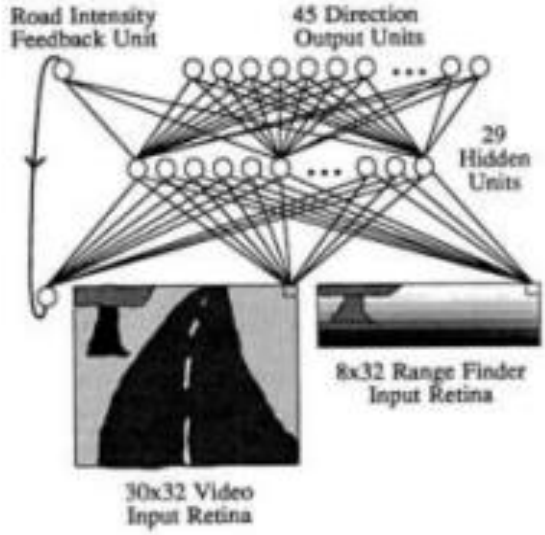
			Time
Kp	Decrease	Increase	Small Change
Ki	Decrease	Increase	Increase
Kd	Minor Decrease	Minor Decrease	Minor Decrease

Parameter	**Steady-State Error**	**Stability**
Kp	Decrease	Decrease
Ki	Decrease Significantly	Decrease
Kd	No Effect	Improves

2.17 Penelitian Terdahulu

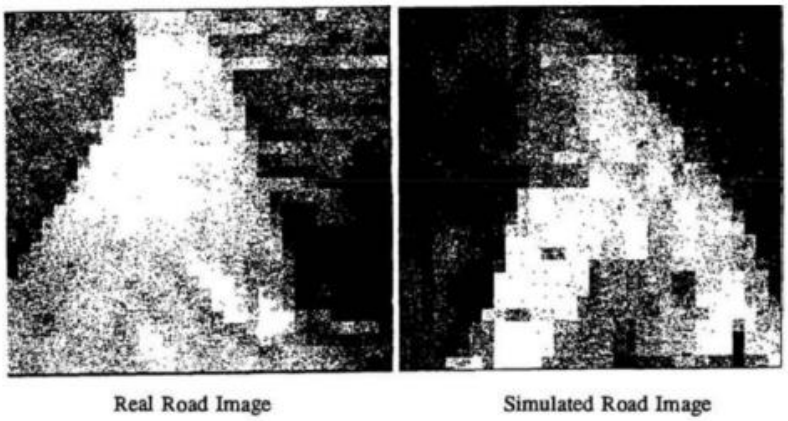
2.17.1 Penelitian oleh Dean A. Pomerleau (1988)

Penelitian ini dilakukan untuk menjawab keterbatasan *Autonomous Navigation Systems* berbasis teknik image processing, dan pengenalan pola secara tradisional yang hanya dapat bekerja optimal pada kondisi spesifik. Penelitian ini meliputi sebuah mobil van yang dikontrol oleh sebuah kecerdasan buatan berarsitektur artificial neural network. Kendaraan ini disebut dengan ALVINN (*Autonomous Land Vehicle In a Neural Network*). Pomerleau menggunakan tiga set data input: dua "retina" dan sebuah intensity feedback unit. Retina pertama berukuran 30x32 satuan, berfungsi untuk menerima video hasil penangkapan kamera dari skenario jalan. Retina kedua berukuran 8x32 satuan, berfungsi untuk menerima input dari sebuah laser pengukur jarak. *Road Intensity Feedback Unit* berfungsi untuk mengindikasi apakah jalanan, dari hasil pengambilan gambar, merupakan daerah yang lebih terang atau lebih gelap dari pada daerah yang bukan merupakan jalan.



Gambar 2.23 Ilustrasi Arsitektur ANN pada ALVINN

Untuk mempermudah proses pengambilan data, dilakukan simulasi generator jalan yang membuat gambar jalan untuk digunakan sebagai contoh pelatihan *neural network*.



Gambar 2.24 Perbandingan gambar jalan hasil penangkapan kamera, dan hasil gambar jalan hasil simulasi generator jalan

Setelah dilakukan eksperimen, didapatkan ALVINN mampu mencapai tingkat akurasi sebesar 90% dimana network dapat bekerja secara akurat pada kecepatan ½ m/s sejauh 400 meter melalui area hutan. Begitu pula pada pengujian dengan kecepatan 1 m/s diperoleh hasil yang serupa.

2.17.2 Penelitian oleh Chirag Goyal, dkk. (2020)

Penelitian ini bertujuan untuk mengidentifikasi berbagai artikel pada sebuah Autonomous Vehicle atau Robot dalam kondisi dan mengelompokkannya dengan menggunakan model CNN dan melalui informasi ini dapat diambil beberapa pilihan berkelanjutan yang dapat digunakan pada *Self Driving Vehicle* atau *Autonomous Car* atau *Driving Assistant System* (DAS).

Penelitian ini bekerja dengan baik dalam skenario waktu nyata untuk Deteksi Objek. Bahkan dapat banyak kelas seperti orang, kendaraan, lampu lalu lintas, hewan. Lalu, selain itu dapat untuk mendeteksi tepi jalur tempat pengemudi bergerak dimana dapat memberi tahu pengemudi jika kendaraan bergerak di jalur dengan benar atau tidak.

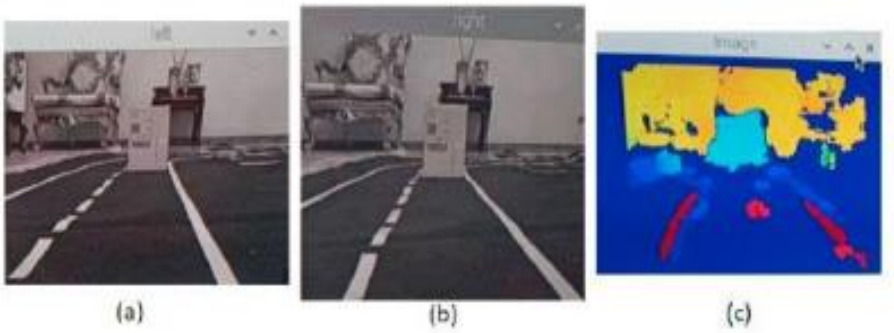
Gambar 2.25 Deteksi Kendaraan dan jalan dengan IoT

2.17.3 Penelitian oleh Mahmoud Fathya, dkk. (2020)

Penelitian ini bertujuan untuk membuat sebuah prototipe mobil self-driving yang mengintegrasikan antara berbagai teknologi termasuk beberapa algoritma yaitu algoritma deteksi jalur jalan, algoritma peta disparitas untuk mendeteksi jarak antara mobil dan kendaraan lain, dan deteksi anomali menggunakan algoritma klasifikasi Support Vector Machine. Untuk menguji prototipe mobil, lingkungan jalan khusus dibangun agar sesuai dengan mobil.

Gambar 2.26 Lintasan Eksperimen

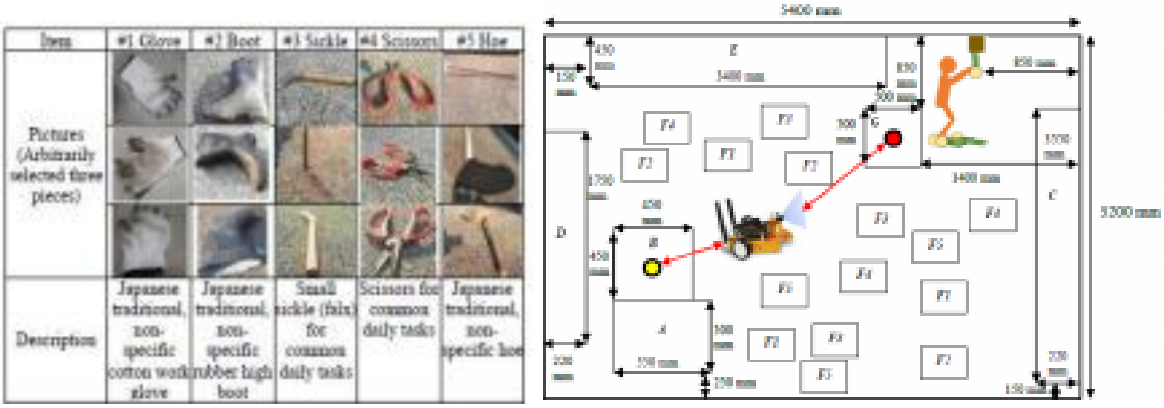
Hasil dari penelitian ini adalah pada pendeteksian lajur, mobil bergerak dengan akurat di jalurnya sesuai dengan sinyal yang dikirimkan ke motor dari algoritma pendeteksi lajur jalan raya. Untuk deteksi anomali, mobil mampu mendeteksi anomali jalan dengan akurasi sebesar 98,6%. Terakhir, untuk hasil pengukuran jarak, mobil mampu mengambil keputusan yang tepat baik bergerak maju, memperlambat, maupun berhenti berdasarkan keluaran algoritma peta disparitas.



Gambar 2.27 (a) Kamera Kiri Video Frame, (b) Kamera Kanan Video Frame, (c) Perbedaan keluaran peta frame foto

2.17.4 Penelitian oleh Shinji Kawakura, dkk. (2020)

Penelitian ini bertujuan untuk mengembangkan sistem mobil self-driving berbasis deep learning untuk mengirimkan barang (misalnya, bawang yang dipanen, peralatan pertanian, botol PET) ke pekerja pertanian di tempat kerja pertanian. Sistem ini didasarkan pada robot berbentuk mobil, NVIDIA Jetbot yang berorientasi kecerdasan buatan (AI). NVIDIA Jetbot dapat menemukan berbagai objek dan menghindarinya.



Gambar 2.28 Objek dan Peta yang akan dilalui NVIDIA JetBot

Penelitian ini terdiri dari enam tahap, yaitu merancang dan mengkonfirmasi validitas seluruh sistem, membangun dan menyetel berbagai pengaturan sistem kecil, mengumpulkan data gambar hambatan, melaksanakan deep learning, melakukan percobaan di gudang untuk mensimulasikan situasi kerja pertanian yang sebenarnya, dan mengevaluasi data uji coba secara kuantitatif dan secara kualitatif. Rasio keberhasilan dari posisi B ke H sebesar 56,2%, dan dari posisi H ke B 69,0%. Namun dalam beberapa kasus, sistem tidak dapat berjalan terutama karena kesalahan deteksi, yaitu Jetbot tidak dapat mendeteksi hambatan dengan cukup cepat sehingga tidak dapat menghindarinya, Jetbot mendeteksi hambatan terlalu dini, dan Jetbot tidak dapat mendeteksi hambatan karena alasan mekanis atau lainnya.

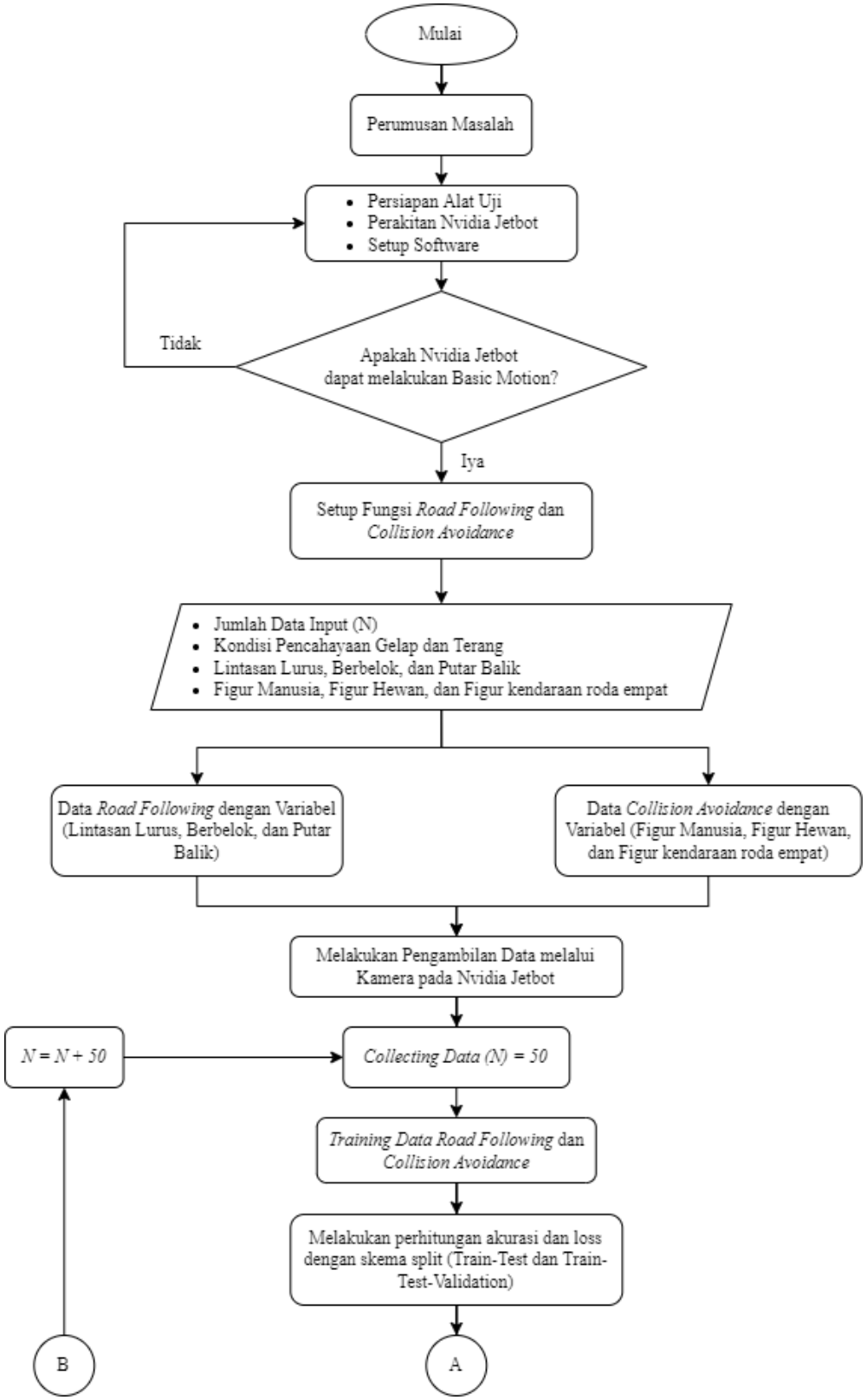
Success ratio (complete success or success) (%)	Trial Time (number)	Collision Count (number)	Collided Objects (Object#, number of times)	Execution Time (s)
(a) from position B to H	5	8.80 (4.23)	#1 0.987 (0.123) #2 0.355 (0.168) #3 2.82 (2.64) #4 1.07 (0.101) #5 3.04 (0.961)	81.8 (33.1)
(b) from position H to B	5	7.29 (6.79)	#1 0.852 (0.556) #2 0.948 (0.388) #3 4.96 (2.21) #4 0.412 (0.280) #5 3.95 (1.36)	123.2 (44.9)

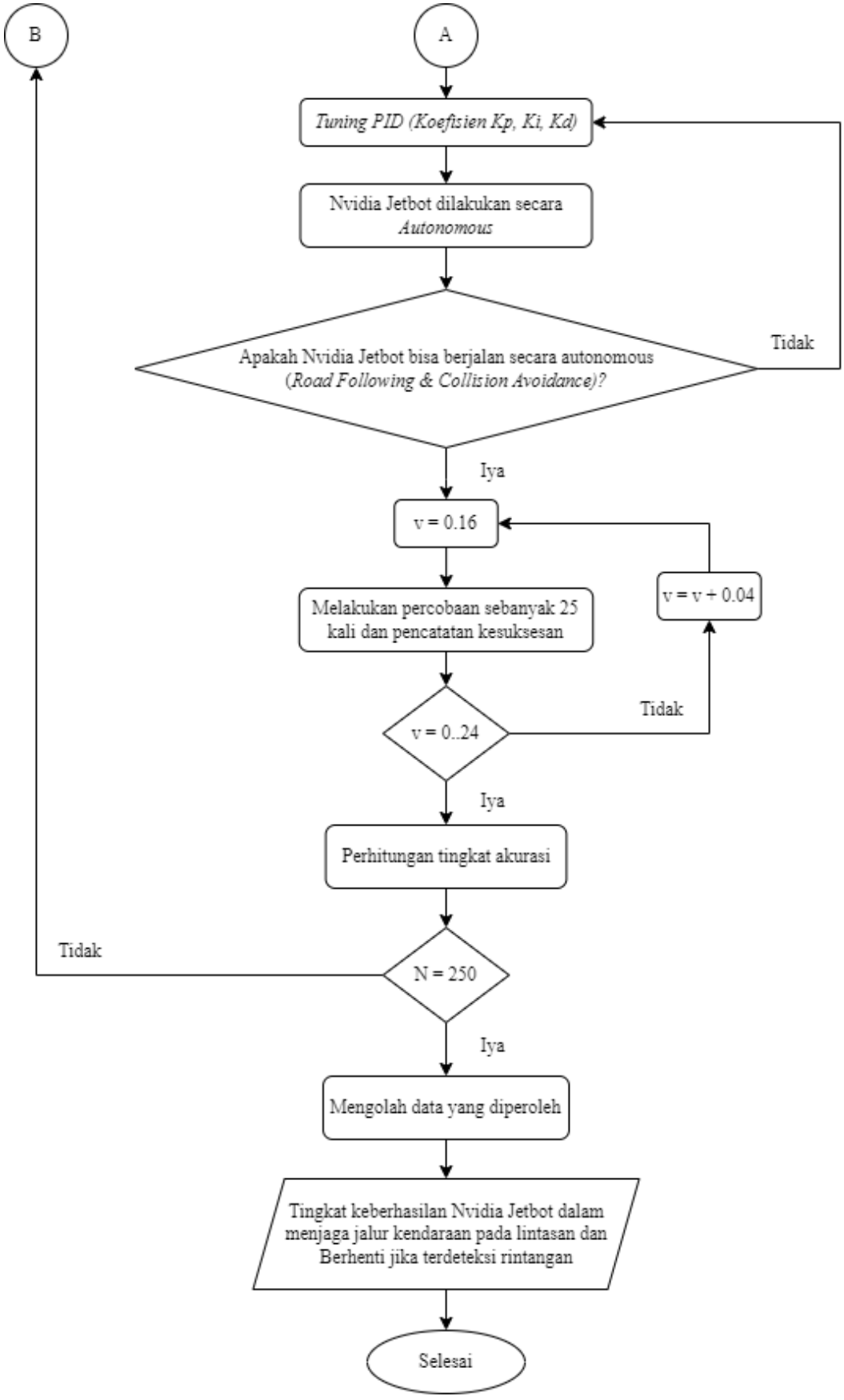
Gambar 2.29 Data hasil eksperimen

“Halaman ini sengaja dikosongkan.”

BAB III
METODE PENELITIAN

3.1 Diagram Alir Penelitian




B
A
Tuning PID (Koefisien Kp, Ki, Kd)
Nvidia Jetbot dilakukan secara Autonomous
Apakah Nvidia Jetbot bisa berjalan secara autonomous (Road Following & Collision Avoidance)?
Tidak
Iya
v = 0.16
v = v + 0.04
Melakukan percobaan sebanyak 25 kali dan pencatatan kesuksesan
v = 0..24
Tidak
Iya
Perhitungan tingkat akurasi
Tidak
N = 250
Iya
Mengolah data yang diperoleh
Tingkat keberhasilan Nvidia Jetbot dalam menjaga jalur kendaraan pada lintasan dan Berhenti jika terdeteksi rintangan
Selesai

Gambar 3.1 Diagram Alir Penelitian

3.2 Persiapan Alat

3.2.1 Alat

Alat yang akan digunakan pada simulasi ini adalah sebagai berikut :

1. Waveshare JetBot AI Kit
   - Merk : Waveshare
   - Type : Waveshare JetBot AI Kit
   - Controller : Jetson Nano
   - Operating system : Ubuntu 18.04 LTS
   - Program Language : Python
   - Camera : 8MP 160° FOV wide angle camera
   - Display : 0.91 inch OLED, 128×32 pixels
   - Wireless NIC : 2.4GHz / 5GHz dual mode WIFI + Bluetooth 4.2
   - Batter Solution : 12.6 V, 18650 batteries with protection
   - Motor : TT motor; reduction rate 1:48; Idle speed 240RPM
   - Chassis : Aluminium alloy chassis
   - Teleoperation : gamepad, webpage
   - Communication : WIFI • Protection : over-current protection, voltage monitoring
2. *Micro SD Card*
3. Monitor Eksternal
4. Kabel HDMI
5. *Keyboard*
6. *Mouse*
7. Lampu Sorot
8. *JupyterLab*

Gambar 3.2 *Waveshare JetBot AI Kit*

3.2.2 Perakitan Alat

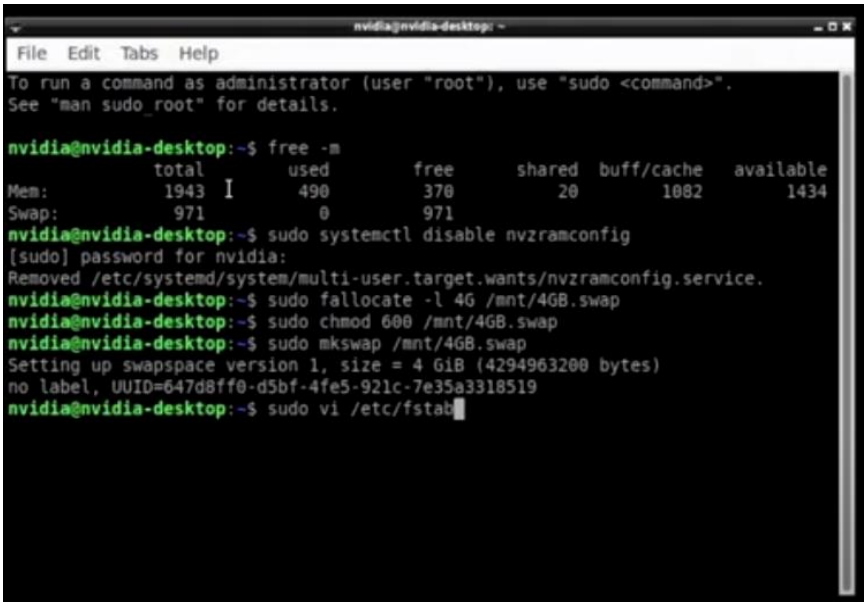
Seluruh bagian Waveshare JetBot AI Kit dirakit sesuai dengan panduan yang ada, dengan bentuk akhir seperti pada gambar 3.1. Adapun bagian-bagian pada JetBot tertera dalam gambar berikut:

Gambar 3.3 Komponen *Waveshare JetBot AI Kit*

3.2.3 Setup Software NVIDIA JetBot dengan Jetpack SDK

Sebelum dapat digunakan, harus dilakukan penyesuaian pada sistem pemrograman JetBot. Adapun langkah-langkah instalasi JetBot sebagai berikut.

1. Melakukan preparasi micro SD card untuk digunakan pada JetBot
2. Memasang micro SD card pada JetBot dimana didalam SD Card tersebut sudah terinstal JetPack SDK untuk Jetson Nano (4GB)
3. Menghubungkan JetBot dengan keyboard, mouse, dan monitor eksternal
4. Melakukan sambungan JetBot dengan memasukkan akun JetBot Default
5. Melakukan sambungan JetBot dengan memasukkan dengan Command Default
6. Menghubungkan JetBot dengan Wifi dengan *Command* yaitu “sudo nmcli device wifi connect <SSID> password <PASSWORD>”
7. Sambungkan JetBot melalui alamat web http://<jetbot_ip_address>:8888 melalui desktop dan melakukan sign in kembali dengan akun JetBot Default



Gambar 3.4 Konfigurasi Operasi *System* pada JetBot

3.3 Data Collecting Collision Avoidance dan Road Following

Penelitian ini, *Collision Avoidance* (rintangan) yang digunakan yaitu menjadi beberapa bentuk yang digunakan sebagai representasi fenomena yang terdapat di jalanan umum di Indonesia. Rintangan digunakan menjadi tiga jenis, yaitu figur manusia, figur hewan, dan figur kendaraan roda empat. Figur manusia akan menjadi representasi untuk Manusia dengan saksi dalam menyebrang jalan. Figur hewan mempresentasikan hewan pada jalanan dengan aksi menyebrang jalan. Dan figur kendaraan roda empat pada hal ini saya menggunakan kubus dengan sisi 4cm sebagai bentuk representasi cone dengan aksi untuk menjadi rintangan yang diam atau statis. Sedangkan *Road Following* digunakan sebagai representasi pada jalur yang terdapat di jalanan umum di Indonesia sebagai bentuk pembatas dari kendaraan agar tetap pada jalur. Rintangan digunakan menjadi tiga jenis, yaitu Lintasan Jurus, Lintasan berbelok, dan lintasan putar balik berbelok. *Data Collecting* akan dilakukan sebanyak 50, 75, 100, dan 125 gambar. Dimana setiap satu uji coba, contohnya Figur manusia akan diambil gambar sebanyak 50 gambar sampai dengan total 125 gambar.

3.3.1 Data Collecting Collision Avoidance

Data Collecting pada *Collision Avoidance* merupakan tahap pengumpulan gambar dengan tujuan untuk mencapai indentifikasi sehingga pengambilan keputusan dalam menghadapi rintangan dengan berbagai bentuk untuk dapat seoptimal mungkin sebagai informasi input. Gambar merupakan hasil penangkapan kamera yang terhubung dimana gambar berisikan kondisi yang terdapat di depan JetBot. Berikut tahap-tahap untuk melakukan *Collecting Data* untuk fungsi *Collision Avoidance*.

1. Menghubungkan robot dengan computer yaitu dengan navigasi ke http://<jetbot_ip_address>:8888 dan melakukan *sign in* dengan Akun default password JetBot.
2. Klik folder dengan nama "*Notebooks*" dan setelah itu klik Kembali folder dengan nama "*Collision_Avoidance"*
3. Buka file dengan nama "data_collection.ipynb" dan mulai menjalankan eksekusi kode tanpa harus dimodifikasi.

4. Setelah menjalankan program kode, JetBot akan otomatis menyalakan kamera dan menghubungkannya ke Komputer yang telah terhubung sehingga akan tertampil tangkapan kamera pada JupyterLab seperti gambar 3.5

```
[1]: import traitlets
     import ipywidgets.widgets as widgets
     from IPython.display import display
     from jetbot import Camera, bgr8_to_jpeg

     camera = Camera.instance(width=224, height=224)

     image = widgets.Image(format='jpeg', width=224, height=224)  # this width and height doesn't necessarily have to match the ca

     camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

     display(image)
```

Gambar 3.5 Aktivasi kamera untuk *Collision Avoidance* pada *JupyterLab*

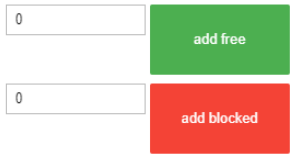
5. Membuat beberapa direktori tempat untuk menyimpan data yang telah diambil dengan membuat folder dari dataset yang berisi dua sub-folder yaitu "*free*" dan "*blocked*", dimana tiap gambar akan disimpan dalam setiap scenario. Proses dapat dilihat pada Gambar 3.6.

```
import os

blocked_dir = 'dataset/blocked'
free_dir = 'dataset/free'

# we have this "try/except" statement because these next functions can throw an error if the directories exist already
try:
    os.makedirs(free_dir)
    os.makedirs(blocked_dir)
except FileExistsError:
    print('Directories not created because they already exist')
```

Gambar 3.6 Proses Pembuatan Direktori Penyimpanan Data *Collecting Collision Avoidance*

6. Membuat dan menampilkan beberapa tombol yang akan digunakan untuk menyimpan snapshot untuk setiap label kelas. Ditambahkan beberapa kotak teks yang akan menampilkan berapa banyak gambar dari setiap kategori yang telah dikumpulkan. Ini berfungsi untuk memastikan bahwa gambar yang diambil dengan label "*free*" mempunyai kuantitas gambar dengan label "*blocked*". Ini juga membantu mengetahui seberapa banyak gambar yang diambil secara keseluruhan.

```
button_layout = widgets.Layout(width='128px', height='64px')
free_button = widgets.Button(description='add free', button_style='success', layout=button_layout)
blocked_button = widgets.Button(description='add blocked', button_style='danger', layout=button_layout)
free_count = widgets.IntText(layout=button_layout, value=len(os.listdir(free_dir)))
blocked_count = widgets.IntText(layout=button_layout, value=len(os.listdir(blocked_dir)))

display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```



Gambar 3.7 Proses Data *Collecting Collision Avoidance* pada *JupyterLab*

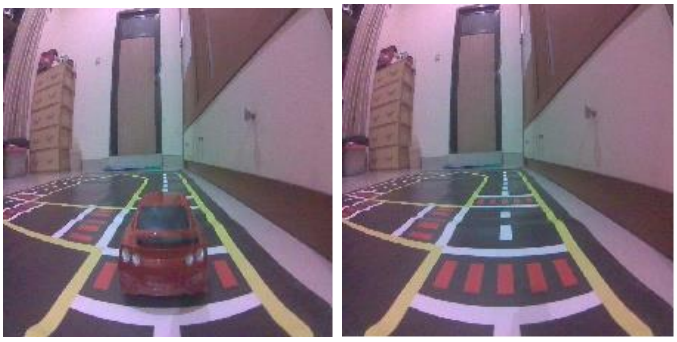
7. Untuk membuat kedua tombol tersebut berfungsi, maka diperlukan melampirkan fungsi penyimpanan gambar untuk setiap kategori ke tombol "*on_click*" *event*. Nilai *Widget* gambar akan tersimpan dalam format *JPEG* terkompresi. Untuk memastikan tidak ada nama file yang terulang, digunakan paket "*uuid*" di *Python*, yang dimana mendefinisikan metode "*uuid1*" untuk menghasilkan pengenal unik. Pengindentifikasi unik ini dihasilkan dari informasi seperti waktu saat ini dan alamat mesin. Dapat dilihat pada gambar 3.8.
8. Mengambil data dengan gambar dimana kondisi rintangan terdapat di depan JetBot. Pengambilan gambar dengan nilai set "*Blocked*" yaitu dilakukan dengan meletakan rintangan dengan jarak 15cm di depan JetBot. Pada kondisi gelap, rintangan akan diletakkan sedikit lebih jauh (kurang lebih 5cm) daripada peletakan rintangan pada kondisi terang. Posisi dan sudut rintangan terhadap JetBot divariasikan untuk menambah tingkat kompleksitas data seperti yang terlihat pada gambar 3.9.

```
from uuid import uuid1

def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_free():
    global free_dir, free_count
    save_snapshot(free_dir)
    free_count.value = len(os.listdir(free_dir))

def save_blocked():
    global blocked_dir, blocked_count
    save_snapshot(blocked_dir)
    blocked_count.value = len(os.listdir(blocked_dir))

# attach the callbacks, we use a 'lambda' function to ignore the
# parameter that the on_click event would provide to our function
# because we don't need it.
free_button.on_click(lambda x: save_free())
blocked_button.on_click(lambda x: save_blocked())
```

Gambar 3.8 Proses Aktivasi Fungsi Penyimpanan Gambar Terkompresi

Gambar 3.9 Proses pengambilan data dengan rintangan di depan JetBot

9. Pengambilan gambar akan dilakukan pada nilai set "*free*" dan "*blocked*" dengan rasio 50 banding 50. Untuk kondisi "*Blocked*" dilakukan pengambilan gambar pada ketiga jenis rintangan (figur manusia, figure hewan, dan benda dengan bentuk sembarang). Dari dua kumpulan nilai set yang akan tersimpan ke dalam *SD Card* dengan variasi pengambilan gambar sebanyak 50, 75, 100, dan 125. Contoh gambar dengan nilai set "*Free*" dan "*Blocked*" hasil penangkapan kamera JetBot pada gambar 3.10.



Gambar 3.10 Bentuk Pengambilan gambar *"Free"* dan *"Blocked"*

10. Dan yang terakhir, disaat semua dataset sudah selesai diambil. Data tersebut dikompres menjadi satu file berbentuk zip. Dan setelah itu mendownload dan mengupload data tersebut ke GPU desktop untuk dilatih mengenai Collision Avoidance.

3.3.2 Data Collecting Road Following

Data Collecting pada *Road Following* merupakan tahap pengumpulan gambar dengan tujuan untuk mencapai indentifikasi sehingga pengambilan keputusan dalam menghadapi jalur dengan jenis berbeda dapat seoptimal mungkin sebagai informasi input. Gambar merupakan hasil penangkapan kamera yang terhubung dimana gambar berisikan kondisi yang terdapat di depan JetBot yang memperlihatkan Jalur Kendaraan. Umumnya semakin banyak data yang diambil, maka semakin baik juga dalam mengenali jalur yang ada, tetapi mengingat keterbatasan memori, dibutuhkan jumlah optimal dimana dengan jumlah data tertentu. Berikut tahap-tahap untuk melakukan *Collecting Data* untuk fungsi *Road Following*.

1. Dimulai dengan mengimpor semua *libraries* yang diperlukan untuk tujuan *Data Collecting*. Terutama menggunakan OpenCV untuk memvisualisasikan dan menyimpan gambar dengan label. *Libraries* seperti *uuid*, *datetime* digunakan untuk *Labelling Gambar*.

```
# IPython Libraries for display and widgets
import ipywidgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Camera and Motor Interface for JetBot
from jetbot import Robot, Camera, bgr8_to_jpeg

# Basic Python packages for image annotation
from uuid import uuid1
import os
import json
import glob
import datetime
import numpy as np
import cv2
import time
```

Gambar 3.11 *Import Libraries*

2. Menampilkan kamera langsung dengan *python widget* (*ipywidget*) yang dinamakan dengan jupyter_clickable_image_widget yang memungkinkan untuk klik gambar dan mengambil koordinat untuk anotasi pada data. Dengan ini tidak diperlukan gamepad untuk anotasi pada data. *Neural Network* ini mengambil gambar 224x224 piksel sebagai input. Kamera diatur ke ukuran itu untuk memimnimalkan ukuran file dari kumpulan data yang akan diambil. Dengan mengeksekusi blok kode pada Gambar 3.12, hal tersebut dapat menampilkan

umpan gambar langsung untuk diklik untuk anotasi di sebelah kiri, serta snapshot dari gambar beranotasi terakhir (dengan lingkaran hijau menunjukkan tempat untuk mengklik) di sebelah kanan. Serta juga menunjukkan jumlah gambar yang telah disimpan. Saat mengklik gambar *live* kiri, file tersebut disimpan di folder "dataset_xy" dengan file Bernama "xy_<*x value*>_<*y value*>_<uuid>.jpg". Disini "<*x value*>" dan "<*y value*>" adalah koordinat dalam piksel (hitung dari sudut kiri atas).

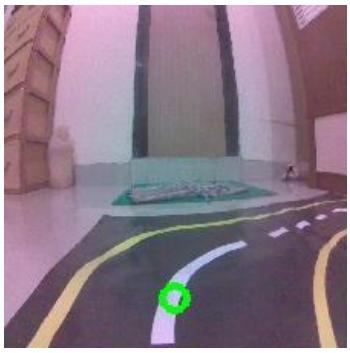
```
from jupyter_clickable_image_widget import ClickableImageWidget

DATASET_DIR = 'dataset_xy'

# we have this "try/except" statement because these next functions can throw an error if the directories exist already
try:
    os.makedirs(DATASET_DIR)
except FileExistsError:
    print('Directories not created because they already exist')

camera = Camera()

# create image preview
camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'), transform=bgr8_to_jpeg)

# create widgets
count_widget = ipywidgets.IntText(description='count')
# manually update counts at initialization
count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']
```

```
        # save to disk
        #dataset.save_entry(category_widget.value, camera.value, x, y)
        uuid = 'xy_%03d_%03d_%s' % (x, y, uuid1())
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(camera_widget.value)

        # display saved snapshot
        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

camera_widget.on_msg(save_snapshot)

data_collection_widget = ipywidgets.VBox([
    ipywidgets.HBox([camera_widget, snapshot_widget]),
    count_widget
])

display(data_collection_widget)
```

Gambar 3.12 *Coding block to Display Live Camera Feed and Collecting Data*

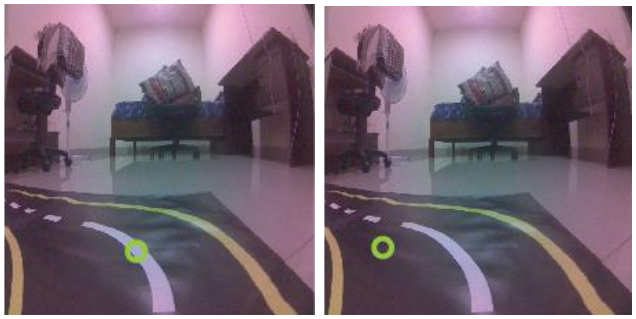
3. Pengambilan data dilakukan dengan meletakkan JetBot pada lintasan yang sudah didesain dengan jenis. Pengambilan data dengan melakukan klik (klik ini menjadi penentu arah gerak dari Nvidia Jetbot) pada titik yang ingin dituju pada gambar yang muncul setelah mengeksekusi Coding Block pada Gambar 3.12. Data gambar yang diambil untuk percobaan ini merupakan perilaku berkendara dari seorang pengendara, dimana data yang diambil yaitu berkembang mulai dari total 50, 75, 100, dan berakhir pada 125 data terbaik untuk membentuk perilaku berkendara dari seorang pengendara. Untuk data terbaik disini adalah pengendara tidak membuat kesalahan dan tetap pada jalur yang ada. Keadaan terbaik juga ditunjukkan pada Gambar 3.14 dimana driver akan berbelok jika sudah dekat dengan belokan. Sama juga dengan putar balik yang diambil dari perilaku pengendara dimana keadaan yang benar yaitu pengendara harus bermanuver sejajar dengan marka garis tepi

dalam. Dan terakhir untuk jalur lurus, data yang benar yaitu pengendara harus bermanuver mengikuti marka garis tepi dalam sama hal nya seperti berputar balik tanpa adanya perpindahan arah. (Gambar 3.13 Bentuk gambarnya kamera ada dengan lingkaran ijo di gambar) (gambar 3.14 bentuk gambar nya sama kayak gambar 3.13 nnya kavin dengan benar dan salah) (Gambar 3.15 bentuk gambarnya dengan lantanas berputar balik benar dan salahnya) (Gambar 3.16 bentuk gambarnya dengan lintasan jalur lurus benar dan salahnya)



Gambar 3.13 Tampilan *Live Camera* dengan penentuan arah

Gambar 3.14 Pengambilan data lintasan berbelok yang benar (kiri) dan salah (kanan)



Gambar 3.15 Pengambilan data lintasan berputar balik yang benar (kiri) dan salah (kanan)

Gambar 3.16 Pengambilan data lintasan jalur lurus yang benar (atas) dan salah (bawah)

4. Setelah data terkumpul sesuai dengan kebutuhan, data tersebut perlu disalin ke Desktop GPU atau Mesin *Cloud* untuk dilakukannya pelatihan. Tanda "!" menunjukkan bahwa kita ingin menjalankan sel sebagai perintah shell (atau terminal). Tanda "-r" pada perintah zip di bawah menunjukkan rekursif sehingga kami menyertakan semua file bersarang (dengan pengaman maksud hal "bersarang"). Dan tanda "-q" menunjukkan diam sehingga perintah zip tidak mencetak output apapun selain tujuan awal.

```
def timestr():
    return str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))

!zip -r -q road_following_{DATASET_DIR}_{timestr()}.zip {DATASET_DIR}
```

Gambar 3.17 *Coding block to Download Data Collection*

5. Dari *Coding Block* sebelumnya akan muncul file Bernama "roadfollowing<Date&Time>.zip" di browser file JupyterLab. Setelah itu melakukan pengunduhan file zip menggunakan browser file JupyterLab dengan mengklik kanan dan memilih unduh.

3.4 Data Training

Data yang telah diperoleh pada tahap collecting data, akan di-training dengan menggunakan model *Deep Residual Net 18* (ResNet18). ResNet18 merupakan CNN yang memiliki 18 layers dimana neural network ini bisa melewati beberapa layers untuk menghindari kesalahan dalam membaca data.

3.4.1 Data Training Collision Avoidance

Data Training pada *Collision Avoidance* merupakan tahap pelatihan data yang sudah diambil pada *Data Collecting* dengan tujuan untuk mencapai identifikasi sehingga pengambilan keputusan dapat lebih efektif dan cepat dalam prosesnya sebagai informasi input. Di tahap ini juga, akan dilatih klasifikasi gambar untuk mendeteksi dua data yaitu "*Free*" dan "*Blocked*". Untuk hal ini, akan digunakan *Deep Learning Library PyTorch*. Berikut tahap-tahap dalam melakukan *Data Training* pada *Collision Avoidance*.

1. Melakukan *import PyTorch Library*.

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
```

Gambar 3.18 *Import Library PyTorch*

2. *Upload* file dengan nama "dataset.zip" yang dibuat pada *Data Collecting* dimana pada *Coding Block* di file "data_collection.ipynb".

```
!unzip -q dataset.zip
```

Gambar 3.19 *Uploading Dataset dari Data Collection*

3. Membuat *dataset instance* dengan menggunakan *ImageFolder dataset class* yang tersedia pada *"torchvision.dataset" package* dan mentransformasikan *"torchvision.transforms"*.

```
dataset = datasets.ImageFolder(
    'dataset',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
```

Gambar 3.20 Pembuatan *Dataset Instance*

4. Membagi *dataset* menjadi *Train Set*, *Test Set* dan *Validation Dataset*. Dimana *Test Set* akan digunakan untuk verifikasi akurasi pada model yang telah dilakukan *Training*. Dan *Test Dataset* sebagai data acuan. Dari *Train dataset* diambil sebanyak 20% secara acak untuk dijadikan *dataset* baru yang Bernama *Validation Dataset*. *Validation Dataset* digunakan untuk menguji apakah *hyperparameter* sudah sesuai atau belum. Sekaligus membuat *data loader* untuk masing-masing *dataset* dengan ukuran *batch* delapan seperti yang terlihat pada gambar berikut.

```
test_percent = 0.25
num_test = int(test_percent * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - num_test, num_test])
```

```
val_percent = 0.20
num_val = int(val_percent * len(train_dataset))
train_dataset, val_dataset = torch.utils.data.random_split(train_dataset, [len(train_dataset) - num_val, num_val])
```

Gambar 3.21 Pembagian Dataset menjadi Train set dan Test set

5. Setelah itu membuat dua *Instance Data Loader*, yang menyediakan utilitas untuk mengacak data, menghasilkan kumpulan gambar, dan memuat sampel secara parallel dengan beberapa pekerja. Pada pengujian ini dilakukan kedua macam pembagian dataset. Yang model pertama adalaha dengan hanya menggunakan train dataset dan test dataset dengan rasio dari 10%-90% sampai 90%-10%. Dan pada model kedua dengan menggunakan train dataset, validation dataset, dan test dataset dengan rasio 80%-10%-10%, 75%-15%-10%, 70%-20%-10%.

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)
val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)
```

Gambar 3.22 Pembuatan *DataLoader* untuk memuat data dalam sejumlah data

6. Mendefinisikan *Neural Network* yang akan dilatih. Dengan *Torchvision Package* menyediakan koleksi model pra-latihan yang dapat digunakan (Model ResNet18 awalnya

dilatih untuk kumpulan data yang memiliki 1000 label kelas, tetapi *dataset* yang digunakan hanya 2 label kelas. Maka dari hal itu lapisan terakhir akan diganti dengan lapisan baru yang tidak terlatih yang hanya memiliki dua keluaran). Dan Setelah itu eksekusi di GPU.

```
model = models.resnet18(pretrained=True)
```

```
model.fc = torch.nn.Linear(512, 2)
```

```
device = torch.device('cuda')
model = model.to(device)
```

Gambar 3.23 *Coding Block* untuk pendefinisian dan pengklasifikasian pada *Neural Network*

7. Setelah itu melatih *Neural Network* untuk 30 epochs. Setelah selesai, file dengan nama "best_model_resnet18.pth" dan lakukan pengunduhan pada file tersebut untuk membuat model pada workstation.

```
NUM_EPOCHS = 30
BEST_MODEL_PATH = 'best_model.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

Gambar 3.24 Melakukan Pelatihan pada Data yang telah didapat

8. Setelah selesai, file dengan nama "best_model_resnet18.pth" di JupyterLab. Setelah itu unduh model ke dalam workstation.

Pelatihan dilakukan dengan pengulangan sebanyak 25 kali dengan sejumlah nilai epoch yang ditentukan. Digunakan *Optimizer* jenis *Stochastic Gradient Descent* (SGD) dengan *loss function* berupa *loss entropy*. Dari proses tersebut diperoleh 25 model baru. Setiap model akan dilakukan validasi untuk mengetahui apakah *hyperparameter* yang digunakan sudah optimal dan dilakukan pengujian kinerja dengan mencocokkan hasil prediksi model dengan acuan pada *Testing Dataset*. Model dengan akurasi validasi dan hasil uji kinerja dengan nilai yang terbaik, akan disimpan dan digunakan untuk menjalankan JetBot dalam melakukan fungsi dalam pemilihan dan keputusan yang lebih baik pada *Collision Avoidance*. Untuk lebih dalamnya terkait kinerja JetBot dalam hasil dan output akan dijelaskan pada Bab IV.

Gambar 3.25 *Output* dari *Training*

3.4.2 Data Training Road Following

Data Training pada *Road Following* merupakan tahap pelatihan data yang sudah diambil pada *Data Collecting* dengan tujuan untuk mencapai indentifikasi sehingga pengambilan keputusan dapat lebih efektif dan cepat dalam prosesnya sebagai informasi input. Di tahap ini juga, akan dilatih dimana ngambil gambar sebagai masukkan (*input*) dan dengan pengeluaran (*output*) sekumpulan nilai x, y yang sesuai dengan target. Untuk hal ini, akan digunakan *Deep Learning Library PyTorch* untuk melatih model, dan arsitektur jaringan saraf *ResNet18* untuk aplikasi pengikut jalan. Berikut tahap-tahap dalam melakukan *Data Training* pada *Road Following*.

1. Melakukan *import PyTorch Library*

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
import glob
import PIL.Image
import os
import numpy as np
```

Gambar 3.26 *Import Library PyTorch*

2. Upload file berikut “road_following_<Date&Time>.zip” yang sebelumnya terbuat dari eksekusi “data_collection.ipynb”

```
!unzip -q road_following.zip
```

Gambar 3.27 *Uploading Dataset* dari *Data Collection*

3. Membuat implementasi “torch.utils.data.Dataset”, dimana implementasi ini mempunya fungsi data kelas “__len__” dan “__getitem__”. Data kelas tersebut bertanggung jawab untuk membuat gambar dan mengurangi nilai x, y dari nama file gambar. Pada hal ini juga, diperlukan fungsi dimana robot dapat mengikuti jalur non-simetris. Dari hal tersebut diperlukan opsi untuk membaca “Flips Horizontal Random” dan juga diperlukan untuk tambahan kumpulan data.

```
def get_x(path, width):
    """Gets the x value from the image filename"""
    return (float(int(path.split("_")[1])) - width/2) / (width/2)

def get_y(path, height):
    """Gets the y value from the image filename"""
    return (float(int(path.split("_")[2])) - height/2) / (height/2)

class XYDataset(torch.utils.data.Dataset):

    def __init__(self, directory, random_hflips=False):
        self.directory = directory
        self.random_hflips = random_hflips
        self.image_paths = glob.glob(os.path.join(self.directory, '*.jpg'))
        self.color_jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]

        image = PIL.Image.open(image_path)
        width, height = image.size
        x = float(get_x(os.path.basename(image_path), width))
        y = float(get_y(os.path.basename(image_path), height))

        if float(np.random.rand(1)) > 0.5:
            image = transforms.functional.hflip(image)
            x = -x

        image = self.color_jitter(image)
        image = transforms.functional.resize(image, (224, 224))
        image = transforms.functional.to_tensor(image)
        image = image.numpy()[::-1].copy()
        image = torch.from_numpy(image)
        image = transforms.functional.normalize(image, [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        return image, torch.tensor([x, y]).float()

dataset = XYDataset('dataset_xy', random_hflips=False)
```

Gambar 3.28 Pembuatan *Dataset Instance*

4. Jika dataset terbaca, dataset akan dilakukan split train dan test sets. Penelitian ini data akan dilakukan split train sets dan test sets mulai dari 10%-90% sampai 90%-10% sebanyak dua kali pengetesan setiap kenaikan atau penurunan 10% dari rasio untuk dicari perbandingan data train sets dan test sets yang optimal. Test sets akan digunakan untuk verfikasi akurasi model yang sudah dilakukan Training.

```
test_percent = 0.1
num_test = int(test_percent * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - num_test, num_test])
```

(a)

```
val_percent = 0.15
num_val = int(val_percent * len(train_dataset))
train_dataset, val_dataset = torch.utils.data.random_split(train_dataset, [len(train_dataset) - num_val, num_val])
```

(b)

Gambar 3.29 *Split dataset into Train Sets with Test Sets (a) and Test sets with Validation Sets (b)*

5. Pembagian data ini dibagi menjadi dua macam pada umumnya, yang model pertama adalah train dataset, validation dataset, dan test dataset, yang model kedua yaitu hanya pada train dataset, dan test dataset. Pada model pembagian data train dataset, validation dataset, dan test dataset, dimana validation dataset digunakan sebagai "fake test data" dan data diambil dari train dataset sebesar 10%-20%. Pada pengujian ini dilakukan kedua macam pembagian dataset. Yang model pertama adalaha dengan hanya menggunakan train dataset dan test dataset dengan rasio dari 10%-90% sampai 90%-10%. Dan pada model kedua dengan menggunakan train dataset, validation dataset, dan test dataset dengan rasio 80%-10%-10%, 75%-15%-10%, 70%-20%-10%.

Gambar 3.30 Pembagian dataset

6. Pembuatan data loaders untuk memuat data pada grup kecil atau dipanggil dengan batches untuk mengacak data dan mengaktifkan multi-subprocesses. Pada penelitian ini digunakan batch size yang digunakan adalah 64. Batch size ini tergantung pada memori yang tersedia pada GPU dan ini akan mempengaruhi akurasi dari model.

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

Gambar 3.31 Pembuatan *data loaders* untuk memuat data dalam grup kecil

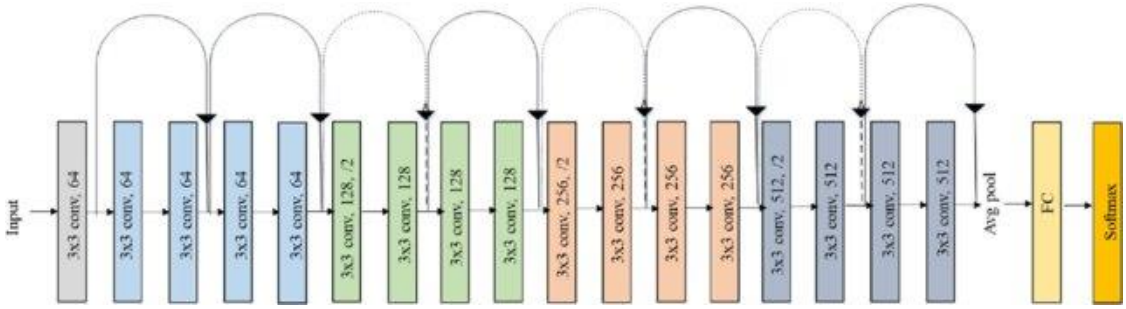
7. Penelitian ini akan digunakan model Neural Network ResNet-18 yang tersedia Pytorch Torchvision. Nvidia JetBot menggunakan transfer learning, yang menggunakan pre-trained model (dilatih dengan jutaan gambar) untuk tugas baru yang mungkin memiliki lebih sedikit data yang tersedia. Model Resnet memiliki lapisan akhir yaitu layer 512 sebagai in_features dan kami akan melatih regresi sehingga out features sebagai 1. Dan setelah itu transfer model akan dieksekusi JetBot.

```
model = models.resnet18(pretrained=True)
```

```
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

Gambar 3.32 Pendefinisian *Neural Network*

8. Pada Transfer Learning, lapisan yang terhubung secara penuh (fc) akan digantikan dengan data yang telah diambil pada tahap data collecting. Gambar 3.33 merupakan layer fc yang belum digantikan. Setelah data tersebut digantikan, maka akan dilakukan pengujian kecocokan transfer learning ini dengan data baru yang akan digunakan sebagai layer fc.



Gambar 3.33 Diagram arsitektur *Resnet18* tanpa replace dari *Data Collection*

Layer Name	Output Size	ResNet-18
conv1	112 × 112 × 64	7 × 7, 64, stride 2
conv2_x	56 × 56 × 64	3 × 3 max pool, stride 2
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	28 × 28 × 128	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	14 × 14 × 256	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	7 × 7 × 512	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
average pool	1 × 1 × 512	7 × 7 average pool
fully connected	1000	512 × 1000 fully connections
softmax	1000	

Gambar 3.34 Arsitektur *Resnet18* dengan *fc replace* dari *Data Collection*

9. Setelah itu akan dilakukan pelatihan dengan format 30 epochs dan menyimpan model terbaik jika loss dikurangi. Jika model sudah dilatih, model akan otomatis menjadi sebuah file dengan nama “best_steering_model_xy.pth” yang dapat dijadikan sebagai referensi untuk demo langsung.

```
NUM_EPOCHS = 70
BEST_MODEL_PATH = 'best_steering_model_xy.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())

for epoch in range(NUM_EPOCHS):

    model.train()
    train_loss = 0.0
    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        train_loss += float(loss)
        loss.backward()
        optimizer.step()
    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        test_loss += float(loss)
    test_loss /= len(test_loader)

    print('%f, %f' % (train_loss, test_loss))
    if test_loss < best_loss:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_loss = test_loss
```

Gambar 3.35 Arsitektur *Resnet18* dengan *fc replace* dari *Data Collection*

3.5 Pengaturan Laju JetBot dan Tuning PID

Sebelum dilakukan simulasi model yang diperoleh, perlu diatur kecepatan laju JetBot saat dijalankan. Pengaturan dapat dilakukan melalui tombol (slider) pengatur kecepatan. Laju JetBot diatur dengan percepatan mulai dari 0.2 sampai dengan 0.4 dengan pertambahan setiap percobaan yaitu sebesar 0.01.

Gambar 3.36 Kontrol PID dari JetBot

Dan pada tahap ini, pandangan yang didapatkan dari kamera Nvidia JetBot yang akan disesuaikan oleh *tuning* PID. PID terdiri dari tiga parameter yaitu, Kd Diferensial, Ki Integral, dan Kp Proporsional yang dapat dilihat di gambar 3.36. Data pengaturan *default* dari JetBot yang digunakan untuk *Collision Avoidance*, yaitu data pada *speed gain* sebesar 0.0. Kp Proporsional dinyatakan oleh *steering gain,* Ki Integral dinyatakan dengan *steering integral*, dan Kd diferensial yang dinyatakan oleh *steering kd*. Semua nilai PID pada Nvidia Jetbot ini memiliki maksimal value sebesar 1.0. Hal ini dikarenakan keterbatasan Nvidia Jetbot dimana robot operation system-nya memiliki set point value tertingginya sebesar 1.0. Sehingga setiap value nilai PID, baik itu Kp, Ki, dan Kd memiliki value kurang dari value tertinggi Nvidia Jetbot, yaitu sebesar 1.0.

Sebelum melakukan tuning PID, diperlukan data transfer function dari permodelan Nvidia Jetbot dengan massa Nvidia Jetbot sebesar 1,1478 kg, koefisien gesek spanduk *polycarbonate* berdasarkan Wikipedia yaitu sebesar 0.31, dan rasio gear sebesar 1:120. Transfer function didefinisikan sebagai berikut.

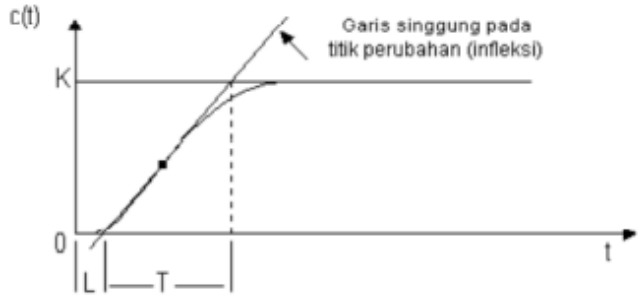
- $input = u(t)$
- $gear\ ratio = -kx(t)$
- $friction = -b\dot{x}(t)$
- $F = ma$
- $u(t) - kx(t) - b\dot{x}(t) = m\ddot{x}(t)$
- $U(s) - kX(s) - bsX(s) = ms^2X(s)$
- $U(s) = X(s)(ms^2 + bs + k)$
- $G_p(s) = \frac{X(s)}{U(s)}$
- $G_p(s) = \frac{1}{ms^2+bs+k}$
- $m = 1{,}1475\ kg$
- $b = 0.31$
- $k = \frac{1}{120}$

Digunakan beberapa metode untuk menentukan nilai Kp, Ki, dan Kd, yaitu dengan menggunakan metode Ziegler-Nichols, metode tuning PID Matlab, dan pengujian dengan menggunakan metode osilasi dimana nilai Kp default dan nilai Ki dan Kd sama dengan nol. Pada metode Ziegler-Nichols digunakan perumusan seperti pada tabel berikut.

Tabel 3.1 Tuning PID metode Ziegler-Nichols

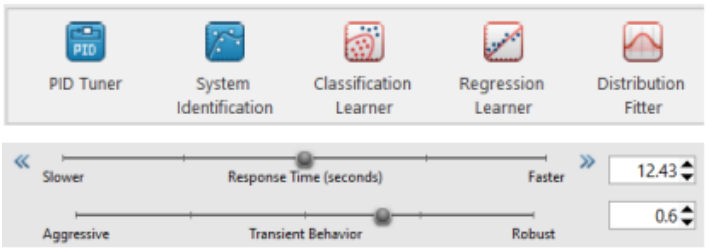
Tipe kontroler	Kp	Ti	Td	Ki	Kd
P	T/L	-	0	-	-
PI	0.92 T/L	L/0.3	0	Kp/Ti	-
PID	1.26 T/L	2L	0.5L	Kp/Ti	Kp*Td

Dimana sebelum melakukan permodelan transfer function dari Nvidia Jetbot dan dimasukkan pada aplikasi Matlab sehingga didapat grafik kurva reaksi berbentuk S. Nilai T dan L didapat dari kurva reaksi berbentuk S seperti pada gambar berikut.



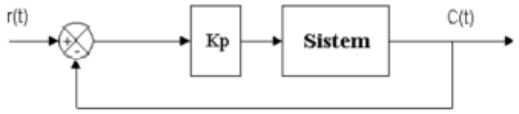
Gambar 3.37 Kurva reaksi

Setelah nilai T dan L didapatkan lalu dimasukkan kedalam perumusan Ziegler-Nichols tersebut sehingga didapatkan nilai Kp, Ki, dan Kd. Pada metode kedua dengan menggunakan PID tuner di aplikasi Matlab seperti pada gambar berikut.

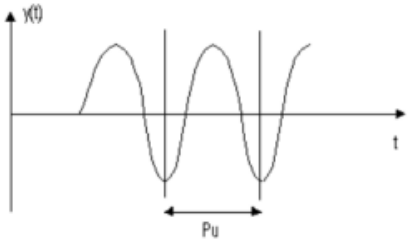


Gambar 3.38 PID *Tuner*

Dan metode yang ketiga adalah metode osilasi yaitu mencari nilai Pu dari grafik yang dihasilkan oleh pengujian nilai Kp default dan nilai Ki dan Kd sama dengan nol dengan grafik seperti pada gambar berikut.



Gambar 3.39 Sistem *close loop* dengan *control proporsional*



Gambar 3.40 Kurva respon osilasi

Setelah didapatkan nilai Pu seperti pada gambar 3.40, maka akan dimasukkan pada perumusan seperti pada tabel 3.2.

Tabel 3.2 Tuning PID metode Osilasi

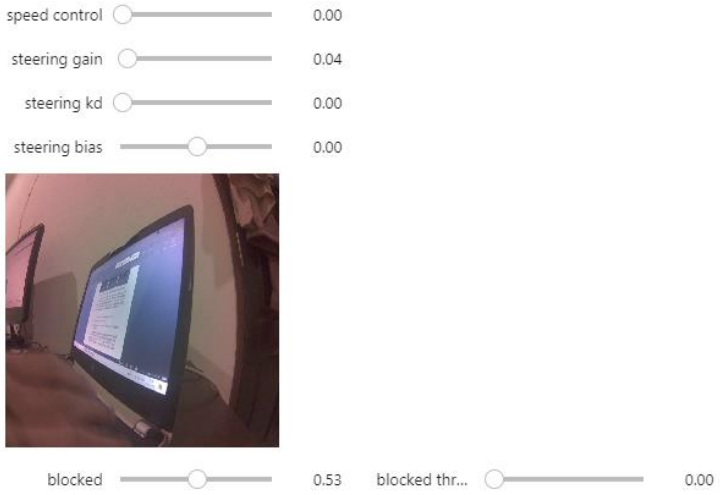
Tipe kontroler	Kp	Ti	Td	Ki	Kd

P	0.62 Ku	-	0	-	-
PI	0.55 Ku	0.55 Pu	0	Kp/Ti	-
PID	0.73 Ku	0.55 Pu	0.2	Kp/Ti	Kp*Td

Tuning PID ini akan gunakan untuk menstabilkan jalan Nvidia Jetbot pada lintasan, dimana digunakan patokan sudut, dimana sudut 0 derajat yaitu pada garis putus-putus di tengah kamera dimana sudut ini ditentukan oleh nilai x dan y yang diambil oleh kamera pada lintasan, dimana nilai x dan y ini merupakan titik yang dihasilkan oleh lingkaran hijau pada kamera Nvidia Jetbot seperti yang ditunjukkan pada gambar 3.41.

Gambar 3.41 Nilai x dan y ditandai dengan lingkaran hijau

Nilai x dan y tersebut akan dicari nilai arctan-nya dan Nvidia Jetbot akan menentukan nilai steering yang ditentukan dengan 3 nilai dibawah dalam identifikasi yaitu *blocked, blocked threshold* sebagai bentuk eksekusi dalam menjalankan JetBot untuk membuat seperti yang ditunjukkan pada gambar 3.42.



Gambar 3.42 Nilai dari tiap *controller*

3.6 Optimization of Trained Model

3.6.1 Optimization of Trained Model for Collision Avoidance

Sebelumnya telah dilakukan pelatihan pada model untuk *Collision Avoidance*. Model yang telah dilatih digunakan untuk mendeteksi apakah robot "*free*" atau "*blocked*" untuk mengaktifkan perilaku dan keputusan dari JetBot untuk menghindari tabrakan pada robot.

Optimisasi ini dilakukan agar model yang mempunyai *gap prediction* dapat terisi sehingga JetBot menjadi lebih baik lagi dalam mengambil keputusan dengan mempertimbang kan *gap prediction sebelum* dilakukan demo langsung. Berikut tahapan-taahapan yang dilakukan untuk mengoptimisasinya.

1. Unduh "best_model_resnet18.pth" ke *workstation*. *Upload* file tersebut ke direktori. Pastikan file tersebut sudah ter-*upload* sebelum melakukan eksekusi kode blok pada gambar berikut.

```
import torch
import torchvision

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
model = model.cuda().eval().half()
```

Gambar 3.43 *Import PyTorch Function for Collision Avoidance*

2. Setelah itu muat trained weights dari file "best_model_resnet18.pth" yang telah diupload.

```
model.load_state_dict(torch.load('best_model_resnet18.pth'))
```

```
device = torch.device('cuda')
```

Gambar 3.44 Pemuatan *Trained Weights for Collision Avoidance*

3. Konversi dan optimalkan model menggunakan torch2trt untuk inferensi yang lebih cepat dengan TensorRT.

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Gambar 3.45 Blok Koding untuk Proses Optimisasi *Collision Avoidance*

4. Melakukan penyimpanan untuk model yang telah dioptimisasi.

```
torch.save(model_trt.state_dict(), 'best_model_trt.pth')
```

Gambar 3.46 Blok Koding untuk menyimpan model *Collision Avoidance* yang telah dioptimisasi

5. Jalankan model yang dioptimalkan dengan mengikut subbab 3.7

3.6.2 Optimization of Trained Model for Road Following

Sebelumnya telah dilakukan pelatihan pada model untuk *Road Following*. Model yang telah dilatih digunakan untuk mendeteksi apakah robot dapat mengikuti jalur dengan baik untuk mengaktifkan perilaku dan keputusan dari JetBot untuk dapat memprediksi dan melakukan posisi yang tepat dalam melakukan perjalanan. Optimisasi ini dilakukan agar model yang mempunyai *gap prediction* dapat terisi sehingga JetBot menjadi lebih baik lagi dalam

mengambil keputusan dengan mempertimbang kan *gap prediction sebelum* dilakukan demo langsung. Berikut tahapan-taahapan yang dilakukan untuk mengoptimisasinya.

1. Unduh "best_steering_model_xy.pth" ke *workstation*. *Upload* file tersebut ke direktori. Pastikan file tersebut sudah ter-*upload* sebelum melakukan eksekusi kode blok pada gambar berikut.

```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
model = model.cuda().eval().half()
```

Gambar 3.47 *Import PyTorch Function for Road Following*

2. Setelah itu muat trained weights dari file "best_steering_model_xy.pth" yang telah diupload.

```
model.load_state_dict(torch.load('best_steering_model_xy.pth'))
```

```
device = torch.device('cuda')
```

Gambar 3.48 Pemuatan *Trained Weights for Road Following*

3. Konversi dan optimalkan model menggunakan torch2trt untuk inferensi yang lebih cepat dengan TensorRT.

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Gambar 3.49 Blok Koding untuk Proses Optimisasi *Road Following*

4. Melakukan penyimpanan untuk model yang telah dioptimisasi.

```
torch.save(model_trt.state_dict(), 'best_steering_model_xy_trt.pth')
```

Gambar 3.50 Blok Koding untuk menyimpan model *Road Following* yang telah dioptimisasi

5. Jalankan model yang dioptimalkan dengan mengikut subbab 3.7

3.7 Eksekusi Demo JetBot Fungsi Collision Avoidance dan Road Following

Pada tahap ini akan dilakukan simulasi model CNN yang sudah didapat dari tahap sebelumnya dengan menggunakan JetBot. JetBot diletakkan pada bidang datar yang cukup luas dengan lintasan yang sudah dibuat sebelumnya. Rintangan-rintangan tersebut disebar secara acak tiap pengujian yang dilakukan. Dari hasil simulasi yang dilakukan, dapat diperoleh data akurasi model deep learning yang sudah dibangun dengan dibuktikan oleh jumlah kegagalan yang dilakukan JetBot. JetBot dikatakan gagal jika menabrak rintangan maupun keluar dari jalur. Berikut tahap-tahap dalam mengeksekusi demo langsung.

1. Telah melakukan seluruh tahap sebelumnya dari *Data Collecting*, *Data Training*, dan *Data Optimization* dari Fungsi *Collision Avoidance* dan Fungsi *Road Following*.
2. Menyimpan file “live_demo_build_resnet18_trt.ipynb” dari Fungsi Collision Avoidance dan Fungsi Road Following di folder yang sama
3. Menggabungkan model regresi dan klasifikasi yang dioptimalkan ke dalam satu notebook sehingga JetBot dapat melakukan pelacakan langsung serta mengaktifkan penghindaran tabrakan pada saat yang sama sehingga dapat menghindari tabrakan dengan rintangan yang datang dalam waktu nyata. Dimulai dengan melakukan pemindahan CPU Memory ke GPU Device.

```
import torch
device = torch.device('cuda')
```

Gambar 3.51 *Importing PyTorch to GPU Device*

4. Unggah Road Following "best_steering_model_xy_trt.pth" yang diperoleh dari "live_demo_build_trt.ipynb" ke direktori notebook ini. Setelah selesai ada file bernama best_steering_model_xy_trt.pth di direktori notebook ini. Dan juga Unggah file model "best_model_trt.pth" yang diperoleh dari "live_demo_resnet18_build_trt.ipnb" ke dalam direktori buku catatan ini. Setelah selesai seharusnya ada file bernama best_model_trt.pth di direktori notebook ini.

```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('best_steering_model_xy_trt.pth')) # well trained road following model

model_trt_collision = TRTModule()
model_trt_collision.load_state_dict(torch.load('best_model_trt.pth')) # well trained collision avoidance model
```

Gambar 3.52 Memuat File TRT Model yang sudah teroptimisasi

5. Konversi dari tata letak HWC ke tata letak CHW. Normalisasikan menggunakan parameter yang sama seperti yang kami lakukan selama pelatihan (kamera kami memberikan nilai dalam rentang [0, 255] dan gambar yang dimuat pelatihan dalam rentang [0, 1] sehingga kami perlu menskalakan 255.0. Transfer data dari memori CPU ke memori GPU. Tambahkan dimensi batch.

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

Gambar 3.53 Memuat *Pre-Processing Function*

6. Melakukan pengaktifan kamera dan membuat robot instance (yang digunakan untuk menjalankan motor)

```
from IPython.display import display
import ipywidgets
import traitlets
from jetbot import Camera, bgr8_to_jpeg

camera = Camera.instance(width=224, height=224, fps=10)
```

```
image_widget = ipywidgets.Image()

traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)
```

```
from jetbot import Robot

robot = Robot()
```

Gambar 3.54 Memuat *Pre-Processing Function*

7. Define Sliders untuk mengontrol JetBot. Mengaktifkan Speed Control Slider (untuk mempercepat kecepatan ataupun melambatkannya), Steering Gain Slider (untuk memperhalus belokan ataupun mempertajamnya), Steering Bias Slider (JetBot condong ke arah ekstrim kanan atau kiri ekstrim trek, Anda harus mengontrol slider ini sampai JetBot mulai mengikuti garis atau trek di tengah), Blocked Slider (Tampilkan probabilitas di mana ada rintangan di depan Jetbot menggunakan model penghindaran tabrakan), Time for stop slider (Untuk secara manual mengatur waktu jetbot harus tetap berhenti setelah objek dipindahkan), dan Blocked Threshold Slider (Untuk secara manual mengatur ambang batas yang diblokir untuk menghentikan Jetbot setelah objek terdeteksi).

```
#Road Following sliders
speed_control_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, description='speed control')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.04, description='steering gain')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001, value=0.0, description='steering kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01, value=0.0, description='steering bias')

display(speed_control_slider, steering_gain_slider, steering_dgain_slider, steering_bias_slider)

#Collision Avoidance sliders
blocked_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, orientation='horizontal', description='blocked')
stopduration_slider= ipywidgets.IntSlider(min=1, max=1000, step=1, value=10, description='time for stop')
blocked_threshold= ipywidgets.FloatSlider(min=0, max=1.0, step=0.01, value=0.8, description='blocked threshold')

display(image_widget)

display(ipywidgets.HBox([blocked_slider, blocked_threshold, stopduration_slider]))
```

Gambar 3.55 Memuat *Slider Control* untuk seluruh fungsi

8. Membuat fungsi yang akan dipanggil setiap kali nilai kamera berubah. Fungsi ini akan melakukan langkah berikut, yaitu Pra-proses gambar kamera, Jalankan model jaringan saraf untuk Mengikuti Jalan dan Penghindaran Tabrakan, Periksa pernyataan if yang memungkinkan Jetbot untuk mengikuti jalan dan berhenti setiap kali rintangan terdeteksi, Hitung perkiraan nilai kemudi, dan Kontrol motor menggunakan kontrol proporsional / turunan (PD).

```
import math

angle = 0.0
angle_last = 0.0
count_stops = 0
go_on = 1
stop_time = 10 # The number of frames to remain stopped
x = 0.0
y = 0.0
speed_value = speed_control_slider.value

def execute(change):
    global angle, angle_last, blocked_slider, robot, count_stops, stop_time, go_on, x, y, blocked_threshold
    global speed_value, steer_gain, steer_dgain, steer_bias

    steer_gain = steering_gain_slider.value
    steer_dgain = steering_dgain_slider.value
    steer_bias = steering_bias_slider.value

    image_preproc = preprocess(change['new']).to(device)

    #Collision Avoidance model:

    prob_blocked = float(F.softmax(model_trt_collision(image_preproc), dim=1).flatten()[0])

    blocked_slider.value = prob_blocked
    stop_time=stopduration_slider.value
```

(a)

```
    if go_on == 1:
        if prob_blocked > blocked_threshold.value: # threshold should be above 0.5
            count_stops += 1
            go_on = 2
        else:
            #start of road following detection
            go_on = 1
            count_stops = 0
            xy = model_trt(image_preproc).detach().float().cpu().numpy().flatten()
            x = xy[0]
            y = (0.5 - xy[1]) / 2.0
            speed_value = speed_control_slider.value
    else:
        count_stops += 1
        if count_stops < stop_time:
            x = 0.0 #set x steering to zero
            y = 0.0 #set y steering to zero
            speed_value = 0 # set speed to zero (can set to turn as well)
        else:
            go_on = 1
            count_stops = 0


    angle = math.atan2(x, y)
    pid = angle * steer_gain + (angle - angle_last) * steer_dgain
    steer_val = pid + steer_bias
    angle_last = angle
    robot.left_motor.value = max(min(speed_value + steer_val, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_value - steer_val, 1.0), 0.0)

execute({'new': camera.value})
```

(b)

Gambar 3.56 *Neural Network Execution Function*

9. Memanggil observe function untuk dapat menjadikan kamera memproses kejadian disekitarnya.

```
camera.observe(execute, names='value')
```

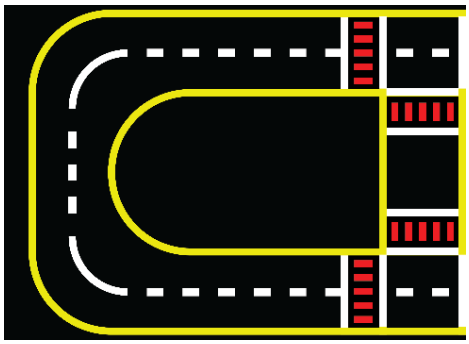
Gambar 3.57 Memanggil *Observe Function*

10. Robot sudah secara otomatis berjalan tanpa harus komando. Jika ingin memberhentikan JetBot dapat melakukan blok kode pada gambar berikut.

```
import time

camera.unobserve(execute, names='value')

time.sleep(0.1)  # add a small sleep to make sure frames have finished processing

robot.stop()
```

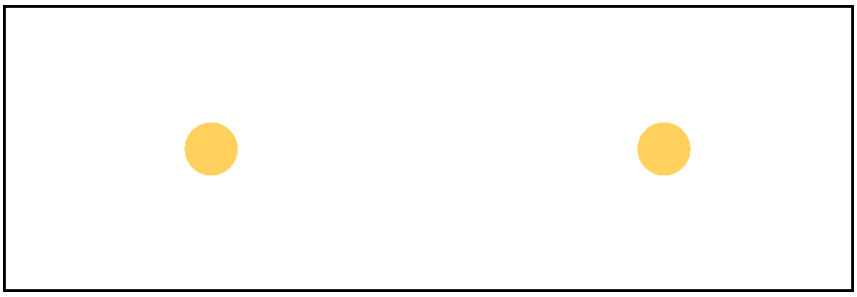
Gambar 3.58 Memanggil fungsi untuk memberhentikan JetBot

3.8 Pengujian Jetbot pada Area yang dengan Kondisi Pencahayaan Terang dan Gelap

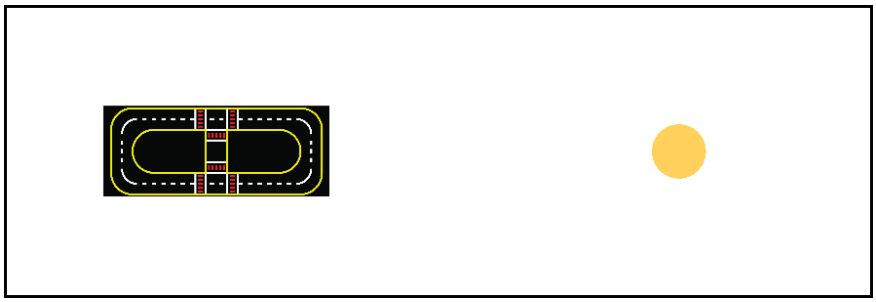


Gambar 3.59 Area Pengujian

Setiap variasi area lintasan akan diuji pada keadaan terang, dan gelap. Pada kondisi gelap, sumber pencahayaan berasal dari lampu sorot berukuran kecil. Pengujian akan dilakukan pada setiap variasi jumlah pengambilan data yang ada yaitu 100, 150, 200, dan 250 dengan kecepatan pada tiap jumlah data yaitu 0.16, 0.2, dan 0.24. Untuk pengambilan uji jetbot dilakukan pada area dengan luas ruangan sebesar 18m^2 dimana pada uji terang dimana menggunakan 2 bohlam yang aktif. Sedangkan pada uji pada pencahayaan gelap hanya menggunakan senter kecil pada satu sisi.



Gambar 3.60 Ruangan Pengujian dengan 2 Bohlam lampu (untuk pengujian kondisi terang)



Gambar 3.61 Penempatan area pengujian pada ruangan dengan 1 Lampu bohlam di sisi berlawanan (untuk pengujian kondisi gelap)

“Halaman ini sengaja dikosongkan.”

BAB IV
HASIL DAN PEMBAHASAN

Data yang telah diambil untuk tugas akhir ini didapatkan dari hasil *training data* yang dioptimisasikan dan didapatkan model terbaik serta dimasukkan pada Nvidia JetBot. Dilakukan uji coba pada fungsi *Collision Avoidance*, *Road Following*, dan Gabungan dari kedua fungsi tersebut. Dimana *Collision Avoidance* dilakukan uji coba terhadap figur manusia, figur hewan, dan figur mobil. Sedangkan fungsi *Road Following* dilakukan uji coba terhadap lintasan lurus, lintasan berbelok, dan lintasan putar balik. Dan yang terakhir, melakukan pengujian dengan menggabungkan kedua fungsi tersebut pada satu kali tes. Ketiga pengujian tersebut dilakukan pada kondisi pencahayaan terang dan gelap.

4.1 Hasil Akhir Data Training

Data foto yang telah dikumpulkan pada fase *Data Collection* akan dijadikan dataset dan dilakukan split data menjadi *train data* dan *test dataset* dengan komposisi *trainset* 10% dan *testset* 90% hingga *trainset* 10%. Setelah split *train-test dataset* telah dilakukan, akan dilakukan split data dengan komposisi *train-validation-test dataset* dengan komposisi 70%-20%-10%, 75%-15%-10%, dan 80%-10%-10%.

4.1.1 Data Training Collision Avoidance

Data training dilakukan pada 50, 75, 100, dan 125 Gambar dengan kondisi pencahayaan terang dan gelap.

Tabel 4.1 Hasil *Training-Test* data pada kondisi terang

Kondisi Terang				
Jumlah	80%-20%		90%-10%	
	Train Accuracy	Test Accuracy	Train Accuracy	Test Accuracy
50 Gambar	1.00	1.00	1.00	1.00
75 Gambar	1.00	1.00	1.00	1.00
100 Gambar	1.00	1.00	1.00	1.00
125 Gambar	1.00	1.00	1.00	1.00

Tabel 4.2 Hasil *Training* data pada kondisi gelap

Kondisi Gelap				
Jumlah	80%-20%		90%-10%	
	Train Accuracy	Test Accuracy	Train Accuracy	Test Accuracy
50 Gambar	1.00	1.00	1.00	1.00
75 Gambar	1.00	1.00	1.00	1.00

100 Gambar	1.00	1.00	1.00	1.00
125 Gambar	1.00	1.00	1.00	1.00

Tabel 4.3 Hasil *Training-Test-Validation* data pada kondisi terang

Kondisi Terang									
	80%-10%-10%			75%-15%-10%			70%-20%-10%		
Jumlah	Train Accuracy	Validation Accuracy	Test Accuracy	Train Accuracy	Validation Accuracy	Test Accuracy	Train Accuracy	Validation Accuracy	Test Accuracy
50 Gambar	1.00	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
75 Gambar	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
100 Gambar	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
125 Gambar	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Tabel 4.4 Hasil *Training-Test-Validation* data pada kondisi gelap

Kondisi Gelap									
	80%-10%-10%			75%-15%-10%			70%-20%-10%		
Jumlah	Train Accuracy	Validation Accuracy	Test Accuracy	Train Accuracy	Validation Accuracy	Test Accuracy	Train Accuracy	Validation Accuracy	Test Accuracy
50 Gambar	1.0000	1.0000	1.0000	1.0000	1.0000	0.6000	1.0000	1.0000	0.9000
75 Gambar	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
100 Gambar	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

125 Gambar	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Dapat diambil bahwa split data terbaik untuk *Training-Testing* ada pada komposisi *Training* 90% dan *Testing* 10%. Dan untuk split data *Training-Validating-Testing* ada pada komposisi *Training* 80%, *Validating* 10%, dan *Testing* 10%.

4.1.2 Data Training Road Following

Data training dilakukan pada 50, 75, 100, dan 125 Gambar dengan kondisi pencahayaan terang dan gelap.

Tabel 4.5 Hasil *Training-Test* data pada kondisi terang

Kondisi Terang				
	80%-20%		90%-10%	
Jumlah	Train Loss	Test Loss	Train Loss	Test Loss
50 Gambar	0.02	0.04	0.03	0.09
75 Gambar	0.01	0.03	0.01	0.02
100 Gambar	0.01	0.01	0.01	0.03
125 Gambar	0.01	0.01	0.01	0.00

Tabel 4.6 Hasil *Training* data pada kondisi gelap

Kondisi Gelap				
	80%-20%		90%-10%	
Jumlah	Train Loss	Test Loss	Train Loss	Test Loss
50 Gambar	0.07	0.11	0.21	0.28
75 Gambar	0.05	0.08	0.08	0.05
100 Gambar	0.04	0.07	0.05	0.06
125 Gambar	0.03	0.06	0.10	0.10

Tabel 4.7 Hasil *Training-Test-Validation* data pada kondisi terang

Kondisi Terang

	80%-10%-10%			75%-15%-10%			70%-20%-10%		
Jumlah	Train Loss	Validation Loss	Test Loss	Train Loss	Validation Loss	Test Loss	Train Loss	Validation Loss	Test Loss
50 Gambar	0.03	0.01	0.01	0.02	0.06	0.05	0.02	0.07	0.04
75 Gambar	0.02	0.01	0.00	0.02	0.03	0.03	0.01	0.02	0.07
100 Gambar	0.01	0.02	0.01	0.01	0.02	0.02	0.02	0.01	0.01
125 Gambar	0.01	0.02	0.02	0.01	0.02	0.02	0.01	0.01	0.01

Tabel 4.8 Hasil *Training-Test-Validation* data pada kondisi gelap

Kondisi Gelap									
	80%-10%-10%			75%-15%-10%			70%-20%-10%		
Jumlah	Train Loss	Validation Loss	Test Loss	Train Loss	Validation Loss	Test Loss	Train Loss	Validation Loss	Test Loss
50 Gambar	0.02	0.01	0.02	0.01	0.01	0.04	0.01	0.01	0.01
75 Gambar	0.02	0.04	0.04	0.02	0.06	0.05	0.02	0.07	0.04
100 Gambar	0.02	0.03	0.01	0.02	0.03	0.02	0.01	0.01	0.16
125 Gambar	0.01	0.02	0.01	0.01	0.02	0.02	0.02	0.01	0.01

Dapat diambil bahwa split data terbaik untuk *Training-Testing* ada pada komposisi *Training* 80% dan *Testing* 20%. Dan untuk split data *Training-Validating-Testing* ada pada komposisi *Training* 80%, *Validating* 10%, dan *Testing* 10%.

4.1 Hasil Tuning PID

Tuning PID dilakukan dengan beberapa metode, yaitu metode Ziegler-Nichols, metode PID tuner Matlab, dan pengujian dengan menggunakan metode osilasi dimana nilai Kp default dan nilai Ki dan Kd sama dengan nol. Dengan transfer function yang telah diperoleh dari permodelan Nvidia Jetbot, yaitu

- $G_p(s) = \frac{1}{1{,}1475ss^2+0.23s+\frac{1}{120}}$

Maka didapatkan hasil dari metode yang ada diatas sebagai berikut.

Tabel 4.9 Hasil *Tuning PID*

Metode	Kp	Ki	Kd
Ziegler-Nichols	17.312	2.431	28.054
PID Tuner Matlab	0.078	0.023	0.213
Osilasi	0.515	0.087	0.218

4.2 Hasil Simulasi Nvidia JetBot pada fungsi Collision Avoidance

Simulasi Nvidia Jetbot dengan menggabungkan fungsi *Collision Avoidance* yang dilakukan pada figur manusia, figure hewan, dan figur mobil dan fungsi *Road Following* yang dilakukan pada lintasan lurus, lintasan putar balik, dan lintasan berbelok sebanyak 25 kali pada 2 variabel yang bersamaan yaitu kecepatan dan jumlah data. Simulasi ini dilakukan pada 2 kondisi pencahayaan yang terang dan gelap. Dan dilakukan dengan 2 dataset koleksi gambar dari kondisi terang dan kondisi gelap. Dan total pengujian Bersama kecepatan yang dimulai dari 0.16 hingga 0.24 yaitu sebanyak 75 kali pengujian untuk satu variabel jumlah data gambar pada 1 kondisi pengujian dan 1 dataset. 1 kali pengujian dikatakan berhasil jika JetBot berada pada lintasan dan tidak keluar jalur serta JetBot berhasil berhenti disaat ada figure yang ada. Sebaliknya, 1 pengujian dikatakan gagal jika JetBot keluar dari lintasan atau JetBot tidak berhenti atau menabrak figur yang ada disekitarnya. Dengan pengujian yang pertama yaitu dilakukan dengan dataset gambar dari koleksi gambar dengan kondisi terang diuji pada kondisi terang, pengujian kedua dilakukan dengan kondisi gelap tetapi dataset gambar dari koleksi gambar dengan kondisi terang dan pengujian terakhir dilakukan pada kondisi gelap dengan dataset gambar dari koleksi gambar dengan kondisi gelap.

4.2.1 Hasil Simulasi pada Kondisi Terang dengan Gambar Dataset dari Kondisi Terang

Melalui perhitungan waktu tempuh JetBot saat dijalankan pada trek lurus sejauh 100 cm dan metode interpolasi dengan mengasumsi kenaikan kecepatan actual dengan sifat linear terhadap kenaikan level kecepatan, diperoleh data kecepatan aktual JetBot untuk masing-masing level kecepatan sebagai berikut.

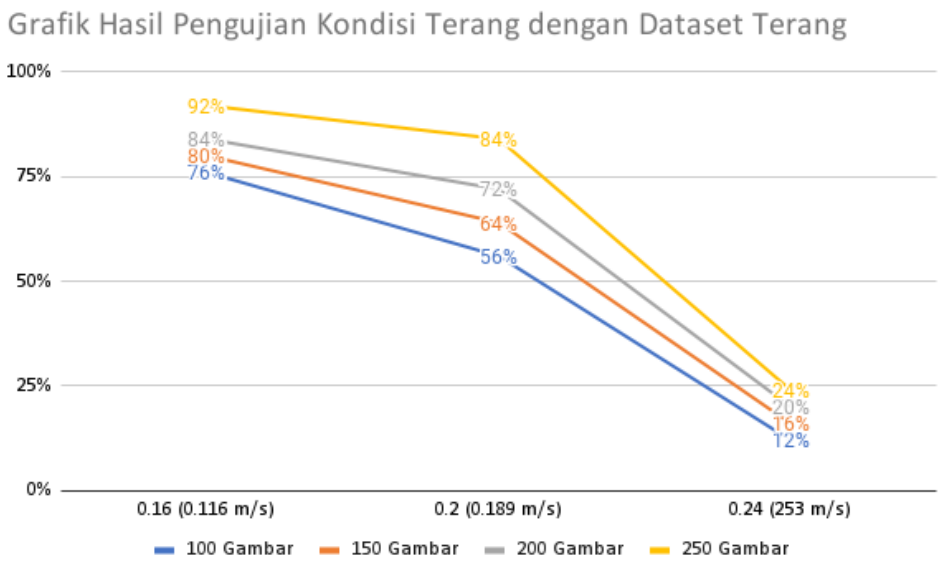
Tabel 4.10 Kecepatan aktual JetBot pada setiap level kecepatan

Level Kecepatan	Kecepatan Aktual (m/s)
0.16	0.116
0.2	0.189
0.24	0.253

Tabel 4.11 Hasil Simulasi Nvidia JetBot pada uji penggabungan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan terang dengan dataset gambar kondisi terang

Uji Kondisi Terang dengan Dataset Kondisi Terang								
Jumlah	100		150		200		250	
Kecepatan	S	G	S	G	S	G	S	G
0.16	19	6	20	5	21	4	23	2
0.2	14	11	16	9	18	7	21	4
0.24	3	22	4	21	5	20	6	19
S = Sukses dan G = Gagal								

Gambar 4.1 Nvidia JetBot pada uji penggabungan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan terang



Gambar 4.2 Perbandingan Grafik Hasil Simulasi fungsi *Collision Avoidance* dan *Road Following* kondisi pencahayaan terang dengan dataset kondisi terang

Untuk total keberhasilan pada pencahayaan terang yaitu pada kecepatan 0.16 didapatkan keberhasilan tertinggi didapatkan pada data 200 gambar dengan tingkat keberhasilan mencapai

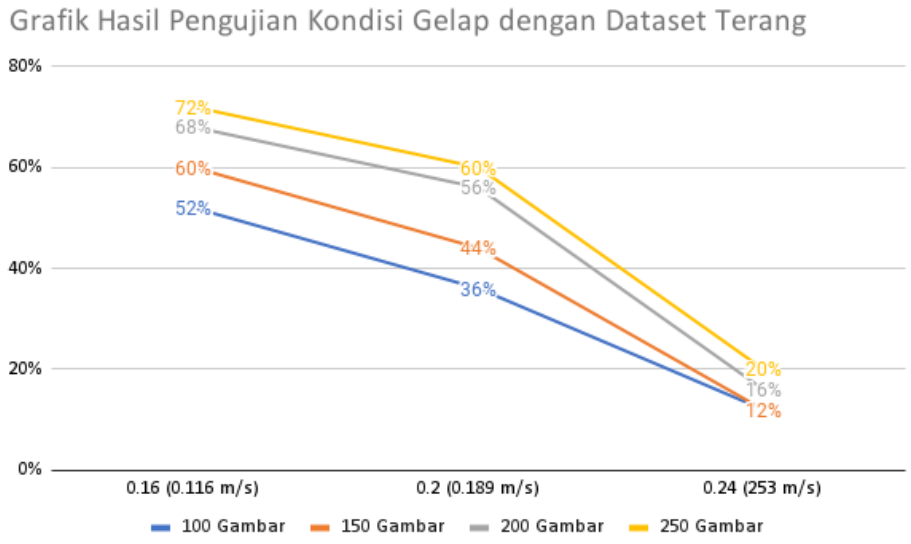
92%, sedangkan untuk tingkat keberhasilan terendah didapatkan pada kecepatan 0.24 pada data 100 gambar yaitu sebesar 12%. Tetapi hampir semua yang berada di kecepatan 0.24 mempunyai tingkat keberhasilan dibawah 30%.

4.2.2 Hasil Simulasi pada Kondisi Gelap dengan Gambar Dataset dari Kondisi Terang

Tabel 4.12 Hasil Simulasi Nvidia JetBot pada uji penggabungan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan gelap dengan dataset gambar kondisi terang

Uji Kondisi Gelap dengan Dataset Kondisi Terang								
Jumlah	100		150		200		250	
Kecepatan	S	G	S	G	S	G	S	G
0.16	13	22	15	10	17	8	18	7
0.2	9	16	11	14	14	11	15	10
0.24	3	22	3	22	5	20	5	20
S = Sukses dan G = Gagal								

Gambar 4.3 Nvidia JetBot pada uji penggabungan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan gelap



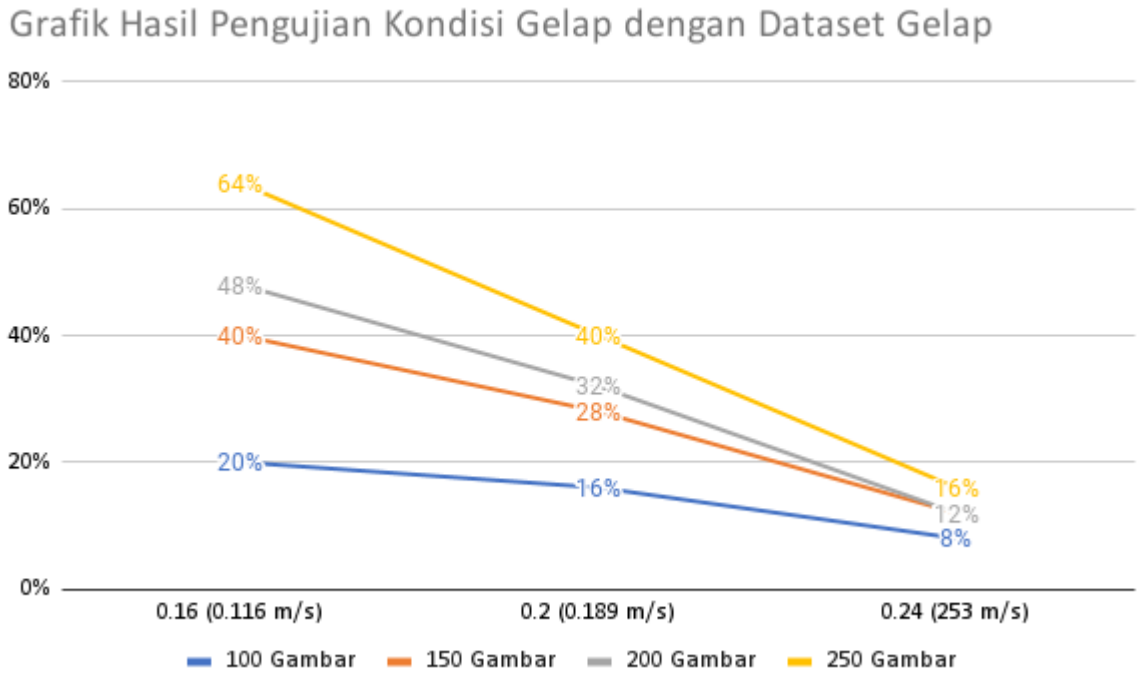
Gambar 4.4 Perbandingan Grafik Hasil Simulasi fungsi *Collision Avoidance* dan *Road Following* kondisi pencahayaan gelap dengan dataset kondisi terang

Untuk total keberhasilan pada pencahayaan terang yaitu pada kecepatan 0.16 didapatkan keberhasilan tertinggi didapatkan pada data 250 gambar dengan tingkat keberhasilan mencapai 72%, sedangkan untuk tingkat keberhasilan terendah didapatkan pada kecepatan 0.24 pada data 100 gambar yaitu sebesar 12%. Untuk yang diatas 60% yaitu hanya pada kecepatan 0.16 untuk semua dataset gambar.

4.2.3 Hasil Simulasi pada Kondisi Gelap dengan Gambar Dataset dari Kondisi Gelap

Tabel 4.13 Hasil Simulasi Nvidia JetBot pada uji penggabungan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan gelap dengan dataset gambar kondisi gelap

Uji Kondisi Gelap dengan Dataset Kondisi Gelap								
Jumlah	100		150		200		250	
Kecepatan	S	G	S	G	S	G	S	G
0.16	5	20	10	15	12	13	16	9
0.2	4	21	7	18	8	17	10	15
0.24	2	22	3	22	3	22	4	21
S = Sukses dan G = Gagal								



Gambar 4.5 Perbandingan Grafik Hasil Simulasi fungsi *Collision Avoidance* dan *Road Following* kondisi pencahayaan gelap dengan dataset kondisi gelap

Untuk total keberhasilan pada pengujan ini yaitu hanya terdapat pada kecepatan 0.16 didapatkan keberhasilan tertinggi didapatkan pada data 250 gambar dengan tingkat

keberhasilan mencapai 64%, sedangkan sisanya tidak ada yang mencapai lebih dari ≥ 60%. Yang mendekati standar keberhasilan hanya pada kecepatan 0.16 data 200 gambar dengan keberhasilan 42%.

4.3 Pembahasan Hasil

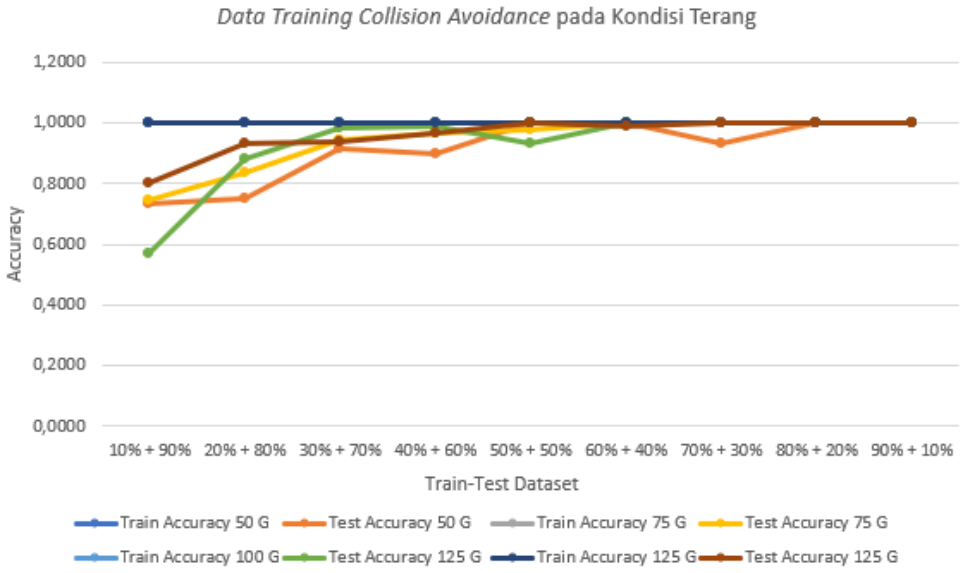
Pada semua subbab sebelum subbab ini diberikan hasil uji coba dan pada kali ini akan dibahas satu per satu mulai dari *Training Data* hingga Pengujian pada fungsi gabungan *Collision Avoidance* dan *Road Following*.

4.3.1 Pembahasan Hasil Data Training

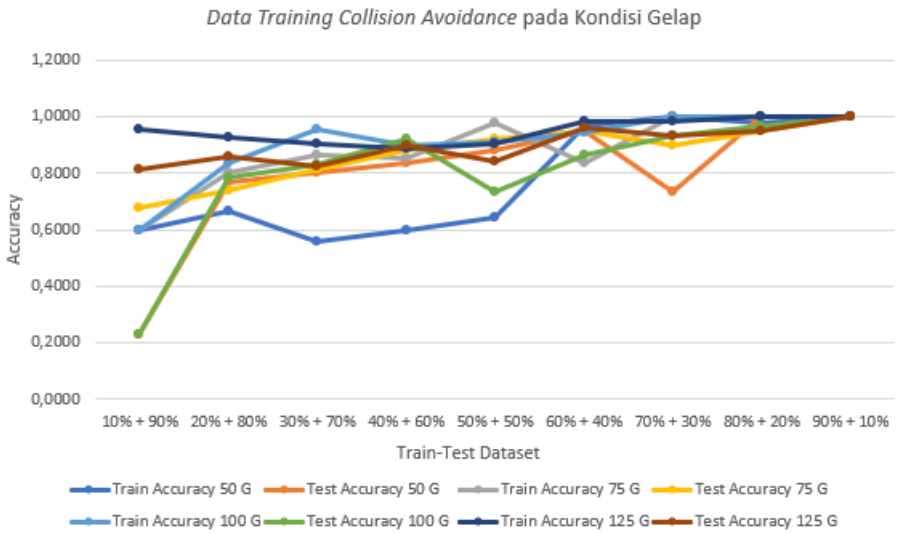
Dilakukan percobaan *Data Training* yaitu dengan split data *train-test dataset*. Dimana komposisi *train-test* mulai dari 10%-90% sampai 90%-10%.

a. Collision Avoidance Data Training

Pada percobaan *Data Training* didapatkan grafik seperti yang ditunjukkan pada gambar berikut.



Gambar 4.6 Grafik Data Training pada fungsi *Collision Avoidance* kondisi pencahayaan terang dengan datasetnya



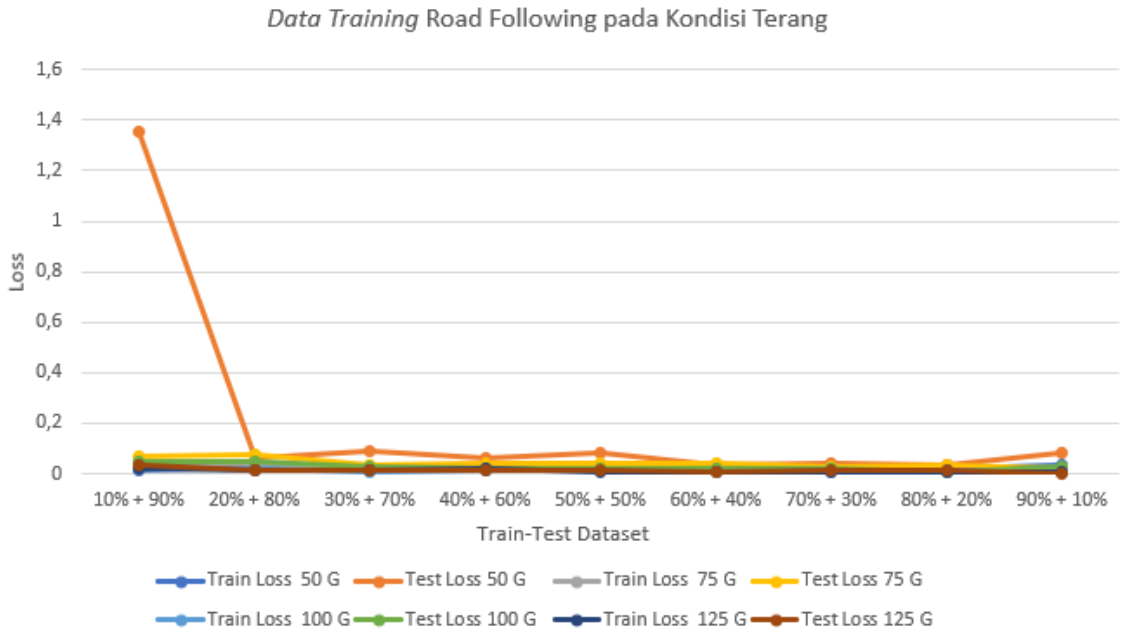
Gambar 4.7 Grafik Data Training pada fungsi *Collision Avoidance* kondisi pencahayaan gelap dengan datasetnya

Dari gambar diatas untuk pencahayaan terang didapatkan kenaikan yang cukup besar antara *train-test dataset* 30%-70%. Dan juga terjadi penurunan cukup kecil pada *train-test dataset* 50%-50% sampai 70%-30%. *Test accuracy* terbesar dan stabil terdapat pada *train-test dataset* 90%-10% dengan *test accuracy* sebesar 100% dan *train accuracy* sebesar 100%. Dan untuk pencahayaan gelap didapatkan kenaikan yang cukup besar pada *train-test dataset* antara 10%-90% sampai 20%-80%. Dan juga terjadi penurunan cukup kecil pada *train-test dataset* 60 %-40% sampai 70%-30%. *Test accuracy* terbesar dan stabil terdapat pada *train-test dataset* 80%-20% dengan *test accuracy* sebesar 96.78% dan *train accuracy* sebesar 99.38%.

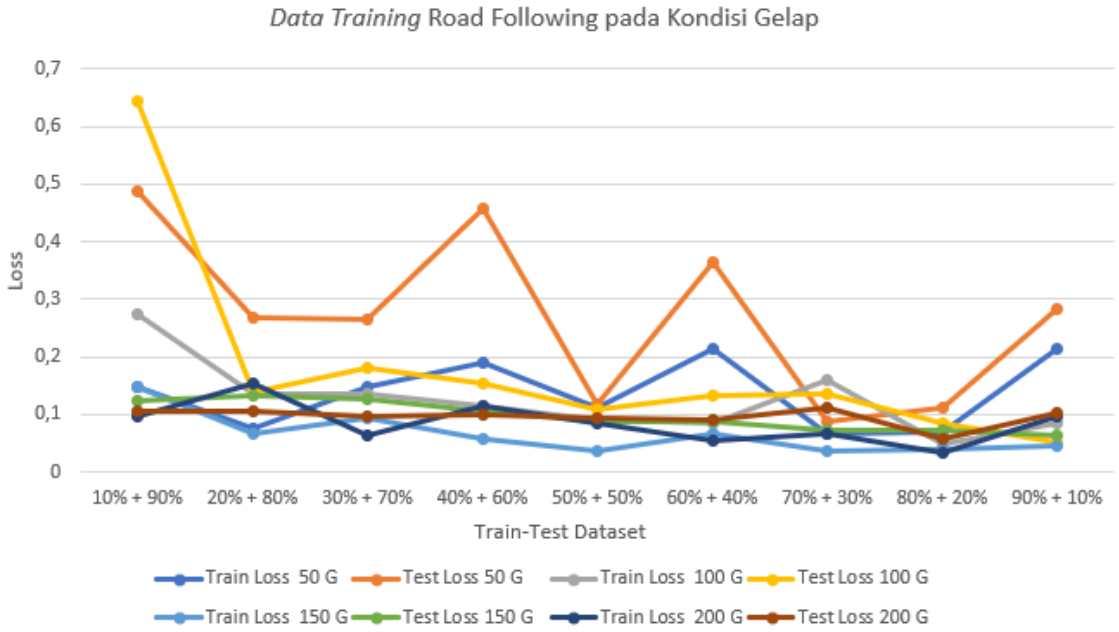
Dari percobaan ini untuk pencahayaan terang dapat digunakan *train-test dataset* 90%-10% untuk *Data Training* sehingga *best model* untuk *Collision Avoidance* yang digunakan merupakan model terbaik untuk melakukan fungsinya. Dan untuk pencahayaan gelap dapat digunakan *train-test dataset* 90%-10% untuk *Data Training* sehingga *best model* untuk *Collision Avoidance* yang digunakan merupakan model terbaik untuk melakukan fungsinya

b. Road Following Data Training

Pada percobaan *Data Training* didapatkan grafik seperti yang ditunjukkan pada gambar berikut.



Gambar 4.8 Grafik Data Training pada *fungsi Road Following* kondisi pencahayaan terang dengan datasetnya



Gambar 4.9 Grafik Data Training pada fungsi *Road Following* kondisi pencahayaan gelap dengan datasetnya

Dari gambar diatas untuk pencahayaan terang didapatkan kenaikan yang cukup besar antara *train-test dataset* 20%-80%. Dan juga terjadi penurunan cukup kecil pada *train-test dataset* 30%-70% sampai 50%-50%. *Test loss* terkecil dan stabil terdapat pada *train-test dataset* 80%-20% dengan *test loss* rata-rata sebesar 7.84% dan *train loss* sebesar 5.06%. Dan untuk pencahayaan gelap didapatkan kenaikan yang cukup besar pada *train-test dataset* 80%-20% dan 50%-50%. Dan juga terjadi penurunan cukup besar pada *train-test dataset* 40%-60% dan 60%-40%. *Test loss* terbesar dan stabil terdapat pada *train-test dataset* 80%-20% dengan *test loss* sebesar 28.24% dan *train loss* sebesar 18.77%.

Dari percobaan ini untuk pencahayaan terang dapat digunakan *train-test dataset* 80%-20% untuk *Data Training* sehingga *best model* untuk *Road Following* yang digunakan merupakan model terbaik untuk melakukan fungsinya. Sama halnya untuk pencahayaan gelap dapat digunakan *train-test dataset* 80%-20% untuk *Data Training* sehingga *best model* untuk *Collision Avoidance* yang digunakan merupakan model terbaik untuk melakukan fungsinya.

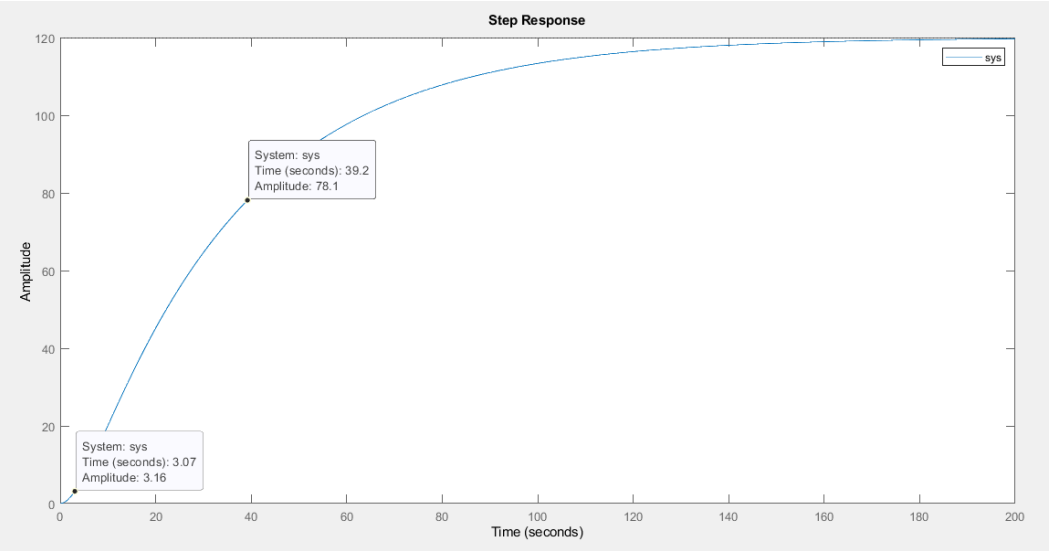
4.3.2 Pembahasan Hasil Data Tuning

Dari ketiga metode, yaitu Ziegler-Nichols, PID Tuner, dan osilasi didapatkan hasil sebagai berikut.

```
>> s=tf('s');
sys=1/[1.1475*s^2+0.31*s+(1/120)]

figure
step(sys);

L=3.07
K=120;
T=39.2-L;

Kp=1.26*(T/L);
Ki=Kp/(2*L);
Kd=Kp*0.5*L;

PID_Controller=pid(Kp,Ki,Kd)

openSys=series(PID_Controller,sys);

closeSys=feedback(openSys,1);
figure
step(closeSys)
```

Gambar 4.10 *Input Transfer Function* pada Matlab

Pada metode Ziegler-Nichols dari transfer function pada program Matlab yang ditunjukkan pada gambar 4.32 didapatkan grafik sistem seperti yang ditunjukkan pada gambar berikut.



Gambar 4.11 Grafik hasil *Transfer Function System (Ziegler-Nichols)*

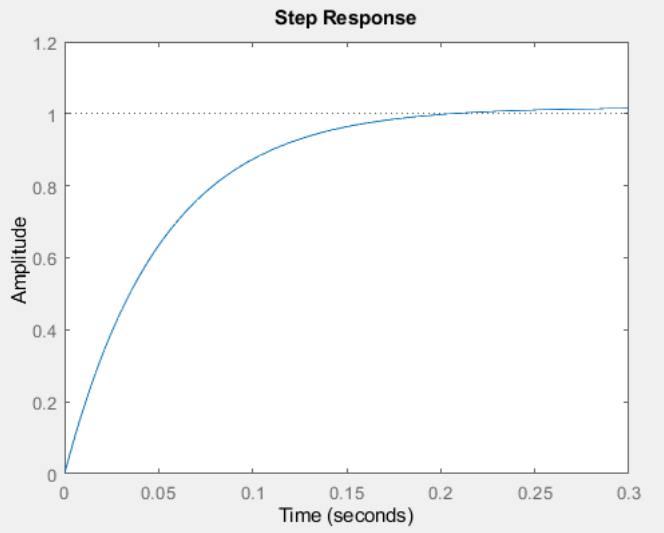
Dari grafik tersebut didapatkan nilai L sebesar 3,07 dan nilai T sebesar 36,13. Kemudan nilai L dan T dimasukkan pada perumusan pada tabel 3.2 dan didapatkan nilai Kp, Ki, dan Kd seperti yang ditunjukkan pada tabel berikut.

Tabel 4.14 Hasil *Tuning PID* dengan Metode *Ziegler-Nichols*

Tipe kontroler	Kp	Ti	Td	Ki	Kd
P	11.77	~	-	-	-
PI	10.83	10.23	-	1.06	-

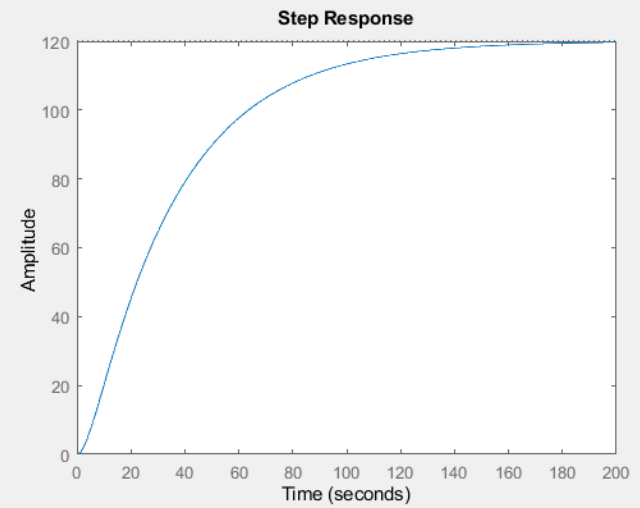
PID	14.83	6.14	1.54	2.42	22.76

Dari metode tersebut, setelah nilai tuning PID dimasukkan ke dalam sistem akan menghasilkan grafik seperti pada gambar berikut.

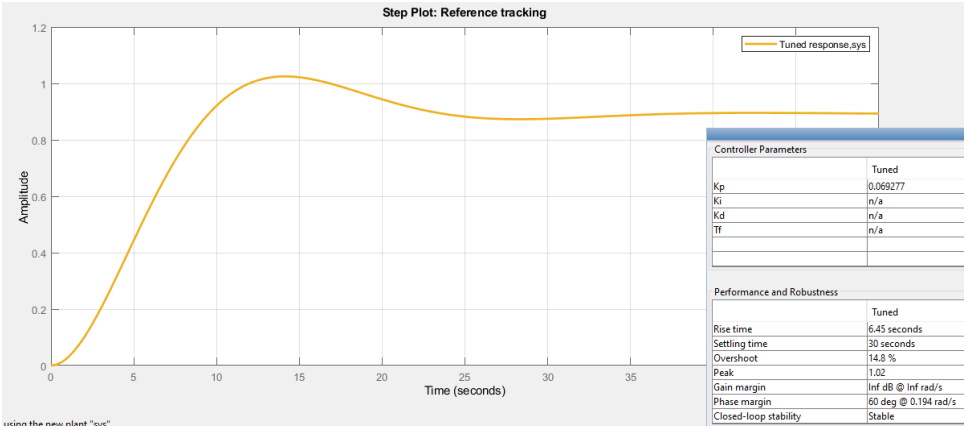


Gambar 4.12 Grafik hasil setelah *Tuning PID (Ziegler-Nichols)*

Pada metode PID tuner dari transfer function didapatkan grafik seperti yang ditunjukkan pada gambar berikut.



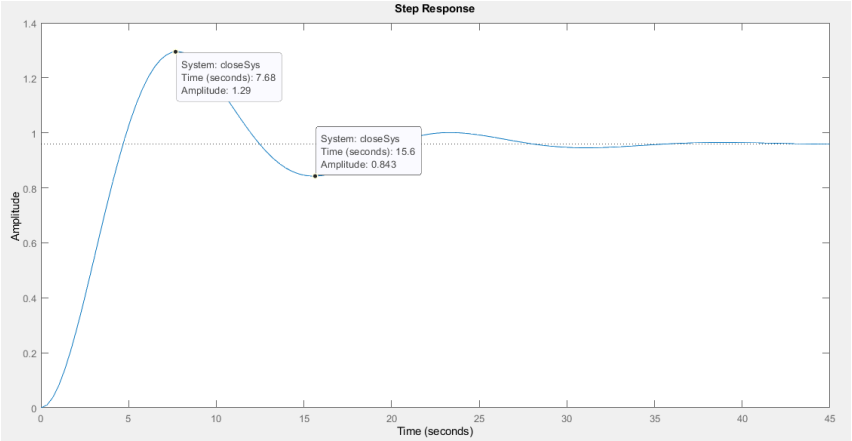
Gambar 4.13 Grafik hasil *Transfer Function System (Metode Ziegler-Nichols)*



Gambar 4.14 Grafik hasil *Transfer Function System (Metode Matlab)*

Dimana pada PID tuner dengan mengatur response time dan transient behavior didapatkan tuning PID seperti pada gambar 4.36. Dari grafik tersebut didapatkan nilai Kp sebesar 0.00612, nilai Ki sebesar 0.00357, dan nilai Kd sebesar 0.262.

Pada metode osilasi, nilai Kp ditetapkan lalu Nvidia Jetbot dijalankan, kemudian dicatat error yang dibuat oleh Nvidia Jetbot. Dengan menggunakan Kp sebesar 0.2, Ki dan Kd sebesar 0. Maka didapat grafik yang ditunjukkan oleh gambar berikut.



Gambar 4.15 Grafik Osilasi *(Error result)*

Tabel 4. 15 Hasil *Tuning PID* dengan Metode Osilasi

Tipe kontroler	Kp	Ti	Td	Ki	Kd
P	4.76	-	0.00	-	-
PI	4.22	8.58	0.00	0.49	-
PID	5.61	8.58	0.20	0.65	1.12

Dari ketiga metode tersebut setelah disimulasikan dengan Nvidia Jetbot, metode PID tuner yang memiliki hasil yang baik dan sesuai dengan keterbatasan dari Nvidia Jetbot, yaitu semua value PID pada Nvidia Jetbot memiliki maksimal value sebesar 1,0 seperti yang sudah dibahas pada bab 3. Namun terdapat beberapa penyesuaian lagi pada tuning di Nvidia Jetbot. Hasil pada metode PID tuner akan dimasukkan pada Nvidia Jetbot.

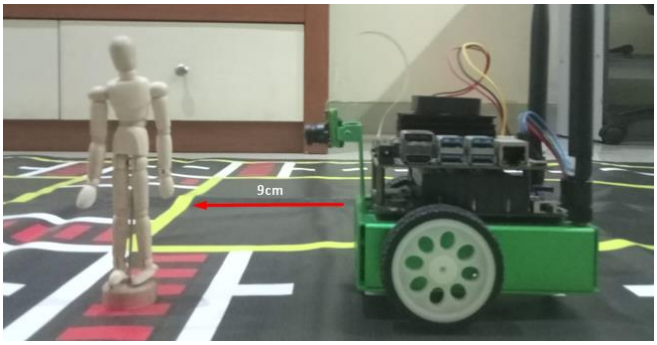
4.3.3 Pembahasan Hasil Pengujian pada Kondisi Terang dengan Dataset Gambar dari Kondisi Terang

Pada Simulasi ini, Jetbot mempunyai waktu reaksi yang sangat baik. Dapat dilihat dari data yang sudah diberikan sebelumnya. Jika dilihat dari grafik perbedaan tingkat keberhasilan antar jumlah data sebanyak 200 dengan 250 gambar terjadi kenaikan dengan rata-rata sebesar 9.33%. Sedangkan kenaikan rata-rata jumlah data 150 gambar dengan 200 gambar terdapat rata-rata kenaikan sebesar 4%. Sedangkan kenaikan rata-rata jumlah data 100 gambar dengan 150 gambar terjadi sebesar 5.33%.

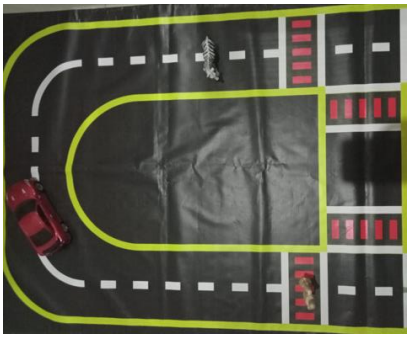
Penurunan tingkat keberhasilan dengan perbandingan kecepatan ini dikarenakan Nvidia JetBot terlalu lambat merespon rintangan dengan kecepatannya yang tidak sebanding dataset yang dipunyainya. Dan sebaliknya kenaikan tingkat keberhasilan dengan perbandingan banyaknya jumlah data ini dikarenakan optimalnya respon JetBot pada pengambilan keputusan mengikuti jalur lintasan. Dan penurunan tingkat keberhasilan dengan penggabungan dua fungsi yang berbeda menyebabkan terjadi kelambatan yang cukup drastis dibandingkan satu fungsi saja, ini dikarenakan banyaknya data dan juga fungsi yang berbeda maka butuh waktu dalam memproses respon untuk fungsi yang teridentifikasi. Maka dari hal ini dapat disimpulkan bahwa untuk mendapatkan keberhasilan yang tinggi, dibutuhkan jumlah data yang banyak dengan kecepatan yang rendah serta kecepatan prosesor dalam memproses dan mengambil keputusan secara cepat.

Untuk karakter pergerakan dari pengujian ini, Jetbot mengalami pergerakan yang cukup baik. Ini dapat dilihat dari pergerakannya di lintasan yang terjadi. Pengujian dengan data 100 gambar mempunyai karakter yang berbeda dengan data 200 gambar ini dikarenakan setiap rintangan yang dipelajari oleh Nvidia JetBot terbatas. Sehingga Ketika Nvidia JetBot berada dalam pada kondisi dimana tidak ada dalam direktori data, maka Nvidia JetBot akan bergerak secara acak untuk Kembali ke posisi yang seperti ada di data. Disaat pengujian dengan jumlah 150 gambar terjadi perubahan yang cukup besar dimana pergerakan JetBot lebih stabil dan lurus, sama halnya pada pengujian dengan jumlah 200 gambar dan 250 gambar. Untuk pengujian dengan kecepatan 0.24, Jetbot sangat sering terjadi keluar dari jalur. Ini dikarenakan waktu respon dari jetbot tidak sebanding dengan kecepatannya, maka terjaldilah Jetbot yang keluar dari jalur lintasan. Tetapi untuk kecepatan 0.2 dan 0.16 cukup stabil dibanding kecepatan 0.24, karakter dari kecepatannya dapat tetap menjaga Jetbot tetap di lintasan.

Untuk karakter pemberhentian pada rintangan sebuah objek pada pengujian ini, Jetbot juga berhenti dengan jarak yang cukup baik. Dimana jarak yang disarankan oleh referensi yang sudah dijelaskan di Bab 2, yaitu standar terbaik yaitu pada jarak 10-15cm. Dan jetbot berhenti di jarak 10-14cm. Untuk setiap data gambar di pengujian ini, melakukan pemberhentiannya dengan sangat baik jika terdeteksi rintangan. Tetapi jika dibandingkan dengan kecepatan, maka kecepatan 0.24 sangat buruk dalam melakukan pemberhentian, sering sekali terjadi hampir tertabrak dengan objek yang menjadi rintangan, terutama jika kecepatan 0.24 memakai data 100 gambar, pemberhentian objek menjadi sangat buruk.



Gambar 4.16 Jarak JetBot berhenti dengan kecepatan 0.24 menggunakan data 150 gambar



Gambar 4.17 Kondisi Eksisting dengan kondisi pencahayaan terang

4.3.4 Pembahasan Hasil Pengujian pada Kondisi Gelap dengan Dataset Gambar dari Kondisi Terang

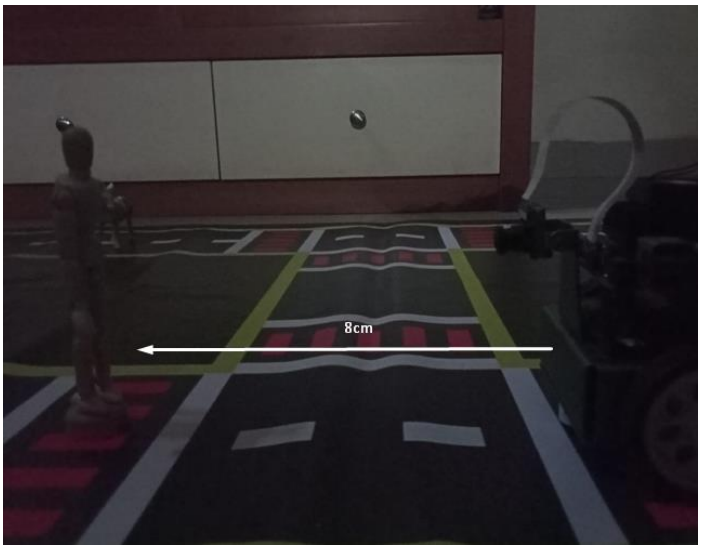
Pada Simulasi ini, Jetbot mempunyai waktu reaksi yang cukup. Dapat dilihat dari data yang sudah diberikan sebelumnya. Jika dilihat dari grafik perbedaan tingkat keberhasilan antar jumlah data sebanyak 200 dengan 250 gambar tidak terlalu jauh, terjadi kenaikan dengan rata-rata sebesar 2.66%. Sedangkan kenaikan rata-rata jumlah data 150 gambar dengan 200 gambar terdapat rata-rata kenaikan sebesar 7%. Sedangkan kenaikan rata-rata jumlah data 100 gambar dengan 150 gambar terjadi sebesar 5.33%.

Penurunan tingkat keberhasilan pada pengujian ini yaitu berada pada factor pencahayaan yang sangat berbeda pada pengujian sebelumnya, dimana pencahayaan yang dipakai pada dataset gambar yaitu dengan pencahayaan yang cukup terang. Dari hal tersebut, keberhasilan yang terjadi untuk melewati setiap rangkaian rintangan juga menurun. Ini dikarenakan penyesuaian antara data gambar yang dipunya dimana gambar dengan pengujian cukup berbeda, dibutuhkan waktu untuk mengidentifikasi dengan kurangnya cahaya pada saat Jetbot melakukan *Live Demo* (Pengujian langsung) tetapi disaat yang bersamaan Jetbot juga sedang bergerak, yang pada akhirnya terjadi tabrakan dengan objek rintangan ataupun keluar dari lintasan yang ada.

Untuk karakter pergerakan dari pengujian ini, Jetbot mengalami pergerakan yang cukup berantakan. Ini terlihat dari bagaimana sulitnya untuk mengidentifikasi mana batas jalur serta juga mengidentifikasi mana jalur yang belok ataupun jalur putaran. Walaupun pergerakan yang terjadi cukup berantakan, Jetbot tetap berhasil untuk tetap dalam lintasan. Tetapi, jika kecepatan semakin tinggi, tingkat keberhasilan Jetbot semakin berkurang dalam mempertahankan dirinya dalam lintasan.

Gambar 4.18 Kondisi Eksisting dengan kondisi pencahayaan gelap

Untuk karakter pemberhentian pada rintangan sebuah objek pada pengujian ini, Jetbot juga berhenti dengan jarak yang cukup buruk. Jetbot berhenti di jarak 5-10cm. Pada kecepatan 0.24 sangat buruk dalam melakukan pemberhentian lebih bahaya daripada pengujian sebelumnya dengan objek yang menjadi rintangan, maksimal pemberhentian terjadi pada jarak 8cm. Melihat kondisi ini, karakter nya dalam merespon identifikasi objek cukup lambat ini dikarenakan pencahayaan yang kurang dengan dataset gambar pada kondisi pencahayaan terang cukup berbeda, dibutuhkan waktu penyesuaian Jetbot pada *Live Demo* dengan Dataset gambar yang dipakai. Karena lambatnya respon tersebut membuat jetbot lambat dalam mengambil keputusan sehingga jarak berhenti sangat dekat.



Gambar 4.19 Jarak JetBot berhenti dengan kecepatan 0.24 menggunakan data 150 gambar

4.3.5 Pembahasan Hasil Pengujian pada Kondisi Gelap dengan Dataset Gambar dari Kondisi Gelap

Pada Simulasi ini, Jetbot mempunyai waktu reaksi yang cukup. Dapat dilihat dari data yang sudah diberikan sebelumnya. Jika dilihat dari grafik perbedaan tingkat keberhasilan antar jumlah data sebanyak 200 dengan 250 gambar tidak terlalu jauh, terjadi kenaikan dengan rata-rata sebesar 2.66%. Sedangkan kenaikan rata-rata jumlah data 150 gambar dengan 200 gambar terdapat rata-rata kenaikan sebesar 7%. Sedangkan kenaikan rata-rata jumlah data 100 gambar dengan 150 gambar terjadi sebesar 5.33%.

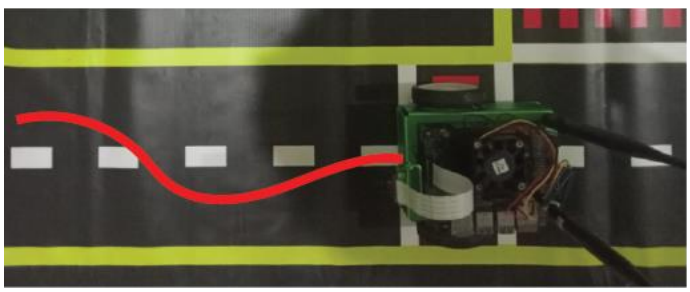
Penurunan tingkat keberhasilan pada pengujian ini dibanding pengujian sebelumnya yaitu berada pada factor pencahayaan juga ditambah dengan data gambar kondisi gelap yang sangat berbeda pada kedua pengujian sebelumnya. Berbeda pada pengujian yang di kondisi gelap sebelumnya, kali ini respon dari pergerakan Jetbot terjadi dengan dikarenakan pembacaan dataset gambar dengan deteksi pada *Live Demo* dipersulit dikarenakan dataset gambar juga kurang cahaya, jadi untuk hal ini keberhasilan berkurang lebih banyak.

Untuk karakter pergerakan dari pengujian ini, Jetbot mengalami pergerakan yang sangat acak. Ini terlihat dari bagaimana sulitnya untuk mengidentifikasi mana batas jalur serta juga mengidentifikasi mana jalur yang belok ataupun jalur putaran. Ini dapat terlihat bagaimana di dataset gambar yang banyak yaitu 250, Jetbot disetiap adanya pergerakan sedikit dia langsung berhenti karena butuh untuk mengidentifikasi lintasannya.

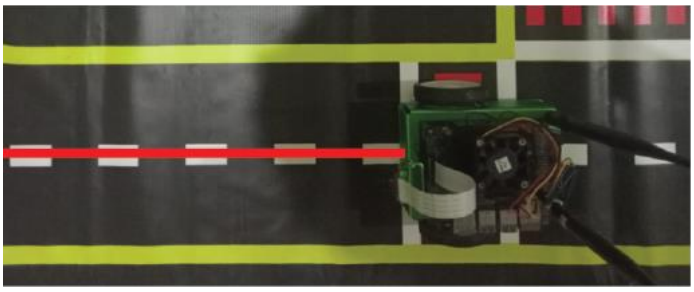
Untuk karakter pemberhentian pada rintangan sebuah objek pada pengujian ini, Jetbot juga berhenti dengan jarak yang cukup buruk. Jetbot berhenti di jarak 3-5cm. Ini dikarenakan respon serta identifikasi dari objek yang ada sangat lambat, maka respon Jetbot berhenti sangat kurang.

4.4 Diskusi Hasil Pengujian

Pada pengujian yang telah dilakukan, kali ini akan dibahas terkait scenario jetbot dalam melewati lintasan. Pada saat pengujian, ada karakteristik Jetbot dalam bergerak di jalan lurus dari jumlah data 100 gambar dengan jumlah data 250 gambar. Disaat pengujian dengan jumlah data 100 gambar JetBot mengalami pergerakan yang tidak stabil dan bergerak maju *zig-zag* ataupun bergelombang. Hal ini dikarenakan jumlah data yang dipelajari oleh Nvidia JetBot terbatas. Sehingga Ketika Nvidia JetBot berada dalam pada kondisi dimana tidak ada dalam direktori data, maka Nvidia JetBot akan bergerak secara acak untuk Kembali ke posisi yang seperti ada di data. Disaat pengujian dengan jumlah 150 gambar terjadi perubahan yang cukup besar dimana pergerakan JetBot lebih stabil dan lurus, sama halnya pada pengujian dengan jumlah 200 gambar dan 250 gambar.

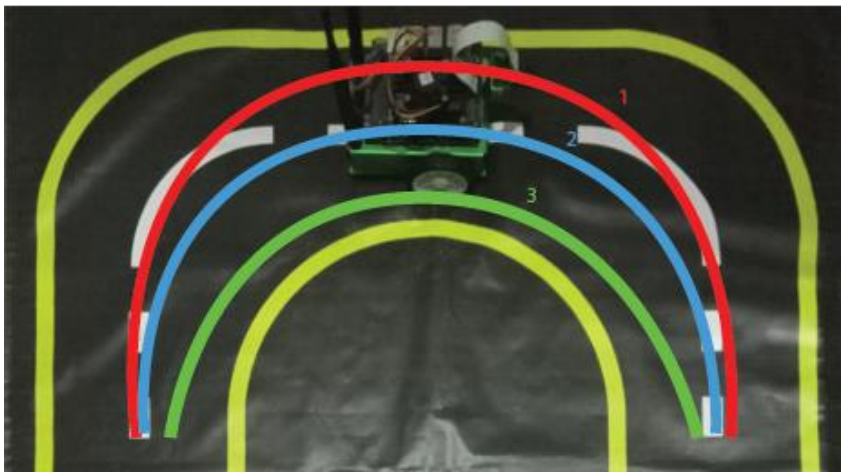


Gambar 4.20 Karakter pergerakan di lintasan lurus JetBot dengan Jumlah Data 50 Gambar

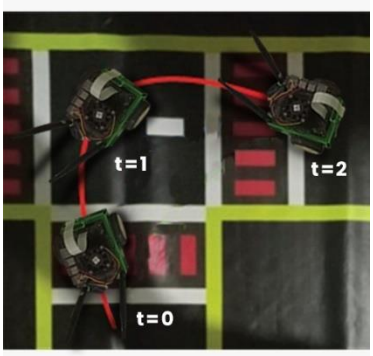


Gambar 4.21 Karakter pergerakan di lintasan lurus JetBot dengan Jumlah Data 200 Gambar

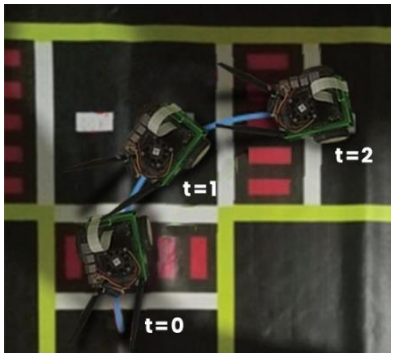
Secara garis besar terdapat tiga skenario dimana Nvidia JetBot melewati lintasan putar balik. yaitu kondisi 1, 2, dan 3. Kondisi 1 yang ditunjukkan oleh garis merah, dimana JetBot terlalu lambat merespon lintasan putar balik sehingga jalur yang diambil yaitu pada sisi terluar. Pada skema ini data yang dipakai yaitu pada kecepatan 0.20 pada jumlah data input 100 gambar dan kecepatan 0.24 pada jumlah data input 100 gambar. Untuk kondisi 2 yang ditunjukkan oleh garis biru merupakan kondisi ideal dimana JetBot dapat mengikut alur garis putus-putus. Pada skema yang terjadi pada ini data yang dipakai yaitu pada kecepatan 0.16 dengan jumlah data input 200, dan 250 gambar, dan kecepatan 0.20 dengan jumlah data input 200, dan 250 gambar. Dan pada kondisi 3 yang ditunjukkan oleh garis hijau merupakan kondisi dimana JetBot terlalu cepat merespon lintasan putar balik sehingga mengambil jalur pada posisi terdalam. Pada skema yang terjadi pada ini data yang dipakai yaitu pada kecepatan 0.16 dengan jumlah data input 100 gambar, dan kecepatan 0.20 dengan jumlah data input 100 gambar.



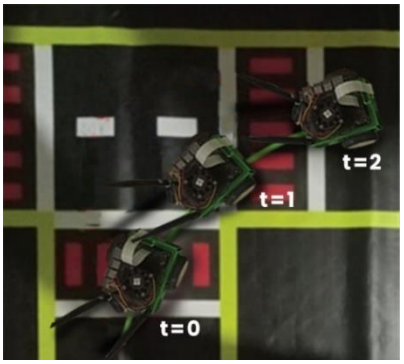
Gambar 4.22 Tiga Skenario JetBot melewati lintasan putar balik



Gambar 4.23 Skenario Pertama JetBot melewati lintasan berbelok

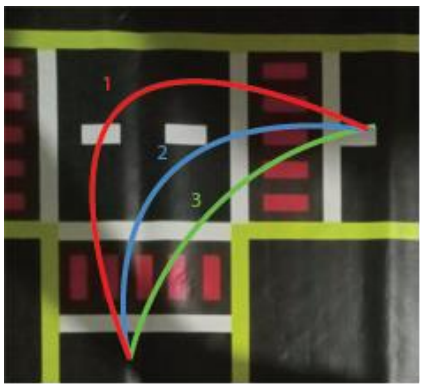


Gambar 4.24 Skenario Kedua JetBot melewati lintasan berbelok



Gambar 4.25 Skenario Ketiga JetBot melewati lintasan berbelok

Terdapat tiga scenario juga dalam melewati lintasan berbelok, yaitu kondisi 1, 2, dan 3. Kondisi 1 yang ditunjukkan oleh garis merah, dimana JetBot terlalu lambat merespon lintasan belok sehingga jalur yang diambil yaitu pada sisi terluar. Pada skema ini data yang dipakai yaitu pada kecepatan 0.20 pada jumlah data input 100 gambar dan kecepatan 0.24 pada jumlah data input 100 gambar. Untuk kondisi 2 yang ditunjukkan oleh garis biru merupakan kondisi ideal dimana JetBot dapat mengikuti alur garis putus-putus dan dan stabil. Pada skema yang terjadi pada ini data yang dipakai yaitu pada kecepatan 0.16 dengan jumlah data input 200 dan 250 gambar, dan kecepatan 0.20 dengan jumlah data input 250 gambar Dan pada kondisi 3 yang ditunjukkan oleh garis hijau merupakan kondisi dimana JetBot terlalu cepat merespon lintasan belok sehingga mengambil jalur pada posisi terdalam. Pada skema yang terjadi pada ini data yang dipakai yaitu pada kecepatan 0.16 dengan jumlah data input 100 gambar, dan kecepatan 0.20 dengan jumlah data input 100 gambar.



Gambar 4.26 Tiga Skenario JetBot melewati lintasan berbelok

Pada percobaan ini, kekurangannya dalam penelitian ini, yaitu tidak adanya perbandingan antara Sistem Pengendali Konvensional dengan Sistem *Artificial Intelligence*.

Dari perbandingan tersebut, penelitian ini dapat membandingkan mana yang lebih unggul terkait pengetesan yang dicoba pada kasus di penelitian ini. Sehingga, dari kasus ini dapat dibandingkan serta dipilih mana sistem yang terbaik untuk dipakai.

Mengacu pada penelitian-penelitian terdahulu yang serupa sudah dilaksanakan oleh Shinji Kawakuran dengan menggunakan Nvidia JetBot untuk mengenali peralatan yang ada di pertanian. *Data Collecting* yang dilakukan pada penilitian tersebut dilakukan dengan jumlah data 100 gambar lima alat pertanian dengan rincian 20 gambar untuk satu alat. Kemudian Nvidia JetBot diperintahkan untuk menghindari alat-alat tersebut yaitu dengan fungsi *Collision Avoidance*. Setelah dilakukan *Data Training* sehingga diperoleh model terbaik, JetBot diujikan dengan model tersebut dengan pengujian peralatan agrikultur pada lintasan yang ada. Setelah dilakukan dua kali pengujian dengan lintasan yang berbeda, didapatkan tingkat keberhasilan sebesar 56,2% dan 69% dengan rata-rata sebesar 58%.

Terdapat pula penilitian yang dilakukan oleh Shao-Kuo Tai dengan fungsi *Road Following* dimana jumlah data yang digunakan yaitu sebanyak 200 gambar dan kemudian pengujian berlanjut dengan dikurangi sebesar 50 gambar secara bertahap dengan kondisi pencahayaan terang dan gelap. Penelitian yang dilakukan oleh Shao-Kuo Tai rata-rata berada diatas 90%. Pada penelitian tersebut dengan lintasan lurus dengan variasi data 50, 100, 150 dan 200 gambar menghasilkan tingkat keberhasilan sebesar 100% pada lintasan lurus dengan pencahayaan terang maupun gelap. Sedangkan pada variasi lintasn putar balik menghasilkan rata-rata sebesar 40%, 82%, 90%, dan 96% pada pencahayaan terang dan 36%, 56%, 72%, dan 80% pada pencahayaan gelap. Dan pada variasi lintasa berbelok menghasilkan tingkat keberhasilan beruturut-turut dengan rata-rata sebesar 46%, 78%, 94%, dan 98%, dan pada pencahayaan gelap berturut-turut sebesar 44%, 64%, 76%, dan 84%.

Dari ketiga penelitian diatas, dengan pengambilan datas dengan jumlah terbanyak dapat memaksimalkan tingkat keberhasilan. Dan dari hal tersebut peneliti berharap terdapat penelitian lebih lanjut terkait *Computer Vision based Autonomous Vehicle*.

“Halaman ini sengaja dikosongkan.”

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilaksanakan, didapatkan beberapa poin yang dapat disimpulkan dari hasil penelitian. Berikut kesimpulan dari penelitian ini.

1. Untuk *Data Training* untuk model terbaik *Collision Avoidance* didapatkan pada komposisi *Train-Test Dataset* 90%-10%. Dan untuk split data terbaik *Collision Avoidance* untuk skema *Training-Validating-Testing* ada pada komposisi *Training* 80%, *Validating* 10%, dan *Testing* 10%. Dan untuk model terbaik *Road Following* didapatkan pada komposisi *Train-Test Dataset* 80%-20%. Dan untuk split data terbaik *Road Following* untuk skema *Training-Validating-Testing* ada pada komposisi *Training* 80%, *Validating* 10%, dan *Testing* 10%.
2. Jumlah data input minimum keberhasil ≥ 60% Nvidia JetBot dalam melakukan *Collision Avoidance* dan *Road Following* yaitu 250 Dataset gambar.
3. Kecepatan maksimum bagi NVIDIA JetBot untuk melakukan fungsi *Collision Avoidance* dan *Road Following* yaitu adalah 0.189 m/s (Level kecepatan 0.2)
4. Performa NVIDIA JetBot dalam melakukan fungsi *Collision Avoidance* dan *Road Following* dengan kondisi pencahayaan terang dengan dataset pada kondisi terang mempunyai tingkat keberhasilan tertinggi diperoleh sebesar 92% (250 Gambar dengan kecepatan 0.116 m/s) dan tingkat keberhasilan terendah diperoleh sebesar 24% (100 Gambar dengan kecepatan 0.253 m/s).
5. Performa NVIDIA JetBot dalam melakukan fungsi *Collision Avoidance* dan *Road Following* dengan baik pada kondisi pencahayaan terang dengan dataset pada kondisi gelap mengalami penurunan yang signifikan dibanding kondisi pencahayaan terang dengan dataset pada kondisi terang. Tingkat keberhasilan tertinggi diperoleh sebesar 72% (250 Gambar dengan kecepatan 0.116 m/s) dan tingkat keberhasilan terendah diperoleh sebesar 12% (100 Gambar dengan kecepatan 0.253 m/s).
6. Performa NVIDIA JetBot dalam melakukan fungsi *Collision Avoidance* dan *Road Following* dengan baik pada kondisi pencahayaan gelap dengan dataset pada kondisi gelap mengalami penurunan yang sangat menurun dibanding kondisi pencahayaan gelap dengan dataset pada kondisi terang. Tingkat keberhasilan tertinggi diperoleh sebesar 64% (250 Gambar dengan kecepatan 0.116 m/s) dan tingkat keberhasilan terendah diperoleh sebesar 8% (100 Gambar dengan kecepatan 0.253 m/s).

5.2 Saran

Dengan selesainya penelitian, peneliti ingin memberikan beberapa saran guna pengembangan lanjutan dari laporan ini. Berikut saran yang dihasilkan dari penelitian ini.

1. Menerapkan pengereman mekanis sehingga JetBot dapat berhenti dengan responsif, dan menambahkan *gearbox* tambahan untuk menyesuaikan putaran motoar sehingga kendaraan dapat lebih stabil.
2. Memperbesar kapasitas memori sehingga dapat menyimpan lebih banyak koleksi data pada *Data Collection*
3. Dapat mengintegrasikan dan mempercepat klasifikasi antara fungsi *Collision Avoidance* dan *Road Following* dengan menggunakan *ROS* pada JetBot
4. Menambahkan rambu lalu lintas yang sesuai dengan rambu di Indonesia
5. Menggunakan Lux Meter untuk mengukur intensitas cahaya.
6. Melakukan ujicoba dengan Sistem Pengendali Konvensional (Penggunaan alat *Ultrasonic, Lidar*, tanpa adanya Sistem *Artificial Intelligence*)

“Halaman ini sengaja dikosongkan.”

DAFTAR PUSTAKA

Mouton, C., Myburgh, J. C., & Davel, M. H. (2021). Stride and translation invariance in CNNs. In Southern African Conference for Artificial Intelligence Research (pp. 267-281). Springer International Publishing. https://doi.org/10.1007/978-3-030-66151-9_17

Géron, Aurélien (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media. https://katalog.ub.uni-heidelberg.de/titel/68502223

Zhang, W., Itoh, K., Tanida, J., & Ichioka, Y. (1990). Parallel distributed processing model with local space-invariant interconnections and its optical architecture. Proceedings of Annual Conference of the Japan Society of Applied Physics. https://doi.org/10.1364/AO.29.004790

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. Springer International Publishing. https://doi.org/10.1007/978-3-642-15825-4_10

Ciregan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. IEEE. https://doi.org/10.1109/cvpr.2012.6248110

Corovic, A., Ilic, V., Marijan, M. (2018). The Real-Time Detection of Traffic Participants Using YOLO Algorithm. Telfor 2018. https://doi.org/10.1109/TELFOR.2018.8611986

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. IEEE. https://doi.org/10.1109/iccv.2015.123

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once:Unified, Real-Time Object Detection. University of Washington, Allen Institute for AI, Facebook AI Research. https://doi.org/10.48550/arXiv.1506.02640

Feng, H., Ma, W., Yin, C., & Cao, D. (2021). Trajectory control of electro-hydraulic position servo system using improved PSO-PID controller. International Research Journal. https://doi.org/10.1016/j.autcon.2021.103722

Lee, M. (2020). An analysis of the effects of artificial intelligence on electric vehicle technology innovation using patent data. Elsevier Ltd. https://doi.org/10.1016/j.wpi.2020.102002

Rosebrock, A. (2017). Deep learning for computer vision with python: ImageNet Bundle. PyImageSearch.

Ketkar, N., & Santana, E. (2017). Deep learning with Python (Vol. 1). Apress Media LLC.

Kawakura, S., & Shibasaki, R. (2020). Deep Learning-Based Self-Driving Car: JetBot with NVIDIA AI Board to Deliver Items at Agricultural Workplace with Object-Finding and Avoidance Functions. https://doi.org/10.24018/ejfood.2020.2.3.45.

Ahmad, I., & Pothuganti, K. (2020). Design & implementation of real time autonomous car by using image processing & IoT. IEEE. https://doi.org/10.1109/ICSSIT48917.2020.9214125.

Fathy, M., Ashraf, N., Ismail, O., Fouad, S., Shaheen, L., & Hamdy, A. (2020). Design and implementation of self-driving car. Elsevier Ltd. https://doi.org/10.1016/j.procs.2020.07.026

Clarke, D.J.B., et al. (2021). Appyters: Turning Jupyter Notebooks into data-driven web apps. New York: Journal Pattern DOI. https://doi.org/10.1016/j.patter.2021.100213

BPS. (2018). Perkembangan Jumlah Kendaraan Bermotor Menurut Jenis 1949-2018, <URL: https://www.bps.go.id/linkTableDinamis/view/id/1133>.

Aguiar, A.S.P., et al. 2020. Vineyard trunk detection using deep learning – An experimental device benchmark. Elsevier Ltd. https://doi.org/10.1016/j.compag.2020.105535.

Kannapiran, S., et al. 2020. “Go-CHART: A miniature remotely accessible self-driving car robot”. IEEE/RSJ.

Marcelino, P. 2018. Transfer learning from pre-trained models. <URL: https://towardsdatascience.com/transfer-learningfrom-pre-trained-models-f2393f124751>

LAMPIRAN

I. *Training data,* dan *test dataset* data *collision avoidance* pada Kondisi Terang

https://docs.google.com/spreadsheets/d/1ovO8X5tvIDro-952D7_VxreP3qZOIJ66/edit#gid=1335942614

II. *Training data train, validation, test dataset data collision avoidance* pada Kondisi Terang

https://docs.google.com/spreadsheets/d/1ovO8X5tvIDro-952D7_VxreP3qZOIJ66/edit#gid=1871356613

III. *Training data,* dan *test dataset* data *road following* pada Kondisi Terang

https://docs.google.com/spreadsheets/d/1ovO8X5tvIDro-952D7_VxreP3qZOIJ66/edit#gid=2111547839

IV. *Training data train, validation, test dataset* data *road following* pada Kondisi Terang

https://docs.google.com/spreadsheets/d/1ovO8X5tvIDro-952D7_VxreP3qZOIJ66/edit#gid=2031977548

V. *Training data,* dan *test dataset* data *collision avoidance* pada Kondisi Gelap

https://docs.google.com/spreadsheets/d/1br-mCR5_BApRdokgZadZFy1cim-0TMD7/edit#gid=265246063

VI. *Training data train, validation, test dataset data collision avoidance* pada Kondisi Gelap

https://docs.google.com/spreadsheets/d/1br-mCR5_BApRdokgZadZFy1cim-0TMD7/edit#gid=403152281

VII. *Training data,* dan *test dataset* data *road following* pada Kondisi Gelap

https://docs.google.com/spreadsheets/d/1br-mCR5_BApRdokgZadZFy1cim-0TMD7/edit#gid=898638146

VIII. *Training data train, validation, test dataset* data *road following* pada Kondisi Gelap

https://docs.google.com/spreadsheets/d/1br-mCR5_BApRdokgZadZFy1cim-0TMD7/edit#gid=129574623

IX. *Query* dan *Syntax Collision Avoidance Program* pada Jetbot

naframo/JetBot at Collision-Avoidance-1 (github.com)

X. *Query* dan *Syntax Road Following Program* pada Jetbot

naframo/JetBot at Road-Following (github.com)

XI. *Query* dan *Syntax execution Road Following* dan *Collision Avoidance Program* pada Jetbot

JetBot/RoadFollowing+CollisionAvoidance.ipynb at main · naframo/JetBot (github.com)

BIODATA PENULIS

Penulis merupakan anak pertama dari 3 bersaudara kelahiran Bekasi, 30 November 1999. Penulis menempuh pendidikan Tingkat Dasar di SDN Jakasetia III, kemudian melanjutkan Sekolah Menengah Pertama di SMPN 117 Jakarta, kemudian melanjutkan Sekolah Menengah Atas di SMAN 71 Jakarta. Pada tahun 2017 penulis diterima di program S-1 Departemen Teknik Mesin, Fakultas Teknologi Industri dan Rekayasa Sistem, Institut Sepuluh Nopember dan terdaftar sebagai mahasiswa dengan NRP 02111740000056. Penulis aktif dalam kepanitiaan kegiatan di lingkungan kampus ITS, seperti menjadi Perwakilan Indonesia dalam Belt and Road Student Exchange 2018 di China, Ketua GERIGI ITS, Menteri Ekonomi Kreatif BEM ITS, dan Aktivitas lain. Sehubungan dengan hasil penelitian yang dilakukan, untuk menghubungi penulis dalam rangka pemenuhan kritik dan saran dari pembaca, dapat dilakukan melalui email: 11monovation@gmail.com.