PENERAPAN *AUTONOMOUS VEHICLE BEHAVIOR* PADA PERMAINAN SIMULASI UJIAN BERKENDARA 3D MENGGUNAKAN METODE *STEERING BEHAVIOR*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Rosikhan Maulana Yusuf
NIM: 115060800111094

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PENERAPAN *AUTONOMOUS VEHICLE BEHAVIOR* PADA PERMAINAN SIMULASI UJIAN BERKENDARA 3D MENGGUNAKAN METODE *STEERING BEHAVIOR*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Rosikhan Maulana Yusuf
NIM: 115060800111094

Skripsi ini telah diuji dan dinyatakan lulus pada
18 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eriq Muhammad Adams J., S.T, M.Kom	Issa Arwani, S.Kom, M.Sc
NIP: 19850410 201212 1 001	NIP. 19830922 201212 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

IDENTITAS PENGUJI

Penguji 1:

Muhammad Aminul Akbar, S.Kom, M.T

NIK. 2016078910131001

Penguji 2:

Komang Candra Brata, S.Kom, M.T, M.Sc

NIK. 2016078907111001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 18 Januari 2018

Rosikhan Maulana Yusuf

NIM: 115060800111094

DAFTAR RIWAYAT HIDUP

Nama	Rosikhan Maulana Yusuf	
Tempat, Tanggal Lahir	Malang, 2 Januari 1992	
Alamat	Jl. Delima 02 Dermo Mulyoagung, Kecamatan Dau, Kabupaten Malang	
Kewarganegaraan	Indonesia	
Jenis Kelamin	Laki-laki	
Status Pernikahan	Belum Menikah	
Nomor Handphone	085649705026	
E-Mail	rosikhanmy@outlook.com	
Riwayat Pendidikan	1999 - 2005	SDN Landungsari 01
	2005 - 2008	SMP Brawijaya Smart School
	2008 - 2011	SMAN 8 Malang
	2011 - 2018	Universitar Brawijaya Malang

UCAPAN TERIMAKASIH

1. Puji dan syukur Penulis panjatkan kehadirat Allah SWT
2. Kedua orang tua Penulis Almarhum Ir. Nur Komar, M.S dan Maimunah
3. Kelima saudara kandung Fajrin Hal Lala, Dayyinul Dalili, Dzunnun Robbi Alhaq, Zur'ata Asykurian Nahr, Fafiru Aqdam
4. Fikhi Nugroho, S.Kom, dan Keluarga Besar Mahasiswa Teknik Informatika/Ilmu Komputer, khususnya angkatan 2011.

ABSTRAK

Rosikhan Maulana Yusuf, Penerapan *Autonomous Vehicle Behavior* Pada Permainan Simulasi Ujian Berkendara 3D Menggunakan Metode *Steering Behavior*

Pembimbing: Eriq M. Adams J., S.T, M.Kom dan Issa Arwani, S.Kom, M.Sc

Simulasi berkendara merupakan salah satu sarana untuk meningkatkan kemampuan seseorang untuk berkendara. Rancang bangun simulasi berkendara 3D yang telah dibuat dapat memberikan kontrol dan visual yang mendukung untuk pembelajaran berkendara. Tetapi dalam dunia nyata pengendara tidak hanya berhadapan dengan rambu-rambu lalulintas. Pemain juga memerlukan pembelajaran berkendara layaknya dunia nyata dengan tambahan mobil lain yang bergerak seperti di dunia nyata. Sehingga diperlukan agen otomatis yang bertindak sebagai mobil. Solusi untuk agen otomatis ini adalah dengan menggunakan *Autonomous Vehicle* yang menggunakan metode *Steering Behavior* sebagai dasar. Pada tahap perancangan, kebutuhan dalam pembuatan *Steering Behavior* disusun dalam perancangan *Car Controller*, perancangan *Steering Behavior* dan perancangan *Waypoint*. Kemudian program diimplementasikan menggunakan Unity3D dengan bahasa pemrograman C#. Pada saat pengujian White Box, dapat disimpulkan bahwa unit modul program sudah memenuhi kebutuhan fungsional. Pada pengujian akurasi dan presisi, dapat disimpulkan bahwa akurasi dan presisi offset adalah 97.9% dan 0.166427546 meter, akurasi dan preisisi magnitude error adalah 68.3% dan 1.605146908 m/s. Pada pengujian performa, dapat disimpulkan bahwa performa terbaik yang dapat diberikan Autonomous Vehicle adalah dengan menjalankan agen secara bersamaan dengan jumlah kurang dari 180 agen.

Kata kunci: autonomous vehicle, steering behavior, simulasi, waypoints, car controller

ABSTRACT

Rosikhan Maulana Yusuf, Applying Autonomous Vehicle Behavior In 3D Driving Exam Simulation Game Using Steering Behavior Method

Preceptors: Eriq M. Adams J., S.T, M.Kom and Issa Arwani, S.Kom, M.Sc

Driving simulation is one means to improve one's ability to drive. The design of 3D driving simulations that have been created can provide control and visual support for driving learning. However, in the real-world riders are not just dealing with traffic signs. Players also require real-world driving learning in addition to other cars that move like in the real world. So it takes an automatic agent that acts as a car. The solution for this automated agent is to use Autonomous Vehicle which uses the Steering Behavior method as the base. At design stage, the requirement in making Steering Behavior is arranged in Car Controller design, Steering Behavior design and Waypoint design. Then the program is implemented using Unity3D with C# programming language. At the time of testing White Box, it can be concluded that the unit module program already meets the functional needs. In the test of accuracy and precision, it can be concluded that offset accuracy is 97.9% and offset precision is 0.166427546 meter. The magnitude error accuracy is 68.3% and magnitude error precision is 1.605146908 m/s. In performance testing, it can be concluded that the best performance of Autonomous Vehicle can handle is when executing agents simultinously with no more than 180 agents.

Keywords: autonomous vehicle, steering behavior, simulation, waypoints, car controller

KATA PENGANTAR

Puji dan syukur Penulis panjatkan ke hadirat Allah SWT, karena hanya dengan rahmat dan karunia-Nya Penulis dapat menyelesaikan skripsi dengan judul "Penerapan *Autonomous Vehicle Behavior* Pada Permainan Simulasi Ujian Berkendara 3D Menggunakan Metode *Steering Behavior*" dengan baik. Melalui kesempatan ini, Penulis ingin menyampaikan rasa hormat dan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan bantuan dan dukungan selama pengerjaan skripsi, di antaranya:

1. Syukur Alhamdulillah kepada Allah SWT atas syukur dan nikmat kesehatan yang telah diberikan.
2. Kedua orang tua Almarhum Nur Komar dan Maimunah yang telah memberi motivasi, kasih sayang serta dukungan moril dan materiil.
3. Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D, Ir. Heru Nurwarsito, M.Kom, Drs. Marji M.T, dan Edy Santoso, S.Si, M.Kom, selaku Dekan, Wakil Dekan I, Wakil Ketua II, dan Wakil Ketua III Fakultas Ilmu Komputer Universitas Brawijaya.
4. Tri Astoto Kurniawan, S.T, M.T, Ph.D, Muhammad Tanzil Furqon, S.Kom, M.CompSc dan Agus Wahyu Widodo, S.T, M.Cs, selaku Ketua Jurusan, Sekretaris Jurusan dan Ketua Program Studi Teknik Informatika Universitas Brawijaya.
5. Bapak Eriq Muhammad Adams Jonemaro S.T, M.Kom selaku dosen pembimbing I yang telah banyak memberikan ilmu, bimbingan, arahan, motivasi, serta meluangkan waktunya selama penyusunan skripsi ini.
6. Bapak Issa Arwani, S.Kom, M.Sc selaku dosen pembimbing II yang telah banyak memberikan ilmu, bimbingan, arahan, nasihat, serta meluangkan waktunya selama penyusunan skripsi ini.
7. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
8. Staf administrasi Program Studi Teknik Informatika, Fakultas Ilmu Komputer yang telah membantu menyelesaikan urusan administrasi.
9. Keluarga Besar Mahasiswa Teknik Informatika khususnya angkatan 2011 yang sama-sama berjuang menyelesaikan tugas akhir, serta Fikhi Nugroho, S.Kom. yang memberikan bantuan lebih dari sekedar dukungan.
10. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang terlibat baik secara langsung maupun tidak langsung demi terselesaikannya skripsi ini.

Semoga jasa dan amal baik mendapatkan balasan dari Allah SWT. Dengan segala kerendahan hati, penulis menyadari sepenuhnya bahwa skripsi ini masih jauh dari sempurna karena keterbatasan materi dan pengetahuan yang dimiliki penulis. Akhirnya, semoga skripsi ini dapat bermanfaat dan berguna bagi pembaca terutama mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya

Malang, 18 Januari 2018

Rosikhan Maulana Yusuf
rosikhanmy@outlook.com

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE PROGRAM

BAB 1 PENDAHULUAN

1.1 Latar belakang

Simulasi berkendara merupakan salah satu sarana untuk meningkatkan kemampuan belajar seseorang dalam hal berkendara baik untuk kendaraan roda dua maupun roda empat. Dalam simulasi berkendara ini akan diajarkan tentang instruksi rambu-rambu lalu lintas dan fungsi dari kendali kendaraan tersebut. Selain itu dalam simulasi tersebut mempunyai fungsi lain yang lebih penting yaitu memberikan pengalaman dalam mengemudi yang menyerupai dunia nyata. Sehingga nantinya diharapkan pemain dapat beradaptasi dengan kondisi jalan raya di dunia nyata dengan lebih cepat.

Aplikasi simulasi berkendara yang akan digunakan untuk berlatih harus memiliki standar operasional yang telah di tetapkan oleh pemerintah. Aplikasi ini juga harus memiliki unsur menyenangkan agar tidak membosankan. Oleh karena itu aplikasi simulasi berkendara di rancang dalam bentuk Permainan Simulasi Ujian Berkendara 3D. Perancangan simulasi tersebut telah dilakukan pada penelitian sebelumnya yang di lakukan oleh Kukuh Heru Irawan dan Mury Fajar Dewantara. Berdasarkan hasil penelitian tersebut, dapat disimpulkan bahwa permainan simulasi berkendara telah berjalan dengan baik dan dapat memberikan hiburan yang menyenangkan bagi pemain. Salah satu saran dari penelitian tersebut adalah penambahan fitur player tambahan pada permainan (Irawan, 2013)(Dewantoro, 2015). Dalam hal ini penulis akan menambahkan fitur *Autonomous Agent* di dalam simulasi tersebut untuk meningkatkan pengalaman bermain.

Autonomous Agent merupakan suatu entitas yang membuat pilihan sendiri tentang bagaimana dia bertindak pada suatu lingkungan tanpa campur tangan dari pengguna maupun perencanaan global (Franklin & Graesser, 1996)(Shiffman, 2012). Informasi yang di dapatkan oleh agen akan digunakan untuk merencanakan aksi pada lingkungan di mana agen itu berada. Dalam lingkup kendaraan, agen di sebut juga dengan Autonomous Vehicle di mana aksi yang dilakukan menirukan pergerakan mobil. Pergerakan mobil terdiri dari sekumpulan behavior independen yang setiap behavior bertanggung jawab mengerjakan beberapa aspek pergerakan mobil. Setiap aspek tersebut memiliki satu nilai parameter kontrol yaitu steering angle dan acceleration. Sekumpulan behavior tersebut terdapat dalam metode Steering Behavior (Wang, 2005).

Steering Behavior merupakan kalkulasi geometris pada sebuah representasi kendali vektor (Reynolds, 1999). Metode ini pada dasarnya memiliki tiga langkah yang salah satunya adalah *Locomotion*. *Locomotion* ini adalah penggerak suatu *agent*. Karena perilaku *locomotion* ini menyerupai kendaraan, maka metode ini cocok untuk diterapkan pada *Autonomous Vehicle.* Oleh karena itu penulis melakukan penelitian dengan judul Penerapan *Autonomous Vehicle Behavior* Pada Permainan Simulasi Ujian Berkendara 3D Menggunakan Metode *Steering Behavior*.

1.2 Rumusan masalah

Berdasarkan latar belakang di atas dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana merancang dan mengimplementasi *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.
2. Bagaimana akurasi dan presisi *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.
3. Bagaimana performa *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.

1.3 Tujuan

Secara umum tujuan penelitian ini berdasarkan rumusan masalah di atas adalah untuk menerapkan metode *Steering Behavior* pada *Autonomous Vehicle*. Tujuan khusus penelitian ini antaran lain:

1. Merancang dan mengimplementasi metode *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.
2. Mengukur akurasi dan presisi *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.
3. Mengukur performa *Steering Behavior* untuk *Autonomous Vehicle* pada Permainan Simulasi Ujian Berkendara 3D.

1.4 Manfaat

Manfaat yang ingin dicapai dari implementasi ini adalah:

1. Bagi Penulis
   a. Menerapkan Ilmu yang didapat dari informatika konsentrasi Game Universitas Brawijaya.
   b. Memperdalam ilmu tentang simulasi dan pengembangan game.
2. Bagi Pengguna

   Menambahkan fitur Autonomous Vehicle untuk meningkatkan pengalaman bermain pada Permainan Simulasi Ujian Berkendara 3D.

1.5 Batasan masalah

Untuk menghindari pembahasan yang tidak berfokus maka penulis membuat batasan ruang lingkup sebagai berikut:

1. Permainan Simulasi Ujian Berkendara 3D diterapkan pada platform PC.
2. Penerapan *Autonomous Vehicle* menggunakan Unity 3D Engine dan Mono Develop dengan bahasa pemrograman C#.
3. *Autonomous Vehicle* bergerak berdasarkan *waypoint* yang telah ditentukan.
4. Penerapan *Autonomous Vehicle* hanya untuk gerakan maju.

5. *Waypoint* yang akan di implementasi merupakan jalur sirkuit tertutup (*closed road circuit*).

1.6 Sistematika pembahasan

Dalam penulisan skripsi ini terdiri dari 6 bab dan beberapa sub bagian. Untuk memudahkan dalam penulisan dan pembahasan pada tiap-tiap bab agar sesuai dengan rumusan masalah yang ada maka digunakan sistematika penulisan seperti dibawah ini.

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika penulisan.

BAB II Landasan Kepustakaan

Menguraikan teori dasar dan teori penunjang serta membahas teori-teori yang mendukung dalam implementasi *Autonomous Vehicle Behavior* menggunakan *Steering Behavior*.

BAB III Metodologi

Membahas metode yang digunakan dalam penelitian yang terdiri dari Identifikasi masalah, studi literatur, perancangan, implementasi, pengujian, analisis dan pengambilan kesimpulan.

BAB IV Perancangan dan Implementasi

Membahas langkah-langkah perancangan dan proses implementasi *Autonomous Vehicle*.

BAB V Pengujian dan Analisis

Memuat tentang hasil pengujian terhadap simulasi yang telah direalisasikan dan menganalisis hasil pengujian.

Bab VI Kesimpulan dan Saran

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian perangkat lunak yang dikembangkan dalam skripsi ini serta saran – saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Implementasi *Autonomous Vehicle Behavior*. Beberapa dasar teori yang dimaksud antara lain adalah teori yang mendasari *Autonomous Agent*, penjelasan tentang *Steering Behavior* beserta metode yang digunakan untuk menerapkan teori.

2.1 Autonomous Vehicle

Autonomous Vehicle merupakan salah satu bagian *Autonomous Agent*. Pengertian agen memiliki berbagai definisi. Tiap peneliti ingin menjelaskan pengertian agen sesuai dengan kebutuhan penelitiannya. Definisi dari agen ini berkisar dari yang sederhana sampai penjelasan panjang yang beruntun. Masing-masing definisi muncul dari himpunan contoh agen. Beberapa di antaranya adalah sebagai berikut:

- Maes Agent : "*Autonomous Agents* adalah sistem komputasi yang menghuni beberapa lingkungan dinamis yang kompleks, merasakan dan bertindak secara otomatis pada lingkungan tersebut, dan dengan demikian mewujudkan serangkaian tujuan atau tugas yang mereka rancangkan."
- KidSim Agent : "Mari kita definisikan agen sebagai entitas *software* yang di dedikasikan pada tujuan tertentu. *"Presistent"* membedakan agen dari subrutin, agen memiliki agenda mereka sendiri. *"Special purpose"* membedakan mereka dari seluruh aplikasi multifungsi, agen biasanya jauh lebih kecil."
- Hayes-Roth Agent : "*Intelligent Agents* terus melakukan tiga fungsi, yaitu persepsi kondisi dinamis dalam lingkungan, tindakan untuk mempengaruhi kondisi lingkungan, dan penalaran untuk menafsirkan persepsi, memecahkan masalah, menarik kesimpulan, dan menentukan tindakan."

Autonomous Agent merupakan sistem yang terletak di dalam dan bagian dari lingkungan yang merasakan lingkungan itu dan bertindak di atasnya, dari waktu ke waktu, dalam kegiatan sendiri sehingga dapat mempengaruhi apa yang dia rasakan berikutnya (Franklin & Graesser, 1996) Berdasarkan definisi di atas dapat diambil tiga inti dalam *Autonomous Agent*, yaitu pengumpulan informasi dari lingkungan agen, penentuan aksi agen berdasarkan informasi dan penerapan aksi pada agen.

Autonomous Vehicle memiliki perlakuan berbeda dari *Autonomous Agent* meskipun sama-sama merupakan bagian dari agen. Karena pergerakannya yang seperti mobil, *Autonomous Vehicle* memerlukan perencanaan lebih matang untuk menggerakkannya karena gerakannya yang terbatas.

2.2 *Steering Behavior*

Pernyataan *Behavior* memiliki banyak arti. Bisa jadi memiliki arti tindakan kompleks dari manusia atau hewan berdasarkan kemauan dan naluri. Bisa juga

berarti tindakan terprediksi dari sistem mekanik sederhana atau tindakan kompleks pada sebuah sistem rumit. Pada *virtual reality* dan *multimedia*, biasanya *Behavior* di gunakan sebagai sinonim dari "animasi". Dalam studi literatur ini, *behavior* merujuk pada tindakan terimprovisasi dan seperti hidup pada *autonomous character*.

Behavior untuk *autonomous character* dapat dipahami dengan baik dengan membaginya menjadi tiga layer seperti pada Gambar 2.1.

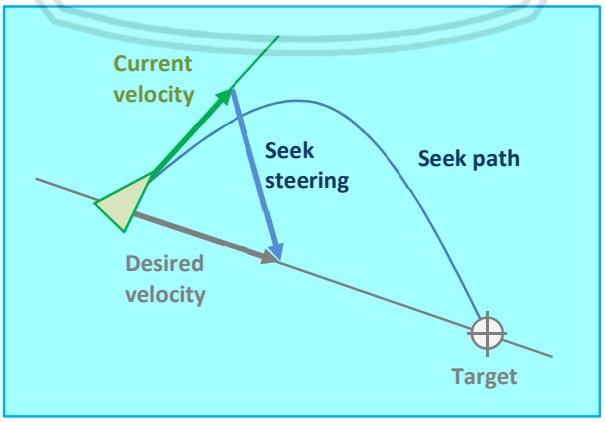
Gambar 2.1 : Pembagian *Layer Autonomous Character*

Sumber: (Reynolds, 1999)

Locomotion adalah bagian terbawah dari tiga layer seperti pada gambar di atas. Layer *Locomotion* merepresentasikan sebuah karakter, mengubah *control* dari layer *steering* menjadi gerakan. Gerakan ini memiliki batasan sesuai kondisi fisik. Seperti interaksi momentum dan tekanan. Untuk pembahasan ini, *Locomotion* di modelkan dalam bentuk kendaraan. Skema *Locomotion* pada kendaraan ini disebut dengan *Steering Behavior* (Reynolds, 1999) (Shiffman, 2012).

Penerapan *Steering Behavior* diawali dengan gerakan sederhana. *Seek* merupakan gerakan mendasar dari semua gerakan yang akan dijelaskan dibawah, antara lain *Persuit* dan *Path Following*.

2.2.1 *Seek*



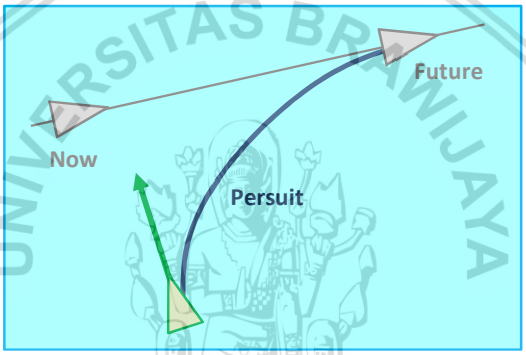
Gambar 2.2 : *Seek* dan *Flee*

Sumber : (Reynolds, 1999)

Seek merupakan proses bergerak menuju target. Jika agen di analogikan dengan vektor, maka langkah pergerakannya diawali dengan menentukan posisi target dan mendapatkan vektor jarak ke target dari posisi agen (*desired velocity*). Kemudian vektor tersebut dijumlahkan dengan vektor kecepatan *agent* (*current velocity*) sehingga mendapatkan vektor baru. Vektor baru ini merupakan *velocity* baru yang akan digunakan untuk bergerak. Jika di animasikan maka akan menghasilkan kurva seperti pada Gambar 2.2.

2.2.2 *Persuit*

Pergerakan *Persuit* mirip dengan pergerakan *Seek*. Perbedaannya adalah target yang dituju bergerak. Sehingga target yang di kejar bukan posisi target, melainkan posisi gerakan ke depan target. Jika di analogikan dengan vektor, kecepatan agen (*current velocity*) dijumlahkan dengan posisi depan target (*future target*) menghasilkan *persuit steering*. *Persuit* berguna untuk penerapan *Path Following* di mana target yang dituju bergerak seiring dengan gerakan agen.

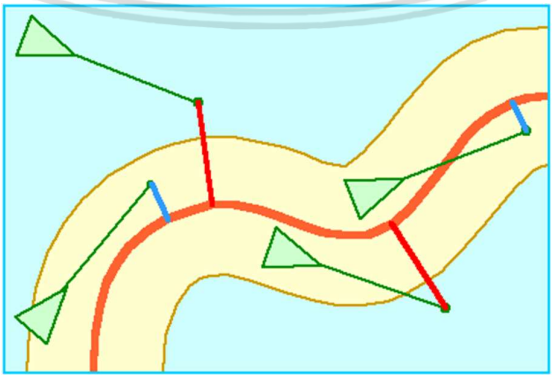


Gambar 2.3 : *Persuit*

Sumber : (Reynolds, 1999)

Pada Gambar 2.3 dapat dilihat agen yang berwarna abu-abu bergerak dari posisi now ke posisi future. Agen dengan posisi dan vektor awal tegak lurus dengan agen abu-abu berangsur-angsur berbelok mengikuti arah agen yang di tuju.

2.2.3 *Path Following*



Gambar 2.4 : *Path Following*

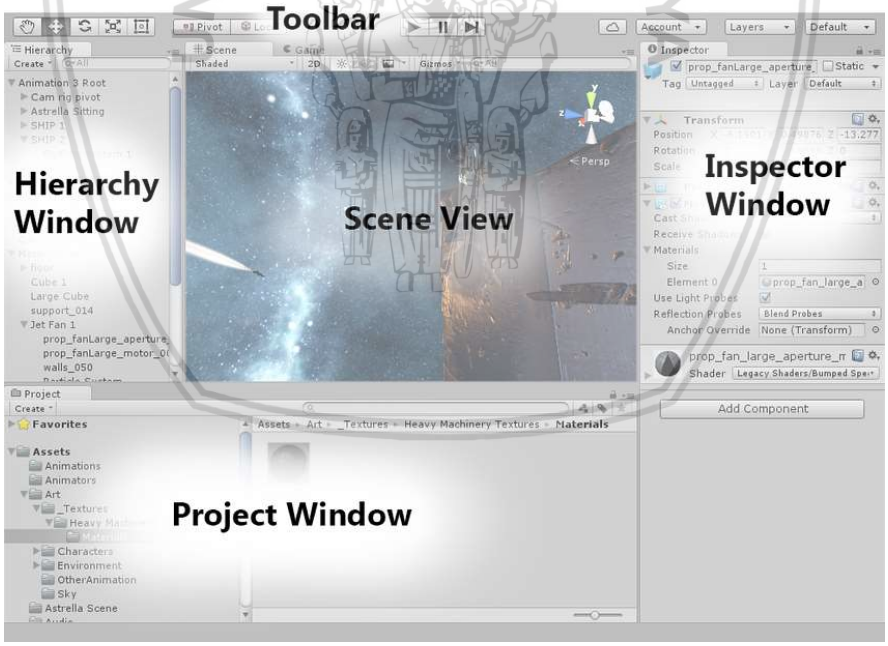
Sumber : (Reynolds, 1999)

Path Following merupakan pengembangan dari *Persuit*. Agen bergerak menuju target yang bergerak menelusuri jalur (*path*) yang telah di tentukan. Pada definisi asli, jalur tersebut memiliki radius dari pusat jalur sehingga lebih mirip selang.

Berdasarkan Gambar 2.4, wilayah hijau dianggap sebagai *Obstacle*. Ketika agen keluar dari area tersebut, maka agen akan dikenakan dorongan menuju ke pusat jalur hingga agen akan berbelok memasuki wilayah radius. Sedangkan ketika agen di dalam area radius, maka agen akan tetap berjalan lurus hingga agen keluar lagi dari area radius. Proses ini akan memberikan kesan agen bergerak mengikuti jalur.

2.3 Unity 3D

Unity 3D adalah *Multiplatform Game Engine* yang di desain khusus untuk membuat *game* dalam bentuk 2D maupun 3D. *Game* yang dibuat dengan Unity 3D dapat di publish pada berbagai macam platform, diantaranya adalah platform PC, Mac, Linux, Android, iPhone, Playstation, XBOX, VR dan berbagai macam platform lainnya. Banyak fitur yang di sediakan oleh Unity 3D untuk membantu pengembangan *game*. Fitur utama yang dimiliki Unity 3D adalah *Editor* dimana fitur ini adalah fitur utama untuk membantu merancang game design sehingga proses pengembangan lebih cepat dan tertata. Terdapat beberapa bagian utama pada *Editor* Unity 3D dengan lokasi default sebagai berikut.



Gambar 2.5 : *User Interface* Pada *Editor* Unity 3D

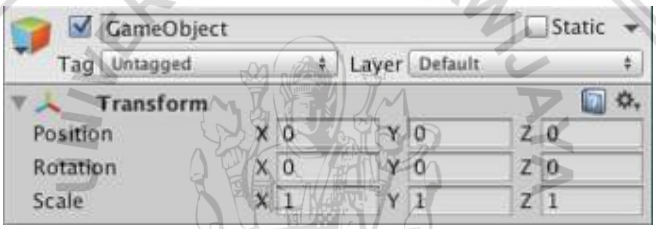
Sumber : (Unity Technologies, 2017)

User Interface utama pada Unity Editor seperti pada Gambar 2.5 terdiri atas:

1. *Scene View* yang berfungsi untuk mengatur dan navigasi dalam suatu scene. *Scene View* dapat menampilkan prespektif 2D maupun 3D sesuai dengan tipe project yang sedang dikembangkan.

2. *Hierarchy Window* adalah representasi teks hirarkis dari setiap objek dalam *Scene*. Setiap item pada *Scene* memiliki entri dalam *Hierarchy*, sehingga kedua *window* tersebut terhubung secara inheren. *Hierarchy* menampilkan struktur objek yang saling berkaitan satu sama lain.
3. *Inspector Window* memungkinkan untuk menampilkan dan mengedit semua properti dari objek yang dipilih saat ini. Karena berbagai jenis objek memiliki rangkaian properti yang berbeda, tata letak dan isi jendela *Inspector* akan bervariasi.
4. *Project Window* menampilkan daftar *Asset* yang tersedia untuk digunakan dalam S*cene*. *Asset* yang telah di *import* akan muncul di sini.

Di dalam *Scene* terdapat *Game Object* dimana tiap instansinya terdapat *Component*. *Component* merupakan bagian dari *Game Object* yang di eksekusi oleh *game engine*. *Component* utama yang pasti ada pada *Game Object* adalah *Transform. Component* ini merupakan item yang memungkinkan *Game Object* tersusun secara hirarkis dengan hubungan *Parent* dan *Child*. *Transform* juga membantu penempatan posisi, rotasi dan skala pada setiap *Game Object* di dalam *Scene.*



Gambar 2.6 : Transform pada Unity 3D

Sumber : (Unity Technologies, 2017)

Pada Gambar 2.6, *Transform* memiliki tiga variabel yang di tampilkan di *Inspector*. Variabel yang di tampilkan tersebut merupakan nilai lokal dari *Transform* itu sendiri. Dari sisi kode, *Transform* memiliki variabel lokal dan global beserta fungsi-fungsi yang dapat membantu menghitung posisi, rotasi dan skala dalam lingkup global maupun lokal.

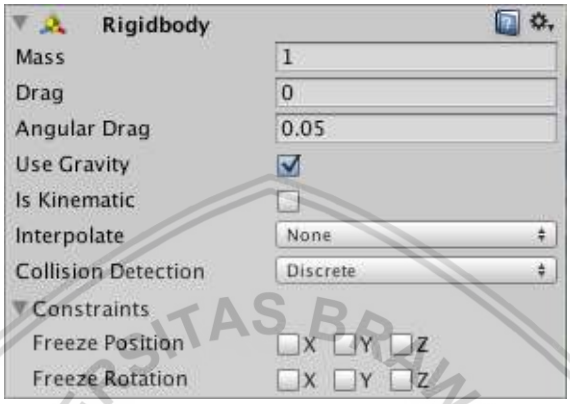
Component dalam Unity 3D di kelompokkan lagi berdasarkan linkup penggunaannya. Dalam penelitian ini, lingkup utama yang akan dibahas adalah Physics. Physics di dalam Unity 3D di bagi menjadi dua macam, yaitu *Rigidbody* dan *Collider*. Penjelasan lebih detil dapat dilihat pada subbab berikut.

2.3.1 *Rigidbody*

Rigidbody merupakan komponen utama dalam penerapan sifat fisika pada *Game Object*. *Game Object* yang telah terpasang *Rigidbody* akan merespon pada gravitasi. Jika pada *Game Object* yang sama terpasang collider atau *Child* dari *Game Object* tersebut terpasang *Collider*, maka *Collider* juga akan.merespon sesuai dengan bentuk dari *Collider* itu sendir.

Pergerakan *Transform* pada *Game Object* yang terpasang *Rigidbody* seluruhnya di kendalikan oleh *Rigidbody.* Sehingga tidak di anjurkan untuk

memodifikasi pergerakan menggunakan *Transform*. Untuk itu *Rigidbody* menyediakan fungsi *Force* untuk menggerakkan *Game Object.* Fungsi tersebut akan menggerakkan *Game Object* sesuai dengan hukum fisika. Jika ingin mengubah sifat fisika menjadi kinematik, maka dapat mencentang variabel *isKinematic*. Fungsi kinematik pada *Rigidbody* berfungsi untuk membuat *Game Object* merespon objek lain yang terpasang *Collider* atau *Rigidbody* dan *Collider* dan mengirimkannya dalam bentuk *event*.



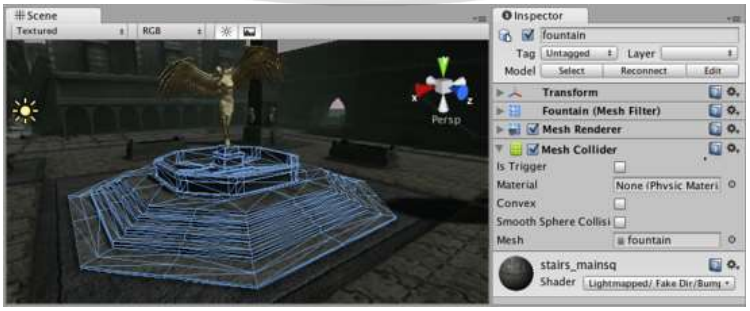
Gambar 2.7 : *Component Rigidbody*

Sumber : (Unity Technologies, 2017)

Pada Gambar 2.7, terdapat beberapa variabel untuk menentukan sifat fisika suatu objek. *Mass* berfungsi sebagai penggatur massa dari objek tersebut. *Drag* berfungsi untuk memberikan hambatan terhadap objek. *Angular Drag* berfungsi untuk memberikan hambatan angular pada objek (Unity Technologies, 2017).

2.3.2 *Collider*

Collider merupakan *Component* yang memberikan bentuk pada suatu objek yang bertujuan untuk memberikan efek persinggungan. Bentuk *Collider* tidak perlu mengikuti bentuk detil dari suatu objek. Cukup dengan perkiraan bentuk dasar dari objek tersebut untuk meringankan beban perhitungan fisika oleh *Game Engine.* Bentuk-bentuk dasar *Collider* yang disediakan Unity 3D antara lain adalah *Box Collider, Sphere Collider* dan *Capsule Collider*.



Gambar 2.8 : *Mesh Collider*

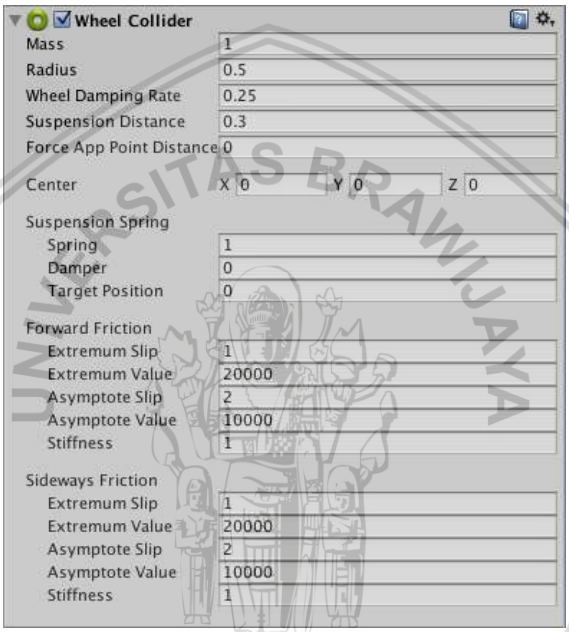
Sumber : (Unity Technologies, 2017)

Untuk kasus tertentu dimana bentuk terlalu kompleks jika menggunakan bentuk *Collider* dasar seperti pada Gambar 2.8, maka dapat digunakan *Mesh*

Collider. *Mesh Collider* mengikuti bentuk objek berdasar data *Mesh* dari objek tersebut. Penggunaan *Mesh Collider* tidak di anjurkan pada objek dinamis, karena akan membebani sistem untuk memproses objek tersebut. Sehingga dianjurkan untuk mengubah *Game Object* yang telah terpasang *Mesh Collider* menjadi *static* (Unity Technologies, 2017).

2.3.3 *Wheel Collider*

Unity 3D juga memiliki *Component Collider* yang di khususkan untuk kendaraan darat yang disebut dengan *Wheel Collider*. *Collider* ini memiliki sistem *Collision Detection* sendiri untuk memproses responnya. Terdapat banyak parameter pada *Component* ini.



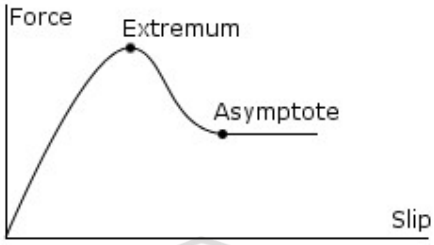
Gambar 2.9 : *Wheel Collider*

Sumber : (Unity Technologies, 2017)

Berdasarkan Gambar 2.9, Terdapat banyak parameter untuk menentukan sifat *Wheel Collider*. Parameter-parameter tersebut antara lain adalah:

1. *Mass* merupakan massa dari roda.
2. *Radius* merupakan jari-jari dari roda.
3. *Wheel Damping Rate* berfungsi sebagai nilai peredam.
4. *Suspension Distance* adalah jarak maksimal ekstensi dari suspensi roda.
5. *Force App Point Distance* berfungsi untuk menentukan point dimana gaya roda diterapkan.
6. *Center* merupakan titik tengah roda pada lingkup objek lokal.
7. *Suspension Spring* mendapatkan penambahan gaya pegas dan peredaman untuk mencapai posisi target.
8. *Forward* dan *Sideways Friction* merupakan properti untuk memberikan gaya gesek depan belakang dan samping.

Terdapat kurva yang menggambarkan gaya gesek pada *Wheel Collider.* Terdapat dua kurva pada gaya gesek, yaitu depan belakang dan samping. Dari kedua arah gaya gesek tersebut, ditentukan seberapa banyak *slip* yang terjadi antara ban dan jalan. Banyaknya nilai *slip* tersebut dapat digunakan untuk menentukan gaya ban atau *tire force* yang terjadi terhadap jalan.



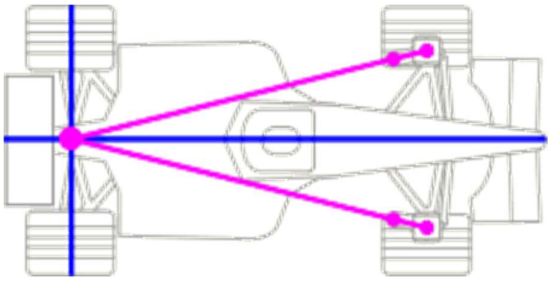
Gambar 2.10 : Kurva Gaya Gesek Pada *Wheel Collider*

Nilai *slip* yang telah difinisikan dimasukkan kedalam kurva pada Gambar 2.10 untuk mendapatkan nilai *force*. Kurva tersebut menggunakan pendekatan dua potongan *spline*. Potongan pertama adalah dari (0,0) menuju (extremum slip, extremum value) dimana nilai tangen kurva adalah nol. Kemudian potongan kedua dimulai dari (extremum slip, extremum value) menuju (asymtote slip, asymtote value) dimana nilai tangen juga berakhir nol.

Sifat dari roda pada dunia nyata adalah ketika nilai *slip* rendah, maka ban akan memberikan gaya dorong yang besar, sedangkan jika nilai slip terlalu tinggi, maka gaya dorong pada ban akan berkurang (Unity Technologies, 2017).

2.4 *Steering Geometry*

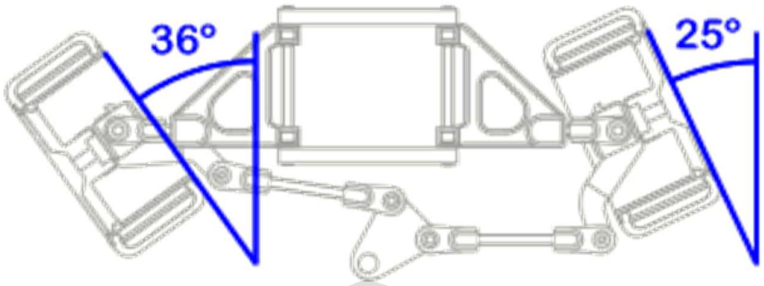
Steering Geometry adalah suatu bentuk matematis dari tata letak sistem kendali kendaraan beroda empat. Secara sederhana terdapat dua roda pendorong yang berada di belakang dan dua rode depan yang berfungsi sebagai *steer.* Jika lengan steer pada masing-masing roda memiliki sudut 90^0, maka sudut putaran roda akan sejajar satu sama lain. Terdapat permasalahan pada model ini karena ketika mobil berbelok, masing-masing roda depan akan menghasilkan kurvatur tikungan yang berbeda. Sehingga arah tikungan mobil sulit di prediksi yang di karenakan slip pada roda. Oleh karena itu Rudolf Ackerman mengembangkan model *Ackerman Steering* untuk menghindari permasalahan tersebut (Burnhill, 2009).



Gambar 2.11 : *Ackerman Steering*

Sumber : (Burnhill, 2009)

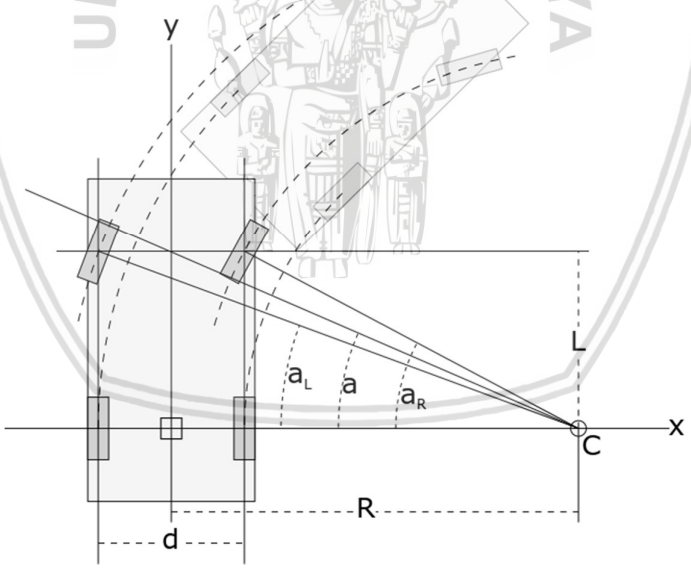
Pada model *Ackerman Steering*, sudut lengan kedua roda depan mengacu pada titik tengah di antara kedua roda belakang. Pada Gambar 2.11, gari berwarna pink merepresentasikan sudut lengan. Ketika *steer* di putar, maka tiap roda akan memiliki sudut yang sedikit berbeda.



Gambar 2.12 : Sudut Putaran Roda Menggunakan *Ackerman Steering*

Sumber : (Burnhill, 2009)

Pada Gambar 2.12, ketika steer di putar ke arah kiri, maka sudut putaran pada roda kiri lebih besar dari pada sudut putaran roda kanan. Hal ini merupakan hasil dari perbedaan sudut lengan pada tiap roda depan. Perhitungan sudut tiap roda dapat dilihat pada Persamaan



Gambar 2.13 : Perubahan Posisi Kendaraan Berdasarkan Radius (R)

Sumber : (Zhao , et al., 2013)

Gambar 2.13 mengilustrasikan kondisi geometri ideal untuk kendaraan yang berbelok ke kanan dimana titik pusat putaran berada pada titik C. Hubungan antara a_L dengan a_R adalah sebagai berikut:

$$L\cot(a_L) - L\cot(a_R) = d$$
$$\cot(a_L) - \cot(a_R) = \frac{d}{L} \quad \textbf{(2.1)}$$

Nilai L pada Persamaan (2.1) merupakan jarak antara roda depan dan roda belakang atau *wheelbase,* d merupakan jarak antara kedua roda depan. a_L dan a_R menunjukkan sudut dari roda kiri dan roda kanan. Persamaan (2.1) merupakan kriteria ideal untuk *Ackerman Turning* (Zhao , et al., 2013).

Jika nilai sudut awal yang diketahui adalah a, maka dapat menggunakan persamaan berikut:

$$R = L\cot(a) \tag{2.2}$$

$$\tan(a_L) = \frac{L}{R + \frac{d}{2}} \tag{2.3}$$

$$\tan(a_R) = \frac{L}{R - \frac{d}{2}}$$

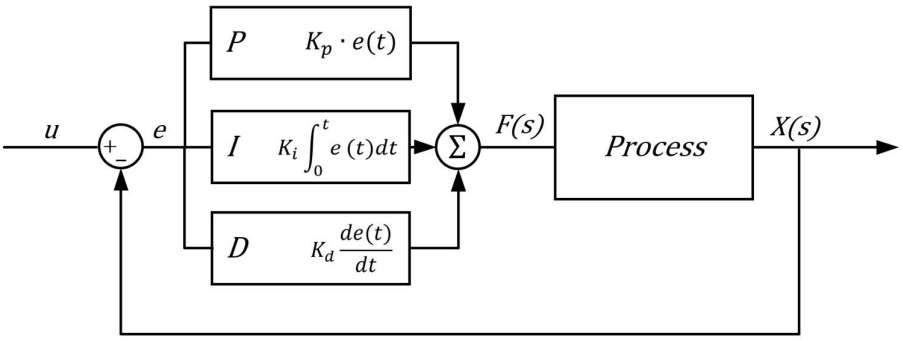
Langkah untuk mendapatkan sudut tiap roda adalah dengan mencari nilai R dengan menggunakan Persamaan (2.2). Kemudian nilai R tersebut dapat digunakan untuk mendapatkan. a_L dan a_R.

2.5 *PID Controller*

PID Controller merupakan sistem kontrol yang membantu proses industri mengendalikan berbagai macam variabel proses. Banyaknya penggunaan *PID Controller* di latarbelakangi kesederhanaan sistem. Terdapat tiga parameter utama yang perlu di atur atau tunning, yaitu faktor Proporsional (P), faktor Integral (I) dan faktor Derivatif (D). Struktur PID Controller yang paling ideal dan sering digunakan pada banyak penerapan adalah sebagai berikut:

$$CO(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{2.4}$$

CO(t) pada Persamaan (2.4) merupakan nilai output dari kontrol. $e(t)$ merupakan selisih antara SetPoint dengan nilai output proses atau dapat disebut dengan *error*. K_p, K_i dan K_d nilai gain Proporsional, Integral dan derivatif (Setiawan, 2008). Masing-masing parameter kontrol ini mempunyai fungsi dan keunggulan yang berbeda. Nilai faktor Proporsional (P) mempunyai keunggulan *rise time* yang cepat, faktor Integral (I) mempunyai keunggulan untuk memperkecil *error* ,dan faktor Derivatif (D) mempunyai keunggulan untuk memperkecil error atau meredam *overshot/undershot*. Oleh karena itu agar mendapatkan hasil output dengan *rise time* secepat mungkin dan error yang sekecil mungkin, maka ketiga aksi kontrol tersebut di gabung menjadi PID Controller seperti pada Gambar 2.14. Kontrol Proporsional Integral derivative (PID) selalu menggunakan parameter berdasarkan tinjauan terhadap karakteristik yang di atur (*plant*). Dengan demikian bagaimanapun rumitnya suatu plant, prilaku *plant* tersebut harus di ketahui terlabih dahulu sebelum pencarian parameter PID itu dilakukan.



Gambar 2.14 : Struktur dasar PID Controller

2.5.1 Kontrol Proporsional

Pengontrol proposional memiliki keluaran yang sebanding atau proposional dengan besarnya sinyal kesalahan (selisih antara besaran yang di inginkan dengan harga aktualnya). Secara lebih sederhana dapat dikatakan bahwa keluaran pengontrol proporsional merupakan perkalian antara konstanta proposional dengan masukannya. Perubahan pada sinyal masukan akan segera menyebabkan sistem secara langsung mengeluarkan output sinyal sebesar konstanta pengalinya.

Sinyal keasalahan (error) merupakan selisih antara besaran setting dengan besaran aktualnya. Selisih ini akan mempengaruhi pengontrol, untuk mengeluarkan sinyal positif (mempercepat pencapaian harga setting) atau negatif (memperlambat tercapainya harga yang diinginkan).

Ketika konstanta proporsional bertambah semakin tinggi, pita proporsional menunjukkan penurunan yang semakin kecil, sehingga lingkup kerja yang dikuatkan akan semakin sempit.

Ciri-ciri pengontrol proposional harus diperhatikan ketika pengontrol tersebut diterapkan pada suatu sistem. Secara eksperimen, pengguna pengontrol propoisional harus memperhatikan ketentuan-ketentuan berikut ini :

1. Jika nilai Kp kecil, pengontrol proposional hanya mampu melakukan koreksi kesalahan yang kecil, sehingga akan menghasilkan respon sisitem yang lambat.
2. Jika nilai Kp dinaikan, respon sistem menunjukan semakin cepat mencapai set point dan keadaan stabil.
3. Jika nilai Kp diperbesar sehingga mencapai harga yang berlebiahan, akan mengakibatkan sistem bekerja tidak stabil, atau respon sistem akan berosolasi

2.5.2 Kontrol Integral

Pengontrol integral berfungsi menghasilkan respon sistem yang memiliki kesalahan keadaan stabil nol. Jika sebuah plant tidak memiliki unsur integrator (1/s), pengontrol proposional tidak akan mampu menjamin keluaran sistem dengan kesalahan keadaan stabilnya nol. Dengan pengontrol integral, respon sistem dapat diperbaiki, yaitu mempunyai kesalahan keadaan stabilnya nol.

Pengontrol integral memiliki karaktiristik seperti halnya sebuah integral. Keluaran sangat dipengaruhi oleh perubahan yang sebanding dengan nilai sinyal kesalahan. Keluaran pengontrol ini merupakan penjumlahan yang terus menerus dari perubahan masukannya. Kalau sinyal kesalahan tidak mengalami perubahan, keluaran akan menjaga keadaan seperti sebelum terjadinya perubahan masukan.

Ketika digunakan, pengontrol integral mempunyai beberapa karakteristik berikut ini:

1. Keluaran pengontrol membutuhkan selang waktu tertentu, sehingga pengontrol integral cenderung memperlambat respon
2. Ketika sinyal kesalahan berharga nol, keluaran pengontrol akan bertahan pada nilai sebelumnya.
3. Jika sinyal kesalahan tidak berharga nol, keluaran akan menunjukkan kenaikan atau penurunan yang dipengaruhi oleh besarnya sinyal kesalahan dan nilai Ki.
4. Konstanta integral Ki yang berharga besar akan mempercepat hilangnya offset. Tetapi semakin besar nilai konstanta Ki akan mengakibatkan peningkatan osilasi dari sinyal keluaran pengontrol.

2.5.3 Kontrol Derivatif

Keluaran pengontrol Derivative memiliki sifat seperti halnya suatu operasi differensial. Perubahan yang mendadak pada masukan pengontrol, akan mengakibatkan perubahan yang sangat besar dan cepat. Ketika masukannya tidak mengalami perubahan, keluaran pengontrol juga tidak mengalami perubahan, sedangkan apabila sinyal masukan berubah mendadak dan menaik (berbentuk fungsi step), keluaran menghasilkan sinyal berbentuk impuls. Jika sinyal masukan berubah naik secara perlahan (fungsi ramp), keluarannya justru merupakan fungsi step yang besar magnitudnya sangat dipengaruhi oleh kecepatan naik dari fungsi ramp dan faktor konstanta diferensialnya.

Karakteristik pengontrol derivative adalah sebagai berikut:

1. Pengontrol ini tidak dapat menghasilkan keluaran bila tidak ada perubahan pada masukannya (berupa sinyal kesalahan).
2. Jika sinyal kesalahan berubah terhadap waktu, maka keluaran yang dihasilkan pengontrol tergantung pada nilai Td dan laju perubahan sinyal kesalahan.
3. Pengontrol derivative mempunyai suatu karakter untuk mendahului, sehingga pengontrol ini dapat menghasilkan koreksi yang signifikan sebelum pembangkit kesalahan menjadi sangat besar. Jadi pengontrol derivative dapat mengantisipasi pembangkit kesalahan, memberikan aksi yang bersifat korektif, dan cenderung meningkatkan stabilitas sistem .

Berdasarkan karakteristik pengontrol tersebut, pengontrol derivative umumnya dipakai untuk mempercepat respon awal suatu sistem, tetapi tidak memperkecil kesalahan pada keadaan stabilnya. Kerja pengontrol derivative hanyalah efektif pada lingkup yang sempit, yaitu pada periode peralihan. Oleh

sebab itu pengontrol derivative tidak pernah digunakan tanpa ada pengontrol lain sebuah sistem(Setiawan, 2008).

2.6 Catmull-Rom Spline

Catmull-Rom Spline merupakan salah satu bagian dari *Spline Interpolation* yang merupakan fungsi khusus yang didefinisikan secara parsial oleh polinomial. Pada notasi standar yang sering digunakan, polinomial pada *Spline Interpolation* memiliki parameter **t.** Sehingga persamaan segmen adalah $\mathrm{S(t)} = \mathrm{T} \cdot \mathrm{C}$ di mana T adalah vektor yang mengandung potensi parameter $[t^3 \quad t^2 \quad t \quad 1]$ dan C adalah matriks koefisien. Matriks koefisien dapat dijabarkan menjadi matriks basis yang dikalikan dengan matriks geometris. Sehingga persamaan segmen juga dapat dituliskan dengan $\mathrm{S(t)} = \mathrm{T} \cdot \mathrm{M} \cdot \mathrm{G}$ dimana M adalah matriks basis dan G adalah matriks geometris seperti pada Persamaan (2.1).

$$\mathbf{S(t)} = [\mathbf{t^3} \quad \mathbf{t^2} \quad \mathbf{t} \quad \mathbf{1}] \begin{bmatrix} \mathbf{m_{00}} & \mathbf{m_{01}} & \mathbf{m_{02}} & \mathbf{m_{03}} \\ \mathbf{m_{10}} & \mathbf{m_{11}} & \mathbf{m_{12}} & \mathbf{m_{13}} \\ \mathbf{m_{20}} & \mathbf{m_{21}} & \mathbf{m_{22}} & \mathbf{m_{23}} \\ \mathbf{m_{30}} & \mathbf{m_{31}} & \mathbf{m_{32}} & \mathbf{m_{33}} \end{bmatrix} \begin{bmatrix} \mathbf{G_0} \\ \mathbf{G_1} \\ \mathbf{G_2} \\ \mathbf{G_3} \end{bmatrix} \quad \textbf{(2.5)}$$

Pada *Catmull-Rom Spline* memiliki basis nilai untuk M. Sedangkan G pada *Catmull-Rom* dilambangkan dengan P atau disebut juga dengan *Point*. P sendiri terdiri dari barisan *control point* P_0, P_1, P_2, P_3. Sehingga *Catmull-Rom Spline* dapat dijabarkan dengan persamaan (2.2)(Sprunk, 2008):

$$\mathrm{P(t)} = [\mathrm{t}^3 \quad \mathrm{t}^2 \quad \mathrm{t} \quad 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{P_0} \\ \mathrm{P_1} \\ \mathrm{P_2} \\ \mathrm{P_3} \end{bmatrix} \quad \textbf{(2.6)}$$

2.7 *Path Planning*

Path Planning bertanggung jawab merancang jalan yang akan dilalui kendaraan. Jalan yang dirancang harus sesuai dengan kondisi yang dibutuhkan kendaraan seperti kurvatur tikungan dan batas kecepatan maksimum. Langkah-langkah yang perlu diambil untuk membuat *Path Planning* antara lain:

1. Menentukan titik-titik yang akan menjadi arah yang harus dilalui kendaraan.
2. Membuat jalan berdasarkan titik-titik yang telah ditentukan dan menghubungkannya menjadi suatu kurvatur.
3. Menentukan kemiringan jalan berdasarkan kondisi jalan pada arena simulasi.

Setelah *path* dibuat, langkah selanjutnya adalah menentukan petunjuk kecepatan maksimum yang harus ditempuh kendaraan sepanjang jalan. Pengukuran batas maksimum kecepatan menggunakan selisih nilai tangen dari kurvatur path. Jika menggunakan Catmull-Rom, maka rumus yang digunakan adalah rumus turunan pertama P(t)'dari Persamaan (2.2) yang dapat dilihat pada Persamaan (2.3).

$$P(t)' = [3t^2 \quad 2t \quad 1 \quad 0]\frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad \textbf{(2.7)}$$

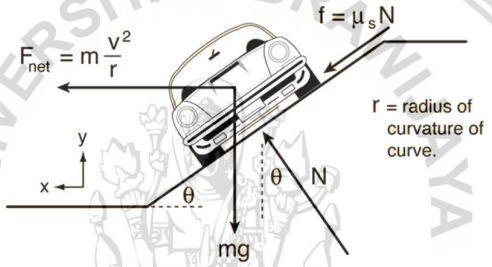
Penentuan kecepatan juga dipengaruhi gaya sentripetal. Gambar 2.10 menunjukkan mobil dengan gravitasi(g) dan radius kurvatur(r) dapat ditentukan kecepatan minimum ketika berbelok dengan rumus sebagai berikut:

- Jika permukaan jalan lurus, maka persamaan yang digunakan adalah Persamaan (2.4).

$$V_{MAX} = \sqrt{rg\,\mu_s} \quad \textbf{(2.8)}$$

- Jika permukaan jalan miring, maka persamaan yang digunakan adalah Persamaan (2.5)

$$V_{MAX} = \sqrt{\frac{rg(\sin(\theta) + \mu_s\cos(\theta))}{\cos(\theta) - \mu_s\sin(\theta)}} \quad \textbf{(2.9)}$$



Gambar 2.15 : Kendaraan ketika di jalan miring

Masing-masing data nilai kecepatan yang diperoleh kemudian dibandingkan antara kecepatan pada kurvatur jalan dan kecepatan berdasarkan rumus setripetal. Langkah selanjutnya adalah tugas navigator untuk mengarahkan kendaraan sesuai dengan data path planning. (Lekkas, 2014)

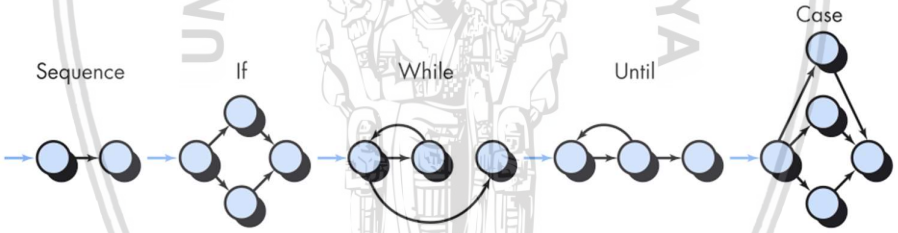
2.8 Pengujian *White Box*

Pengembangan perangkat lunak merupakan rangkaian proses produksi perangkat lunak. Proses ini dilakukan oleh manusia dimana masih ada kemungkinan untuk melakukan kesalahan. Kesalahan tersebut dapat muncul pada awal proses produksi yang menyebabkan tujuan tidak sesuai dengan hasil atau bahkan tujuan tidak terspesifikasi secara tepat. Faktor kesalahan ini menyebabkan pengembangan perangkat lunak memerlukan aktivitas penjaminan kualitas. Pengujian perangkat lunak merupakan elemen penting dalam menjamin kualitas perangkat lunak yang merepresentasikan tinjauan utama spesifikasi, desain dan pembuatan kode program. Salah satu cara untuk menguji perangkat lunak tersebut dengan menggunakan pendekatan cara kerja internal produk tersebut. Pengujian dapat dilakukan dengan memastikan setiap operasi internal berjalan sesuai dengan spesifikasi. Pendekatan ini dapat disebut juga dengan pengujian *White Box* (Pressman, 2005).

Pengujian *White Box* adalah metode perancangan kasus uji yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan kasus uji. Pengujian *White Box* juga dapat disebut juga dengan pengujian *Glass Box*. Kasus uji yang dihasilah dapat digunakan sebagai:

1. Pemberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan minimal satu kali.
2. Penguji untuk memastikan semua keputusan logis pada percabangan kode.
3. Penguji setiap perulangan pada batasan mereka dan pada batas operasional mereka.
4. Penguji struktur data internal untuk menjamin validitasnya.

Dalam penelitian ini, metode pengujian *White Box* yang digunakan adalah pengujian *basis path*. Metode ini memungkinkan perancang kasus uji untuk memperoleh ukuran kompleksitas logis dari sebuah rancangan prosedural. Hasil ukuran tersebut dapat digunakan sebagai pedoman untuk mendefinisikan *basis set* dari jalur eksekusi. Kasus uji dibuat untuk menjamin *basis set* mengeksekusi setiap baris statmen di dalam program minimal satu kali selama pengujian. Metode *basis path* dilakukan dengan membuat notasi yang merepresentasikan aliran kontrol yang disebut dengan *flow graph*. *Flow graph* dapat dijadikan sebagai alat untuk memudahkan pembacaan aliran kontrol. Struktur *flow graph* ditunjukkan pada Gambar 2.9.



Gambar 2.16 : *Flow Graph*

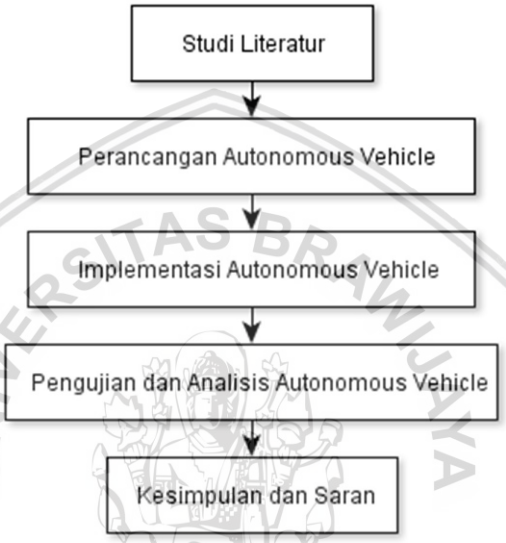
Sumber : (Pressman, 2005)

Setelah flow graph didefinisikan, langkah selanjutnya adalah menentukan ukuran kompleksitas atau *cyclomatic complexity*. Saat digunakan dalam konteks pengujian *basis path*, nilai untuk *cyclomatic complexity* mendefinisikan jumlah jalur independen atau *independent path* dalam *basis set* program dan memberikan batas atas jumlah pengujian yang harus dilakukan untuk memastikan semua statment telah dieksekusi minimal satu kali. *Independent path* adalah sembarang *path* di dalam program yang memberikan paling sedikit satu set statmen proses baru atau kondisi baru. *Complexity* dapat dihitung dengan beberapa cara, diantaranya (Pressman, 2005):

1. Jumlah region pada *flow graph* sesuai degnan *cyclomatic complexity*.
2. *Cyclomatic complexity* V(G) untuk grafik G adalah V(G) = E – N + 2, dimana E adalah jumlah *edge* dan N adalah jumlah *node*.
3. *Cyclomatic complexity* juga dapat di definisikan dengan V(G) = P + 1, dimana P adalah jumlah *predicate node* yang ada pada *flow graph*.

BAB 3 METODOLOGI

Bab metodologi akan membahas metode dan alasan digunakannya metode tersebut. Pembahasan meliputi studi literatur yang akan menjabarkan metode Steering Behavior beserta metode pendukung lainnya. Pembahasan dilanjutkan dengan langkah-langkah penelitian berupa perancangan dan implementasi *Autonomous Vehicle*. Hasil dari implementasi akan didukung dengan pengujian. Alur tahapan penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 : Alur tahapan penelitian

3.1 Studi Literatur

Penerapan Autonomous Vehicle membutuhkan metode khusus yang mendekati dengan sifat tersebut. Steering Behavior digunakan karena memiliki urutan kerja yang serupa dengan Autonomous Vehicle. Metode ini juga dapat menjadi pendukung penerapan Autonomous Vehicle pada simulasi yang akan di rancang. Tetapi untuk menggunakan menggunakan metode ini diperlukan metode pendukung agar dapat beradaptasi dengan lingkup simulasi yang diinginkan. Metode pendukung tersebut antara lain adalah sebagai berikut.

1. Steering Geometry
2. Proportional Integral Derivative Controller
3. Catmull-Rom Spline
4. Path Planning

3.2 Perancangan Autonomous Vehicle

Penerapan Autonomous Vehicle dibagi menjadi beberapa komponen. Pembagian ini bertujuan agar mempermudah proses implementasi. Komponen

yang dimaksut adalah komponen kendali mobil atau Car Controller, Steering Behavior dan Waypoint.

Car Controller pada Autonomous Vehicle adalah penggerak dari mobil. Nilai yang akan dihasilkan oleh Steering Behavior akan dikalkulasikan oleh komponen ini. Hasil dari kalkulasi tersebut akan dijadikan acuan untuk menggerakkan mobil kedepan dan mengendalikan arah mobil. Kalkulasi tersebut melibatkan PID Controller dan Ackerman Steering Geometry.Sehingga mobil dapat bergerak seperti di dunia nyata.

Steering Behavior merupakan pusat dari Artificial Intelligence dari Autonomous Vehicle. Steering Behavior bertugas mengumpulkan data dari lingkungan berupa informasi mobil-mobil lain yang ada didekatnya dan juga informasi seputar jalur yang akan lewati dari komponen Waypoint. Komponen ini akan menghasilkan nilai berupa target kecepatan dan steer angle berdasarkan informasi yang sudah di olah.

Waypoint adalah komponen penyedia jalur navigasi untuk Autonomous Vehicle. Komponen ini menyediakan informasi berupa posisi target yang akan dituju. Informasi posisi target yang diberikan akan selalu diperbarui seiring dengan terlewatinya jalur yang sudah dilewati sebelumnya. Komponen ini juga dapat memberikan informasi posisi pada jarak tertentu. Infomasi itu akan digunakan sebagai perencanaan jalur dan kecepatan yang akan dicapai oleh Autonomous Vehicle.

3.3 Implementasi Autonomous Vehicle

Implementasi merupakan penerapan langkah-langkah proses yang telah dijabarkan di perancangan. Penerapan ini menggunakan Platform Unity3D sebagai perantara. Bahasa pemrograman yang digunakan adalah C# yang merupakan bahasa dasar pada Unity3D. Sumber daya yang diperlukan untuk implementasi ini berasal dari Standart Asset yang telah disediakan Unity3D dan Vehicle Physics Toolkit yang juga disediakan Unity3D di Asset Store secara gratis.

3.4 Pengujian dan Analisis Autonomous Vehicle

Metode Steering Behavior yang telah ditetapkan harus melalui tahap pengujian. Pengujian ini dilakukan untuk menjawab permasalahan yang telah di rumuskan. Pengujian dilakukan berdasarkan sekenario. Terdapat tiga pengujian untuk mengetahui kinerja program. Macam pengujian tersebut antara lain sebagai berikut:

1. Pengujian White Box
2. Pengujian Akurasi dan Presisi
3. Pengujian Performa

Langkah selanjutnya yang akan dilakukan adalah analisis hasil pengujian. Analisis dilakukan agar mendapatkan kesimpulan yang akan menjawab permasalahan yang telah dirumuskan.

3.5 Kesimpulan dan Saran

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem aplikasi telah selesai dilakukan. Kesimpulan diambil untuk menjawab rumusan masalah yang telah ditetapkan sebelumnya. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

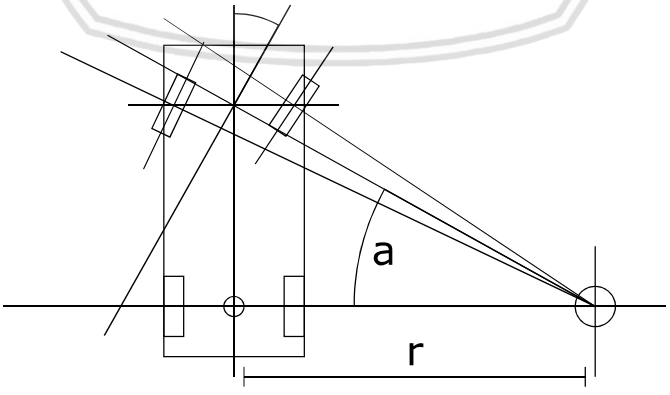
Bab ini akan menjelaskan langkah-langkah perancangan *Autonomous Vehicle* menggunakan metode *Steering Behavior* dan proses implementasinya pada simulasi tiga dimensi. Pada subbab perancangan akan menjelaskan detil dari perancangan *Autonomous Vehicle*. Subbab implementasi menunjukkan detil proses implementasi dan hasilnya pada simulasi.

4.1 Perancangan *Autonomous Vehicle*

Autonomous Vehicle adalah agen kendaraan yang dapat bergerak secara otomatis setelah memproses informasi yang didapatkan dari lingkungannya. Metode yang digunakan untuk menerapkan *Autonomous Vehicle* adalah *Steering Behavior*. Metode ini pada dasarnya adalah untuk pergerakan karakter. Untuk mengadaptasi metode ini pada mobil diperlukan komponen khusus sebagai penggerak yaitu *Car Controller*. Kemudian komponen utama sebagai otak dari kendaraan ini adalah *Steering Behavior*. Kendaraan yang akan disimulasikan bergerak mengikuti jalur yang telah ditetapkan. Komponen untuk jalur ini adalah *Waypoint*.

4.1.1 Perancangan *Car Controller*

Car Controller adalah dasar dari penggerak khusus kendaraan sejenis mobil. Mobil pada intinya memiliki tubuh sebagai pusat fisika (*rigidbody*) dan roda yang merupakan inti dari penggerak mobil ini. Dua roda belakang pada mobil berfungsi untuk memberikan dorongan mobil kearah depan. Titik tengan antara dua roda belakang akan dijadikan titik pusat koordinat untuk menghitung semua kalkulasi matematis *Steering Behavior*. Dua roda depan memiliki fungsi utama sebagai penentu arah tikungan mobil. Titik tengah diantara dua roda depan berfungsi sebaai titik acuan sudut putaran setir atau *steer angle* (a) mobil berdasarkan radius tikungan atau *turn radius* (r). Skema rancangan mobil dapat dilihat pada Gambar 4.1.



Gambar 4.1 : Skema rancangan mobil untuk *Car Controller*

Rigidbody adalah pusat dari kalkulasi fisika pada mobil. *Rigidbody* di Unity3D memiliki beberapa atribut utama yaitu massa, drag sebagai hambatan dan angular

drag sebagai habatan perputaran. Untuk menggerakkan rigidbody diperlukan gaya dorong. Mobil memiliki sumber pendorong dari dua roda belakang. Roda ini mendapatkan gaya dorongannya dari torsi dan gaya gesek permukaan ban dengan permukaan aspal. Unity3D sudah memiliki komponen sebagai simulator roda yang bernama *Wheel Collider*. Sehingga untuk mendapatkan sifat mobil yang mendekati dunia nyata maka atribut-atribut tiap komponen perlu diisi sesuai tabel berikut.

Tabel 4.1 : Nilai Atribut Untuk *Rigidbody*

Atribut	Nilai
Mass	1500
Drag	0.1
Angular Drag	0.05

Nilai atribut untuk *Wheel Collider* untuk masing-masing roda adalah sebagai berikut.

Tabel 4.2 : Nilai Atribut Untuk *Wheel Collider*

Atribut		Nilai
Mass		250
Radius		0.53
Wheel Damping Rate		5
Suspension Distance		0.25
Force App Point Distance		0.25
Suspension Spring	Spring	35000
	Damper	3500
	Target Position	0.5
Forward Friction	Extemum Slip	0.4
	Extemum Value	1
	Asymptote Slip	0.8
	Asymptote Value	0.75
	Stiffness	1.75
Sideways Friction	Extemum Slip	0.25
	Extemum Value	1
	Asymptote Slip	0.5

	Asymptote Value	1
	Stiffness	2

4.1.1.1 Perancangan *Steer Controller*

Steer Controller adalah kendali arah untuk mobil. *Controller* ini diperlukan karena *Steering Behavior* menyampaikan pesan untuk berbelok berupa nilai sudut tikungan (a) atau *steer angle* seperi pada gambar 4.1. Sebelum menterjemahkan nilai tersebut ke *Wheel Collider*, diperlukan beberapa atribut pendukung. Atribut pertama adalah pembatas maksimal sudut setir atau maximum steer angle agar sudut tikungan tidak melebihi batas maksimal. Atribut selanjutnya adalah *angular steer magnitude* sebagai kendali kecepatan sudut *steer angle* untuk memberikan kesan *smooth* dalam menyetir. Atribut ketiga adalah *target steer angle* sebagai acuan kemana arah putaran *steer angle*. Atribut ini merupakan variabel yang akan digunakan sebagai *Steering Behavior* sebagai masukan.

Distribusi nilai *steer angle* ke *Wheel Collider* pada kedua roda depan menggunakan *Ackerman Steering Geometry*. Nilai dari *steer angle* (a) dan jarak antara roda depan dengan belakang (L) akan di hitung menggunakan metode tersebut sehingga menghasilkan *turn radius* (r). Variabel inilah yang akan digunakan untuk menentukan nilai steer angle pada masing-masing *Wheel Collider*. Penggunaan metode ini bertujuan untuk mengurangi slip antara roda dengan aspal ketika berbelok tajam sehingga meningkatkan akurasi arah belokan.

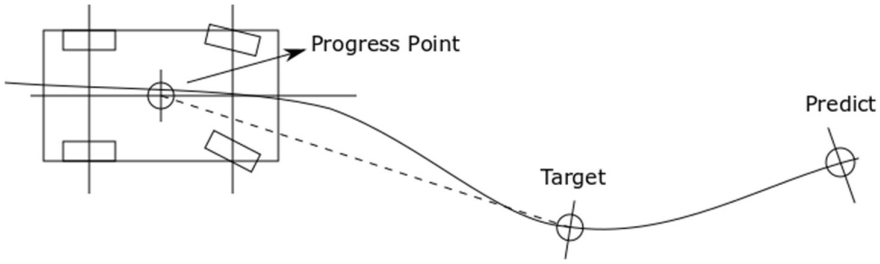
4.1.1.2 Perancangan *Acceleration Controller*

Wheel Collider memiliki nilai keluaran berupa revolution per minute (rpm) dan gaya dorong dalam bentuk Newton. Untuk mendapatkan keluaran tersebut variabel motor torque harus di beri nilai dalam satuan *torque* (Nm). Nilai positif untuk variabel ini berfungasi sebagai putaran kedepan begitu juga sebaliknya. Nilai keluaran rpm perlu di kendalikan untuk memenuhi kebutuhan kecepatan yang diinginkan *Steering Behavior*. Oleh karena itu *PID Controller* digunakan untuk menerapkan kendali kecepatan.

Acceleration Controller membutuhkan atribut untuk kendali kecepatan. Atribut tersebut antara lain adalah *pid gain value* untuk *acceleration* dan *pid gain value* untuk *breaking*. Nilai acuan sebagai nilai input kedua pid controller tersebut di ambil dari *target magnitude*.

4.1.2 Perancangan *Steering Behavior*

Steering Behavior berfungsi untuk mengolah hasil pembacaan lingkungan menjadi hasil keluaran berupa variabel sudut belokan dan variabel magnitude. Data yang diambil berupa empat titik transform dan array yang berisikan data hasil dari path-planning. Titik-titik transform memberikan data berupa arah *Target*(T), Progress Point(P), titik posisi mobil saat ini(C) dan titik *Predict*(F). titik F, P dan T hasil dari kalkulasi path-planning.



Gambar 4.2 : Rancangan Steering Behavior

Path-Planning merupakan kumpulan data dari hasil proses menelusuri *waypoint*. Data dimulai dari titik P hingga 100 point didepan(F), kemudian dibagi menjadi 100 titik. Tiap titik terdiri dari posisi(p), arah(tan), magnitude(mag), dan sudut kemiringan jalan(roll). Kumpulan data disimpan dalam bentuk antrian(*Queue*). Setiap mobil melewati titik awal, maka titik awal tersebut akan di hapus dari antrian(*Dequeue*) dan menambahkan data baru yang di letakkan di paling depan(*Enqueue*). Nilai T didapatkan dari data kesepuluh dari titik P.

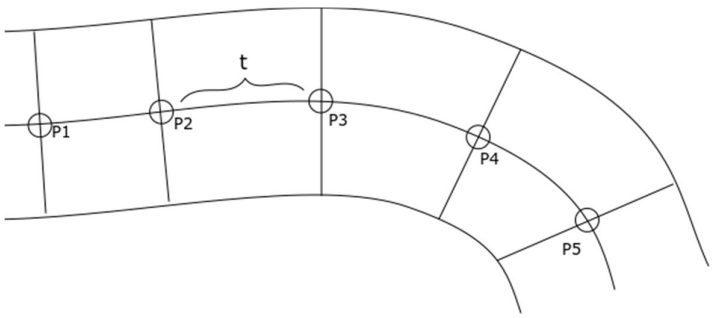
Sudut (a) merupakan sudut roda setir agar dapat berbelok mengikuti kurvatur dari *waypoint*. Nilai a didapat dari perhitungan *steer geometry* dari titik C menuju titik T berdasarkan arah mobil. Hasil kalkulasi tersebut menghasilkan nilai besaran sudut yang diperlukan untuk berbelok.

Nilai *Magnitude* merupakan kecepatan yang harus di tempuh oleh mobil. *Magnitude* berubah-ubah seiring dengan besarnya sudut belokan dan kemiringan jalan. Dari kumpulan data path-planning terdapat variabel tan dan roll. Variabel ini dikalkulasikan menggunakan rumus fisika sentripetal sehingga menghasilkan nilai magnitude dan disimpan pada variabel mag. Kemudian dari ke 100 data tersebut dicari nilai mag terkecil sehingga mendapatkan magnitude yang diinginkan.

4.1.3 Perancangan *Waypoint*

Waypoint merupakan sekumpulan titik yang terhubung satu sama lain sehingga membentuk suatu jalur. Bentuk data *waypoint* adalah *array* dari *transform* sebagai perwakilan satu point, variabel I untuk menunjukan index dari point tersebut dan variabel t untuk penunjuk titik diantara dua titik I. Fungsi-fungsi yang diperlukan untuk mendapatkan data tersebut antara lain fungsi pencari titik terdekat, fungsi untuk mendapatkan posisi titik, fungsi untuk mendapatkan sudut kemiringan jalan (roll) dan fungsi untuk memindahkan titik sejauh jarak tertentu.

Fungsi untuk mencari jarak terdekat menggunakan cara penelusuran tiap point dan mendapatkan titik terdekat terhadap mobil. Diawali dengan mencari index point yang paling dekat denga mobil dengan pengulangan. Kemudian mencari titik di antara dua point dengan membandingkan antara titik terdekat dengan titik berikutnya dan titik terdekat dengan titik sebelumnya. Hasil pencarian tersebut menghasilkan nilai t.



Gambar 4.3 : Rancangan *Waypoint*

Informasi yang didapatkan dari waypoint berupa index(I) dan titik antara(t). Sehingga untuk mendapatkan posisi titik, diperlukan fungsi yang mengolah informasi tersebut untuk mendapatkan data berupa posisi. Rumus yang digunakan untuk mencari posisi ini adalah *Catmull-rom*. Hasil dari rumus ini terdiri dari posisi dan arah(tan).

Fungsi untuk mendapatkan sudut kemiringan jalan didapatkan dari sudut *euler* z pada *transform*. Nilai yang diambil merupakan nilai yang berada di antara dua point pada titik t. Hasil yang didapat berupa sudut kemiringan terhadap sudut normal jalan. Nilai ini berguna untuk menentukan kecepatan mobil pada kemiringan tertentu.

Mobil mendapatakan nilai posisi ketika mobil tersebut bergerak. Oleh karena itu diperlukan fungsi yang dapat mengambil data posisi ketika mobil sedang berjalan. Dari perpindahan mobil tersebut didapatkan nilai jarak. Nilai ini di kalkulasikan dengan memindahkan point I dan t. Hasil kalkulasi tersebut berupa I dan t baru. Untuk mendapatkan posisinya menggunakan fungsi untuk mendapatkan posisi titik.

4.2 Implementasi *Autonomous Vehicle*

Dalam tahap implementasi ini akan dilakukan penerapan dari perancangan sebelumnya. Pada subbab ini akan dijelaskan spesifikasi perangkat keras dan perangkat lunak untuk mengimplementasi A*utonomous Vehicle*. Kemudian akan dijelaskan setiap kode program yang telah di implementasi beserta penerapannya pada *Autonomous Vehicle.*

4.2.1 Spesifikasi Perangkat Keras

Berikut ini merupakan lingkungan perangkat keras yang digunakan untuk mengimplementasi *Autonomous Vehicle*.

1. Prosesor: Inter(R) Core(TM) i5-2450M CPU @2.50GHz (4CPUs)
2. RAM: 4,00 GB
3. Graphic Card: Nvidia Geforce GT 630M 2GB
4. Monitor: 1366x768 (32bit) 60Hz
5. Kapasistas Harddisk: 750 GB

4.2.2 Spesifikasi Perangkat Lunak

Berikut ini merupakan perangkat lunak yang digunakan untuk mengimplementasi *Autonomous Vehicle*.

1. Sistem Operasi Windows 10 Pro 64 bit version 1703
2. Unity 2017.2.0f3 versi free.
3. MonoDevelop dengan bahasa pemrograman C#

4.2.3 Implementasi *Car Controller*

Implementasi *Car Controller* berfungsi untuk merealisasikan nilai input yang diberikan pada mobil. Implementasi *Car Controller* terbagi menjadi tiga fungsi, yaitu fungsi *Apply Steer*, *Apply Acceleration*, dan *Apply Brake*.

4.2.3.1 Fungsi *Apply Steer*

Apply Steer merupakan perintah untuk menerapkan hasil kalkulasi dari nilai *Target Steer Angle* pada *Wheel Collider*. Terdapat dua mode dalam penerapan tersebut, yaitu dengan menggunakan kalkulasi *Ackerman Geometry* dan dengan menggunakan kalkulasi pendekatan sudut posisi target terhadap arah mobil.

Kode Program 4.1 : Fungsi *Apply Steer*

```
1  void ApplySteer ()
2  {
3      m_steerAngle = Mathf.MoveTowards (m_steerAngle,
   targetSteerAngle, maxSteerAngularMagnitude * Time.fixedDeltaTime);
4      m_steerAngle = Mathf.Clamp (m_steerAngle, -maxWheelSteerAngle,
   maxWheelSteerAngle);
5      m_turnRadius = CarGeometry.GetTurnRadius (m_frontLocalPos -
   m_rearLocalPos, m_steerAngle * Mathf.Deg2Rad);
6      if (useAckerman) {
7          FLWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_flLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
8          FRWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_frLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
9          RLWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_rlLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
10         RRWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_rrLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
11     } else {
12         FLWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_frontLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
13         FRWheelCollider.steerAngle = FLWheelCollider.steerAngle;
14         RLWheelCollider.steerAngle =
   CarGeometry.GetYAngleByPosition (m_rearLocalPos -
   m_steerCenterLocalPos, m_turnRadius);
15         RRWheelCollider.steerAngle = RLWheelCollider.steerAngle;
16     }
17 }
```

Penjelasan kode program 4.1 untuk implementasi fungsi *Apply Steer* adalah sebagai berikut:

Baris 3 : Menginterpolasikan *Target Steer Angle* untuk memperhalus perubahan *Steer Angle*.

Baris 4 : Membatasi *Steer Angle* agar tidak melebihi batas maksimum *Steer Angle*.

Baris 5 : Mendapatkan nilai *Turn Radius* dari kalkulasi *Steer Angle* dengan posisi tengah poros roda depan dan posisi tengah poros roda belakang. Nilai *Turn Radius* merupakan dasar untuk menentukan sudut belokan pada roda.

Baris 6-10 : Menerapkan sudut belokan pada masing-masing roda menggunakan kalkulasi *Ackerman Geometry* untuk mengurangi selip terhadap permukaan jalan.

Baris 11-15 : Menerapkan sudut belokan pada masing-masing roda menggunakan pendekatan sudut posisi target terhadap arah mobil.

4.2.3.2 Fungsi *Apply Acceleration*

Apply Acceleration merupakan perintah untuk menerapkan hasil kalkulasi *Target Magnitude* dengan menggunakan rumus *PID Controller* agar menghasilkan nilai *Torque* untuk menggerakkan mobil.

Kode Program 4.2 : Fungsi *Apply Acceleration*

```
 1  void ApplyAcceleration ()
 2  {
 3      float mt, clampTargetMagnitude;
 4      clampTargetMagnitude = Mathf.Min (maxMagnitude, Mathf.Abs
    (targetMagnitude));
 5      clampTargetMagnitude *= Mathf.Sign (targetMagnitude);
 6      switch (driveType) {
 7      case WheelDriveType.FrontWheelDrive:
 8          m_wheelTorque = motorTorqueController.UpdateLimitedEach
    (m_frontWheelMagnitude, clampTargetMagnitude, maxWheelTorque);
 9          mt = m_wheelTorque / 2f;
10          FLWheelCollider.motorTorque = mt;
11          FRWheelCollider.motorTorque = mt;
12          break;
13      case WheelDriveType.RearWheelDrive:
14          m_wheelTorque = motorTorqueController.UpdateLimitedEach
    (m_rearWheelMagnitude, clampTargetMagnitude, maxWheelTorque);
15          mt = m_wheelTorque / 2f;
16          RLWheelCollider.motorTorque = mt;
17          RRWheelCollider.motorTorque = mt;
18          break;
19      case WheelDriveType.FourWheelDrive:
20          float desiredMagnitude = (m_rearWheelMagnitude +
    m_frontWheelMagnitude) * 0.5f;
21          m_wheelTorque = motorTorqueController.UpdateLimitedEach
    (desiredMagnitude, clampTargetMagnitude, maxWheelTorque);
22          mt = m_wheelTorque / 4f;
23          FLWheelCollider.motorTorque = mt;
24          FRWheelCollider.motorTorque = mt;
25          RLWheelCollider.motorTorque = mt;
26          RRWheelCollider.motorTorque = mt;
27          break;
28      }
```

```
29 }
```

Penjelasan kode program 4.2 untuk implementasi fungsi *Apply Acceleration* adalah sebagai berikut:

Baris 4-5 : Membatasi nilai *Target Magnitude* agar tidak melebihi batas maksimal *Magnitude*.

Baris 7-12 : Menkalkulasikan *Target Magnitude* menggunakan *PID Controller* dengan nilai awal dari *magnitude* roda depan. Kemudian menerapkannya pada masing-masing roda depan dengan membagi hasil kalkullasi dengan 2 sebagai *sistem Front Wheel Drive.*

Baris 13-18 : Menkalkulasikan *Target Magnitude* menggunakan *PID Controller* dengan nilai awal dari *magnitude* roda belakang. Kemudian membagi hasil kalkulasi dengan 2 dan diterapkan pada masing-masing roda belakang sebagai sistem *Rear Wheel Drive.*

Baris 19-27 : Menkalkulasikan *Target Magnitude* menggunakan *PID Controller* dengan nilai awal dari rata-rata *magnitude* roda depan dan roda belakang. Kemudian membagi hasil kalkulasi dengan 4 dan diterapkan pada keempat roda sebagai sistem *Four Wheel Drive*.

4.2.3.3 Fungsi *Apply Break*

Apply Break merupakan perintah untuk mengurangi laju mobil dengan menerapkan nilai *Break Torque*. Fungsi ini berfungsi untuk membantu mobil dalam mengurangi laju mobil lebih cepat.

Kode Program 4.3 : Fungsi *Apply Break*

```
 1 void ApplyBrakes ()
 2 {
 3     if (Mathf.Abs (targetMagnitude) < Mathf.Abs
   (m_localVelocity.z) ||
 4         (targetMagnitude < 0 && m_localVelocity.z > 0) ||
 5         (targetMagnitude > 0 && m_localVelocity.z < 0)) {
 6         m_brakeTorque = Mathf.Clamp (
 7             brakeTorqueFactor * Mathf.Abs (targetMagnitude -
   m_localVelocity.z),
 8             0, maxBrakeTorque) / 4f;
 9     } else {
10         m_brakeTorque = 0;
11     }
12     FLWheelCollider.brakeTorque = m_brakeTorque;
13     FRWheelCollider.brakeTorque = m_brakeTorque;
14     RLWheelCollider.brakeTorque = m_brakeTorque;
15     RRWheelCollider.brakeTorque = m_brakeTorque;
16 }
```

Penjelasan kode program 4.3 untuk implementasi fungsi *Apply Break* adalah sebagai berikut:

Baris 3 : Menentukan kondisi mobil apakah *Target Magnitude* kurang dari *magnitude* mobil atau *Target Magnitude* berlawanan

dengan *magnitude* mobil. Jika salah satu atau semua bernilai *true*, maka menuju baris 6.

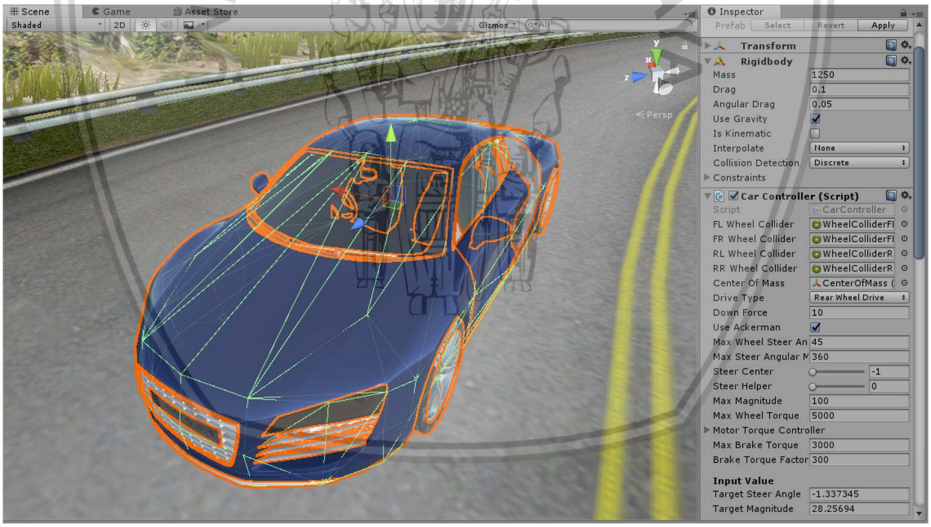
Baris 6-8 : Menkalkulasikan *Brake Torque* dengan mengalikan selisih antara *Target Magnitude* dan *magnitude* mobil dengan *Brake Torque Factor*. Kemudian membatasi *Brake Torque* agar tidak melebihi batas maksimum *Brake Torque*. Nilai *Brake Torque* dibagi 4 sesuai dengan jumlah roda.

Baris 9-10 : Jika kondisi mobil sedang *Accleration*(baris 3 bernilai false), maka nilai *Brake Torque* adalah 0.

Baris 12-15 : Menerapkan *Brake Torque* pada masing-masing roda.

4.2.3.4 Implementasi *Car Controller* pada *Autonomous Vehicle*

Car Controller yang telah di buat kemudian di implementasikan pada *Autonomous Vehicle*. Objek mobil harus memiliki komponen *Rigidbody* dan *Collider* agar dapat merespon terhadap gravitasi. Penempatan *Rigidbody* terdapat pada *Parent* dari objek mobil. Sedangkan penempatan Collider dapat di tempatkan pada *Child* dari objek mobil. Penempatan *Mesh* juga diperlukan untuk memberikan bentuk detil mobil seperti badan mobil dan keempat roda. Penampakan implementasi *Car Controller* dapat dilihat pada Gambar 4.4.



Gambar 4.4 : Implementasi *Car Controller* Pada *Autonomous Vehicle*

Car Controller di tempatkan pada satu object dimana terdapat komponen *Rigidbody*. Pada Gambar 4.4, terdapat empat referensi objek yang harus dimasukkan dimana objek-objek tersebut merupakan *Wheel Collider* yang bertindak sebagai roda. Nilai properties dari *Rigidbody* dan *Wheel Collider* dapat diisikan dengan nilai yang telah dijabarkan pada perancangan. Setiap atribut untuk *Car Controller* memiliki fungsi sebagai berikut:

1. *Drive Type* : Berfungsi untuk menentukan tipe pendorong mobil yang terdiri dari tiga, yaitu *Rear Wheel Drive, Front Wheel Drive dan Four Wheel Drive*.

2. *Down Force* : Berfungsi untuk memberikan tekanan kebawah mobil sesuai dengan kecepatan mobil dalam satuan *Newton*.
3. *Use Ackerman* : Menentukan cara steer berbelok apakah menggunakan *paralel seering* atau *ackerman steering*.
4. *Max Wheel Steer Angle* : Berfungsi untuk memberi batasan sudut tikungan yang dapat dilakukan mobil.
5. *Max Wheel Angular Magnitude* : Berfungsi untuk membatasi kecepatan putaran *steer*.
6. *Steer Center* : Menentukan titik pusat mobil untuk berbelok. Nilai -1 untuk titik pusat di roda belakang dan nilai 1 untuk titik pusat di roda depan.
7. *Steer Helper* : Berfungsi untuk membantu menstabilkan mobil sesuai dengan *velocity* jika bernilai 1.
8. *Max Magnitude* : Membatasi batas kecepatan maksimal mobil dalam satuan meter per detik.
9. *Max Wheel Torque* : Membatasi torsi yang dapat diberikan pada mobil.
10. *Motor Torque Controller* : Sistem kendali torsi mesin berdasarkan nilai perlambatan *target magnitude.*
11. *Max Brake Torque* : Membatasi kemampuan mobil untuk mengerem.
12. *Brake Torque Factor* : Merupakan pengali kekuatan rem mobil berdasarkan perlambatan kecepatan mobil.

Nilai atribut yang dimasukkan pada Car Controller diisikan dengan nilai standar yang dianjurkan. Nilai masukan dapat dilihat pada Tabel 4.3

Tabel 4.3 : Data Nilai Atribut *Car Controller*

Atribut	Nilai
Drive Type	Rear Wheel Drive
Down Force	10
Use Ackerman	true
Max Wheel Steer Angle	45
Max Wheel Angular Magnitude	360
Steer Center	-1
Steer Helper	0
Max Magnitude	100
Max Wheel Torque	5000
Motor Torque Controller	P=0, I=500, D=0
Max Brake Torque	3000
Brake Torque Factor	300

4.2.4 Implementasi *Steering Behavior*

Implementasi *Steering Behavior* diawali dengan pengumpulan data kondisi lingkungan. Kemudian merencanakan gerakan dan akhirnya di terapkan. Implementasi ini terdiri dari tiga fungsi, yaitu *Waypoint Progress Tracker*, *Steer Calculation*, dan *Magnitude Calculation*. *Steer Calculation* dan *Magnitude Calculation* berfungsi untuk merencanakan sekaligus menerapkan hasil kalkulasi.

4.2.4.1 Fungsi *Waypoint Progress Tracker*

Waypoint Progress Tracker merupakan perintah untuk mengetahui posisi *real-time* mobil terhadap jalur yang dilaluinya beserta hasil kalkulasi posisi bantu untuk merencanakan aksi selanjutnya.

Kode Program 4.4 : Fungsi *Waypoint Progress Tracker*

```
 1  void WaypointProgressTracker ()
 2  {
 3      float wayTrough;
 4      Vector3 tangent;
 5      Vector3 positionTemp;
 6      positionTemp = waypoint.GetMovePointPosition (indexWaypoint,
    tWaypoint, targetDistance, out tangent);
 7      positionTemp += Quaternion.AngleAxis (90, Vector3.up) *
    Vector3.ProjectOnPlane (tangent, Vector3.up).normalized * offset;
 8      target.position = transform.position + Vector3.ProjectOnPlane
    (positionTemp - transform.position, transform.up).normalized *
    targetDistance;
 9      target.rotation = Quaternion.LookRotation (tangent);
10      positionTemp = waypoint.GetPointPosition (indexWaypoint,
    tWaypoint, out tangent);
11      positionTemp += Quaternion.AngleAxis (90, Vector3.up) *
    Vector3.ProjectOnPlane (tangent, Vector3.up).normalized * offset;
12      progressPoint.position = positionTemp;
13      progressPoint.rotation = Quaternion.LookRotation (tangent);
14      wayTrough = Vector3.Dot (tangent, progressPoint.position -
    transform.position);
15      if (wayTrough < 0) {
16          waypoint.MovePoint (ref indexWaypoint, ref tWaypoint, -
    wayTrough);
17      }
18  }
```

Penjelasan kode program 4.4 untuk implementasi fungsi *Waypoint Progress Tracker* adalah sebagai berikut:

Baris 6 : Mengambil posisi dengan jarak *Target Distance* dari titik posisi *progress* mobil saat ini menggunakan fungsi *Get Move Point Position*. Fungsi ini juga mengembalikan nilai tangen pada titik tersebut.

Baris 7 : Menggeser *Position Temp* dengan jarak *Offset*, tegak lurus terhadap tangen. Hal ini berguna untuk menentukan target ketika mobil berada pada posisi *offset*.

Baris 8-9 : Menerapkan hasil kalkulasi pada *Transform Target* untuk digunakan di fungsi lain.

Baris 10-11 : Mengambil posisi mobil saat ini dan kemudian di geser sejarak *offset* yang tegak lurus dengan tangen.

Baris 12-13 : Menerapkan hasil kalkulasi pada *Transform Progress Point* untuk digunakan di fungsi lain.

Baris 14-16 : Memperbarui posisi terbaru mobil terhadap *waypoint* dengan menghitung jarak tempuh sejak kalkulasi terakhir dan memanggil fungsi *Move Point*.

4.2.4.2 Fungsi *Steer Calculation*

Steer Calculation merupakan fungsi untuk menkalkulasikan *Target Steer Angel* dengan empat macam pendekatan. Tiap pendekatan memiliki bobot yang dapat diatur sesuai kebutuhan. Pendekatan-pendekatan tersebut antara lain berdasarkan *Target Steering* dan *Ackerman Steering*.

Kode Program 4.5 : Fungsi *Steer Calculation*

```
 1  void SteerCalculation ()
 2  {
 3      Vector3 relativeTarget = transform.InverseTransformPoint
    (target.position) - carController.steerCenterLocalPos;
 4      float DegreeSteerAngle = 0;
 5      float div = 0;
 6      if (useTargetSteering) {
 7          DegreeSteerAngle += CarGeometry.GetAngle (relativeTarget)
    * Mathf.Rad2Deg * targetFactor;
 8          div += 1;
 9      }
10      if (useAckermanSteering) {
11          Vector3 relativePoint = transform.InverseTransformPoint
    (transform.position) - carController.steerCenterLocalPos;
12          float newTurnRadius = CarGeometry.GetTurnRadius
    (relativeTarget, relativePoint);
13          DegreeSteerAngle += CarGeometry.GetYAngleByPosition
    (carController.frontLocalPos - carController.rearLocalPos,
    newTurnRadius) * ackermanFactor;
14          div += 1;
15      }
16      if (div > 0) {
17          targetSteerAngle = DegreeSteerAngle / div;
18      } else {
19          targetSteerAngle = 0;
20      }
21  }
```

Penjelasan kode program 4.5 untuk implementasi fungsi *Steer Calculation* adalah sebagai berikut:

Baris 3 : Mengambil posisi target relative terhadap posisi dan arah mobil untuk mendapatkan posisi target lokal.

Baris 6-8 : Jika menggunkan pendekatan *Target Steering*, maka kalkulasikan sudut antara arah mobil dengan arah relative target. Kemudian hasil kalkulasi di kalikan dengan *Target Factor*.

Baris 10-14 : Jika menggunakan pendekatan *Ackerman Steering*, maka kalkulasikan sudut menggunakan *Ackerman Geometry*

berdasarkan *Turn Radius* menuju posisi relative Target. Kemudian kalikan dengan *Ackerman Factor*.

Baris 16-20 : Hitung nilai rata-rata dari masing-masing sudut yang digunakan, kemudian masukkan pada variable *Target Steer Angle*.

4.2.4.3 Fungsi *Magnitude Calculation*

Magnitude Calculation berisi kalkulasi untuk menentukan *Target Magnitude* berdasarkan kurvatur jalan agar control Steer lebih stabil. Didalam fungsi ini terdapat pengulangan untuk menkalkulasikan besarnya sudut antar segmen sebagai acuan penentuan *Target Magnitude*.

Kode Program 4.6 : Fungsi *Magnitude Calculation*

```
1  void MagnitudeCalculation ()
2  {
3      bool isChange = false;
4      while (m_PredictedPoint.Count > 0 && Vector3.Dot
(progressPoint.forward, m_PredictedPoint.PeekHead ().pos -
progressPoint.position) < 0) {
5          m_PredictedPoint.Dequeue ();
6      }
7      while (m_PredictedPoint.Count < predictionLength) {
8          if (m_PredictedPoint.Count == 0) {
9              m_PredictedPoint.Enqueue (new WaypointProgressStruct
(indexWaypoint, tWaypoint, progressPoint.position,
progressPoint.forward, float.MaxValue, 0f, 10f));
10         } else {
11             var pt = m_PredictedPoint.PeekTail ();
12             Vector3 tan, pos, fwdPos;
13             fwdPos = waypoint.GetMovePointPosition (pt.i, pt.t,
targetDistance, out tan);
14             fwdPos += Quaternion.AngleAxis (90, Vector3.up) *
Vector3.ProjectOnPlane (tan, Vector3.up).normalized * wayOffset;
15             predictPoint.position = pt.pos;
16             predictPoint.rotation = Quaternion.LookRotation
(pt.tan);
17             waypoint.MovePoint (ref pt.i, ref pt.t,
predictionDistance);
18             pos = waypoint.GetPointPosition (pt.i, pt.t, out tan);
19             pt.pos = pos + Quaternion.AngleAxis (90, Vector3.up) *
Vector3.ProjectOnPlane (tan, Vector3.up).normalized * wayOffset;
20             pt.radius = CarGeometry.GetTurnRadius
(predictPoint.InverseTransformPoint (fwdPos));
21             pt.roll = waypoint.GetRollAngle (pt.i, pt.t);
22             float tanRoll = Mathf.Tan (pt.roll * -Mathf.Sign
(pt.radius) * Mathf.Deg2Rad);
23             pt.mag = Mathf.Sqrt (Physics.gravity.magnitude *
Mathf.Abs (pt.radius) * (tanRoll + tireFriction) /
24              (1f - tireFriction * tanRoll));
25             pt.tan = tan;
26             m_PredictedPoint.Enqueue (pt);
27         }
28         isChange = true;
29     }
30     if (isChange && m_PredictedPoint.Count == predictionLength) {
31         SmoothingMagnitudePrediction ();
32     }
33     int idxMag = m_PredictedPoint.Count - 1;
```

```
34      targetMagnitude = Mathf.Clamp (m_PredictedPoint.GetElement
(idxMag).mag, -carController.maxMagnitude,
carController.maxMagnitude);
35  }
```

Penjelasan kode program 4.6 untuk implementasi fungsi *Magnitude Calculation* adalah sebagai berikut:

Baris 4-5 : Menghapus segmen-segmen point yang telah dilewati mobil dengan mengambil dot product dari transform mobil dan transform segmen point. Jika nilainya kurang dari 0, maka hapus segmen point.

Baris 7 : Lakukan pengumpulan segmen point selama jumlah data masih kurang dari *Prediction Length*.

Baris 8-9 : Jika data segmen berjumlah 0, maka buat satu data segment point berdasarkan transform mobil saat ini.

Baris 30-31 : Jika ada perubahan data dan data yang terkumpul berjumlah lebih dari *prediction length*, maka lakukan operasi *Smoothing Magnitude Prediction* yang berfungsi untuk merapikan perubahan *magnitude* berdasrkan *accel rate* dan *decel rate*.

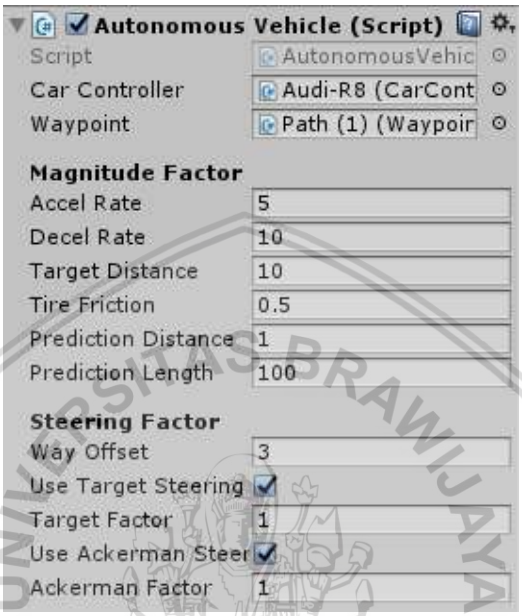
Baris 33-34 : Mengambil nilai magnitude dari segmen pertama *predict point* dan kemudian dimasukkan ke *Target Magnitude.*

4.2.4.4 Implementasi *Steering Behavior* Pada *Autonomous Vehicle*

Seluruh fungsional dari Steering Behavior berada dalam komponen *Autonomous Vehilce*. Komponen ini dimasukkan dalam objek yang sama dimana *Car Controller* berada. Penampakan implementasi *Autonomous Vehicle* dapat dilihat pada Gambar 4.5. Terdapat dua referensi objek yang harus diisikan, yaitu *Car Controller* dan *Waypoint*. Tiap atribut pada *Autonomous Vehicle* memiliki fungsi sebagai berikut:

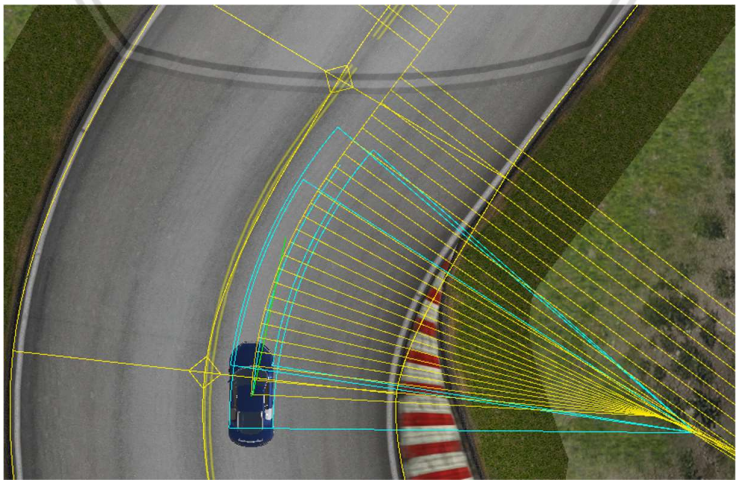
1. *Accel Rate* : Berfungsi untuk menentukan percepatan yang dapat dilakukan mobil.
2. *Decel Rate* : Berfungsi untuk menentukan perlambatan yang dapat dilakukan mobil.
3. *Target Distance* : Jarak Target yang dituju mobil untuk menentukan sudut steer.
4. *Tire Friction* : Untuk menentukan gaya gesek antara ban mobil dan aspal untuk menentukan kecepatan masksimal yang dapat dilakukan ketika berbelok.
5. *Prediction Distance* : Merupakan jarak antar segmen pada *prediction point*.
6. *Prediction Length* : Merupakan jumlah data yang dapat ditampung *prediction point*.
7. *Way Offset* : Berfungsi untuk memberikan nilai offset pada posisi mobil terhadap *waypoint.*
8. *Use Target Steering* : Berfungsi untuk menentukan penggunaan arah target untuk menentukan sudut *steer*.

9. *Target Factor* : Berfungsi untuk menentukan faktor besarnya sudut tikungan.
10. *Use Ackerman Steering* : Berfungsi untuk menentukan penggunaan metode *Ackerman Geometry* untuk mentukan sudut *steer*.
11. *Ackerman Factor* : Berfungsi untuk menentukan faktor besarnya sudut tikungan.



Gambar 4.5 : Implementasi *Steering Behavior* Pada *Autonomous Vehicle*

Nilai atribut pada Gambar 4.5 merupakan nilai default yang telah ditentukan. Jika *Autonomous Vehicle* dijalankan menggunakan nilai atribut tersebut maka hasil dari penerapan tersebut dapat dilihat pada Gambar 4.6. Kurvatur berwarna cyan menunjukkan seberapa besar mobil berbelok. Sedangkan *Prediction Point* akan menghasilkan nilai prediksi yang digambarkan dalam bentuk garis berwarna kuning.



Gambar 4.6 : Penerapan *Autonomous Vehicle* Menggunakan Attribut *Default*

4.2.5 Implementasi *Waypoint*

Waypoint memberikan seluruh informasi tentang jalan yang akan dilewati oleh mobil. Didalam *Waypoint* terdapat daftar Transform yang terletak di sepanjang jalan beserta arah dan sudut kemiringan jalan tersebut. Untuk mendapatkan data informasi waypoint, diperlukan Index Point dan T Point. Index Point merepresentasikan tiap Transform, dan T Point merepresentasikan posisi di antara dua Transform. Terdapat tiga cara untuk mendapatkan informasi *Wapoint*, yaitu dengan fungsi *Get Closest Point*, *Move Point*, dan *Get Point Position*.

4.2.5.1 Fungsi *Get Closest Point*

Get Closest Point merupakan perintah untuk mendapatkan nilai Index Point dan T Point terdekat berdasarkan posisi saat ini. Terdapat parameter res untuk mengatur tingkat akurasi penentuan posisi.

Kode Program 4.7 : Fungsi *Get Closest Point*

```
1  public void GetClosestPoint (out int indexPoint, out float t,
   Vector3 worldPosition, int res=5){
2      indexPoint = 0;
3      t = 0;
4      float distance = float.MaxValue;
5      for (int i=0; i<waypoints.Length; i++) {
6          float temp = Vector3.Distance (waypoints [i].position,
   worldPosition);
7          if (temp < distance) {
8              distance = temp;
9              indexPoint = i;
10         }
11     }
12     float
13     chunk = 1f / (float)res,
14     distF, distB, closest = float.MaxValue;
15     Vector3
16     p0 = waypoints [IndexRepeater (indexPoint - 2,
   waypoints.Length)].position,
17     p1 = waypoints [IndexRepeater (indexPoint - 1,
   waypoints.Length)].position,
18     p2 = waypoints [indexPoint].position,
19     p3 = waypoints [IndexRepeater (indexPoint + 1,
   waypoints.Length)].position,
20     p4 = waypoints [IndexRepeater (indexPoint + 2,
   waypoints.Length)].position;
21     for (int i=0; i<=res; i++) {
22         distF = Vector3.Distance (worldPosition,
   CatmullRomInterpolate (p1, p2, p3, p4, chunk * i));
23         distB = Vector3.Distance (worldPosition,
   CatmullRomInterpolate (p0, p1, p2, p3, 1 - chunk * i));
24         if (distF < closest) {
25             t = chunk * i;
26             closest = distF;
27         }
28         if (distB < closest) {
29             t = -chunk * i;
30             closest = distB;
31         }
32     }
33     if (t < 0) {
34         t += 1;
35         indexPoint--;
```

```
36      }
37      indexPoint = IndexRepeater (indexPoint, waypoints.Length);
38  }
```

Penjelasan kode program 4.7 untuk implementasi fungsi *Get Closest Point* adalah sebagai berikut:

Baris 3-9 : Mencari posisi Index Point terdekat dengan iterasi ke setiap Transform dan membandingkan jarak yang terdekat.

Baris 16-20 : Mengambil sample point dengan asumsi P2 adalah Transform Index Point, P1 adalah Transform pada Index Point -1, P0 adalah Transform pada Index Point -2, P3 adalah Transform pada Index Point +1, P4 adalah Transform pada Index Point +2.

Baris 21-32 : Melakukan pemcarian nilai T Point dari sample point diatas menggunakan Catmull-Rom, sehingga ditemukan titik terdekat. Semakin besar nilai res, semakin akurat nilai T Pointnya.

Baris 33-37 : Memastikan nilai Index Point tidak kurang dari 0 dan tidak lebih dari jumlah total Transform Point.

4.2.5.2 Fungsi *Move Point*

Move Point berfungsi untuk memindah posisi point berdasarkan *Distance*. Nilai yang di kembalikan berupa Index Point dan T Point yang telah diperbarui.

Kode Program 4.8 : Fungsi *Move Point*

```
1   public void MovePoint (ref int indexPoint, ref float t, float
    distance){
2       if (Mathf.Abs (distance) < Mathf.Epsilon) {
3           return;
4       }
5       SafeIndexAndT (ref indexPoint, ref t);
6       float distRemain, distTrough, distNow;
7       if (distance > 0) {
8           distRemain = m_distances [indexPoint] * (1 - t);
9           distTrough = m_distances [indexPoint] * t;
10          distNow = distance;
11          while (distRemain<distNow) {
12              distNow -= distRemain;
13              indexPoint = IndexRepeater (indexPoint + 1,
    waypoints.Length);
14              distRemain = m_distances [indexPoint];
15              distTrough = 0;
16          }
17          t = (distTrough + distNow) / m_distances [indexPoint];
18      } else if (distance < 0) {
19          distRemain = m_distances [indexPoint] * t;
20          distNow = distance;
21          while (distRemain<-distNow) {
22              distNow += distRemain;
23              indexPoint = IndexRepeater (indexPoint - 1,
    waypoints.Length);
24              distRemain = m_distances [indexPoint];
25          }
26          t = (distRemain + distNow) / m_distances [indexPoint];
27      }
28  }
```

Penjelasan kode program 4.8 untuk implementasi fungsi *Move Point* adalah sebagai berikut:

Baris 2-3 : Jika nilai *distance* mendekati nol, maka hentikan proses.

Baris 7-17 : Jika nilai *distance* lebih dari 0 atau kea rah maju, maka kalkulasi dilakukan dengan perhitungan positif.

Baris 18-26 : Jika nilai *distance* kurang dari 0 atau kea rah mundur, maka kalkulasi dilakukan dengan perhitungan negatif.

4.2.5.3 Fungsi *Get Point Position*

Get Point Position berfungsi untuk mengubah nilai Index Point dan T Point menjadi posisi dan tangen.

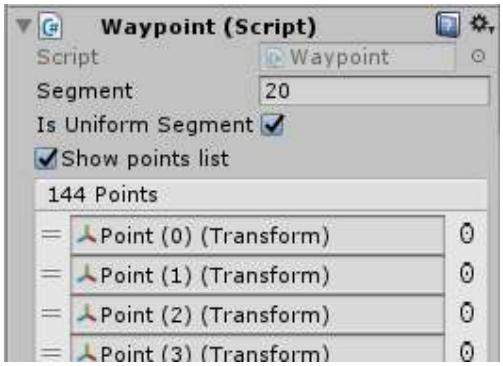
Kode Program 4.9 : Fungsi *Get Point Position*

```
 1  public Vector3 GetPointPosition (int indexPoint, float t, out
    Vector3 tangent){
 2      SafeIndexAndT (ref indexPoint, ref t);
 3      Vector3
 4      p0 = waypoints [IndexRepeater (indexPoint - 1,
    waypoints.Length)].position,
 5      p1 = waypoints [indexPoint].position,
 6      p2 = waypoints [IndexRepeater (indexPoint + 1,
    waypoints.Length)].position,
 7      p3 = waypoints [IndexRepeater (indexPoint + 2,
    waypoints.Length)].position;
 8      tangent = CatmullRomTangent (p0, p1, p2, p3, t).normalized;
 9      return CatmullRomInterpolate (p0, p1, p2, p3, t);
10  }
```

Pada kode program 4.9, kalkulasi dilakukan dengan mengambil empat sample P. P1 adalah Transform *Waypoint* pada Index Point, P0 adalah Transform *Waypoint* pada Index Point-1, P2 adalah Transform pada Index Point+1, P3 adalah Transform pada Index Point+2. Dengan menggunakan *Catmull-Rom*, maka didapatkan nilai posisi dan nilai tangen.

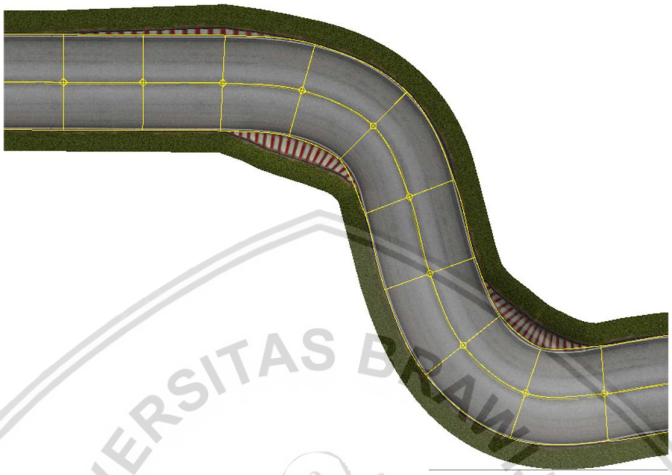
4.2.5.4 Implementasi *Waypoint* Pada Sirkuit

Waypoint diimplementasi dalam bentuk komponen Waypoint. Komponen ini dimasukkan ke objek parent dari kumpulan transform posisi point jalan. Terdapat tiga atribut pada komponen ini, yaitu Segment, Is Uniform Segment dan Points.



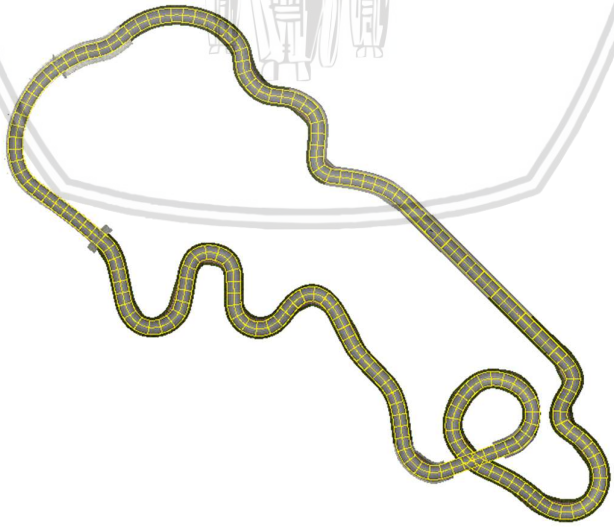
Gambar 4.7 : Implementasi *Waypoint*

Segment pada Gambar 4.7 merupakan jumlah iterasi yang perlu dilakukan Watpoint untuk mengukur jarak tiap poin agar menghasilkan nilai yang akurat. Ada dua tipe penghitungan jarak dan posisi tiap Segment, yaitu dengan perhitungan standar Catmull-Rom dan perhitungan khusus dimana tiap point akan di samakan jaraknya tetapi dengan arah titik yang sama. Untuk mengatur pemilihan tersebut menggunakan atribut Use Uniform Segment.



Gambar 4.8 : Penataan Tiap Point Sepanjang Jalan

Atribut selanjutnya adalah Points yang berisi kumpulan titik jalan. Titik tersebut berupa transform yang ditata ditengah-tengah sepanjang jalan seperti pada Gambar 4.8. Penamaan objek tiap titik harus berurutan agar pembacaan jalan tidak terjadi loop. Seluruh transfor titik tersebut di masukkan di bawah hirarki objek yang memiliki komponen waypoint. Penampakan seluruh penempatan point dapat dilihat pada Gambar 4.9.



Gambar 4.9 : Implementasi *Waypoint* Pada Sirkuit

Titik-titik jalan diimplementasikan pada sirkuit tertutup dari Environtment Vehicle Physics Toolkit. Sirkuit ini akan dijadikan jalan utaman dalam pengujian selanjutnya.

BAB 5 PENGUJIAN DAN ANALISIS

Pada bab ini akan dijelaskan tentang pengujian dan analisa sistem yang telah diimplementasikan pada bab sebelumnya. Pengujian sistem akan dilakukan berdasarkan skenario pengujian.

5.1 Sekenario Pengujian

Sebelum melakukan pengujian terhadap penelitian ini, diperlukan perencanaan tahap-tahap pengujian. Perencanaan pengujian ini bertujuan agar mendapatkan hasil yang sistematis yang kemudian akan di Analisa hasilnya. Oleh karena itu diperlukan perancangan sekenario pengujian untuk menguji hasil implementasi Autonomous Vehicle menggunakan Steering Behavior. Terdapat tiga scenario pengujian sebagai berikut:

1. Skenario Pengujian White Box
2. Skenario Pengujian Akurasi dan Presisi
3. Skenario Pengujian Performa

5.1.1 Skenario Pengujian White Box

Skenario ini bertujuan untuk menguji kompleksitas kode yang telah diimplementasi. Kode yang diuji merupakan kode utama yang menjalankan metode yang telah diimplementasi. Dalam hal ini ada tiga Class utama yang menjalankan peranan penting. Tiap Class akan diuji fungsi inti yang telah mencakup keseluruhan proses. Berikut daftar Class dan fungsinya yang akan diuji:

1. Class Car Controller
    a. Fungsi Apply Steer
    b. Fungsi Apply Acceleration
    c. Fungsi Apply Brake
2. Class Steering Behavior
    a. Fungsi Waypoint Progress Tracker
    b. Fungsi Steer Calculation
    c. Fungsi Magnitude Calculation
3. Class Waypoint
    a. Fungsi Get Closest Point
    b. Fungsi Move Point

5.1.2 Skenario Pengujian Akurasi dan Presisi

Autonomous Vehicle merupakan kendaraan yang bergerak sesuai dengan pembacaan dan perencanaan. Lingkungan yang dimaksud adalah berupa Waypoint yang akan dilewati mobil. Sehingga diperlukan pengujian untuk mengetahui apakah mobil didalam Waypoint. Parameter yang akah dihitung berupa selisih penyimpangan posisi mobil dengan waypoint. Dalam hal ini, batas

toleransi penyimpangan jalur yang diizinkan adalah tidak lebih dari satu meter. Magnitude mobil juga akan di uji akurasinya dengan menghitung percepatannya. Parameter yang akan di ukur adalah percepatan mobil dengan batas toleransi percepatan 10m/s^2 dan perlambatan 20m/s^2. Pengujian akan dilakukan pada kondisi ketika mobil sedang berjalan lurus, kemudian berbelok

5.1.3 Skenario Pengujian Performa

Pengujian performa bertujuan untuk mengetahui kemampuan program dalam mengeksekusi Autonomous Vehicle. Pengujian ini menggunakan metode Benchmarking yang berfokus pada waktu yang diperlukan fungsi untuk memproses perintah tiap framenya. Pengujian dilakukan dengan pendekatan stress testing dengan skenario pengujian terdiri dari dua kondisi, yaitu:

1. Pengujian dilakukan tanpa menyertakan render
2. Pengujian dengan mobil berjumlah 10 hingga 200

Hasil pengujian yang akan diperoleh adalah berupa waktu yang diperlukan untuk menjalankan program setiap frame nya dalam satuan milisecond. Hasil tersebut yang akan digunakan untuk analisis.

5.2 Pengujian

Pengujian adalah langkah untuk mengetahui penerapan Steering Behavior pada *Autonomous Vehicle* berjalan sesuai rencana. Berdasarkan scenario pengujian diatas, maka ada tiga pendekatan pengujian. Pengujian pertama adalah pengujian fungsional terhadap fungsi yang telah diimplementasi. Pengujian non fungsional adalah untuk mengetahui akurasi dan performa *Autonomous Vehicle*.

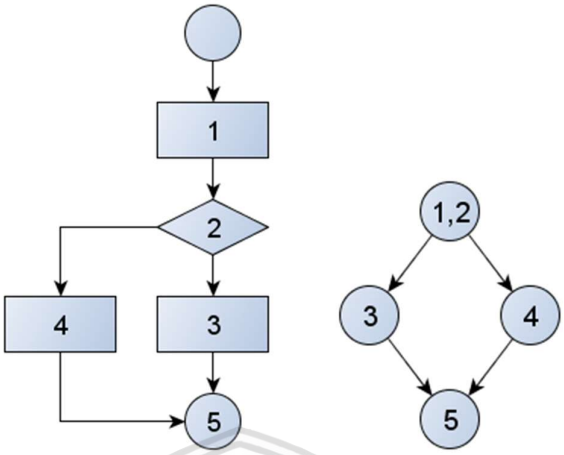
5.2.1 Hasil Pengujian *White Box*

Pengujian *White Box* dilakukan berdasarkan scenario pengujian *White Box* yang telah dijabarkan sebelumnya. Terdapat delapan fungsi utama dari tiga *class* yang diuji. Hasil pengujian fungsi-fungsi adalah sebagai berikut:

1. Fungsi *Apply Steer*

   Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.3.1**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar5.1. Berikut penjelasan tiap nomor node:

   1. Mengambil nilai Target Steer Angle yang telah di perhalus perubahannya dan dibatasi berdasarkan Max Wheel Steer Angle. Kemudian hitung Turn Radius nya untuk di gunakan pada perhitungan berikutnya.
   2. Jika menggunakan Ackerman Geometry, maka lanjut ke node 3.
   3. Terapkan Steer Angle berdasarkan perhitungan Ackerman Geometry.
   4. Jika tidak menggunakan Ackerman Geometry, maka terapkan Steer Angle berdasarkan perhitungan Parallel Geometry.



Gambar 5.1 : Flowchart dan Flowgraph Fungsi Apply Steer

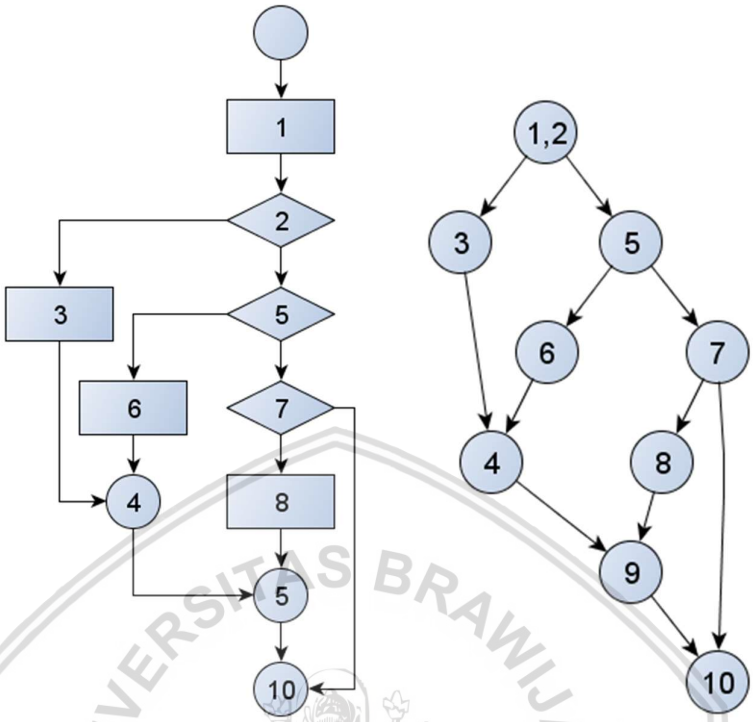
Berdasarkan flowgraph pada Gambar 5.1, jumlah node adalah 4 dan edge adalah 4, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=4-4+2=2. Dari hasil perhitungan tersebut didapatkan 2 basis jalur independen, yaitu:

- Path 1 : 1,2-3-5
- Path 3 : 1,2-4-5

2. Fungsi *Apply Accelration*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.3.2**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.2. Berikut penjelasan tiap nomor node:

5. Mengambil nilai Target Magnitude yang telah di batasi dengan nilai Max Magnitude.
1. Jika menggunakan Front Wheel Drive, maka menuju Node 3.
2. Terapkan Motor Torque pada kedua roda depan dengan hasil perhitungan PID dari Target Magnitude.
3. Node Penghubung.
4. Jika menggunakan Rear Wheel Drive, maka menuju Node 6.
5. Terapkan Motor Torque pada kedua roda belakang dengan hasil perhitungan PID dari Target Magnitude.
6. Jika menggunakan Four Wheel Drive, maka menuju Node 8.
7. Terapkan Motor Torque pada keempat roda dengan hasil perhitungan PID dari Target Magnitude.



Gambar 5.2 : Flowchart dan Flowgraph Fungai Apply Acceleration

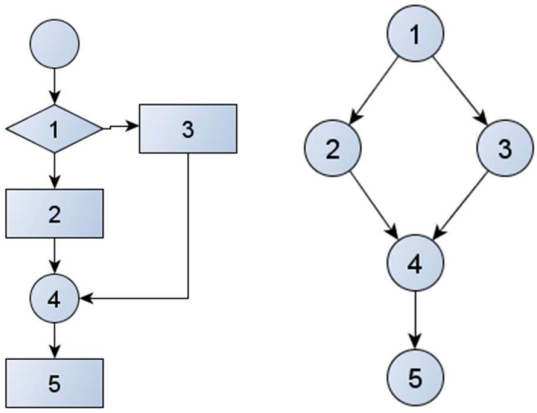
Berdasarkan flowgraph pada Gambar 5.2, jumlah node adalah 9 dan edge adalah 11, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=11-9+2=4. Dari hasil perhitungan tersebut didapatkan 4 basis jalur independen, yaitu:

- Path 1 : 1-2-5-7-10
- Path 2 : 1-2-3-4-9-10
- Path 3 : 1-2-5-8-4-9-10
- Path 4 : 1-2-5-7-8-9-10

3. Fungsi *Apply Brake*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.3.3**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.3. Berikut penjelasan tiap nomor node:

1. Jika nilai absolut Target magnitude kurang dari nilai absolut velocity sumbu z, maka menuju node 2.
2. Hitung Brake Torque berdasarkan error dari Target Magnitude dan velocity sumbu z yang dikalikan dengan Brake Torque Factor.
3. Jika node 1 bernilai false, maka Brake Torque bernilai 0.
4. Terapkan nilai Brake Torque ke Wheel Collider pada keempat roda..



Gambar 5.3 : Flowchart dan Flowgraph Fungsi Apply Brake

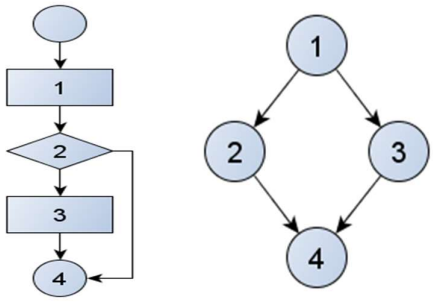
Berdasarkan flowgraph pada Gambar 5.3, jumlah node adalah 5 dan edge adalah 5, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=5-5+2=2. Dari hasil perhitungan tersebut didapatkan 2 basis jalur independen, yaitu:

- Path 1 : 1-2-4-5
- Path 2 : 1-3-4-5

4. Fungsi *Waypoint Progress Tracker*

Flowchart dan flowgraph pada Gambar5.4, didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.4.1**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.4. Berikut penjelasan tiap nomor node:

1. Menghitung Target Position berdasarkan perhitungan nilai dari Target Distance yang didapatkan dari Waypoint. Menghitung Progress Point Position berdasarkan posisi mobil saat ini yang didapat dari Waypoint. Kemudian cari nilai dot product dari tangen Progress Point dan posisi mobil saat ini dan dimasukkan dalam variabel WayTrough.
2. Jika WayTrough lebih kecil dari 0, maka menuju node 3.
3. Panggil fungsi Move Point yang menghasilkan indeks waypoint dan t waypoint baru berdasarkan WayTrough.



Gambar 5.4 : Flowchart dan Flowgraph Fungsi Waypoint Progress Tracker

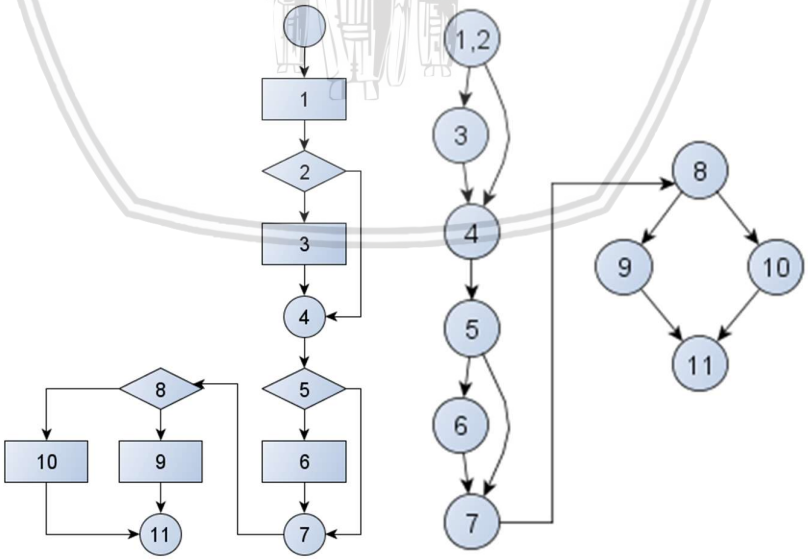
Berdasarkan flowgraph pada Gambar 5.4, jumlah node adalah 4 dan edge adalah 4, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=4-4+2=2. Dari hasil perhitungan tersebut didapatkan 2 basis jalur independen, yaitu:

- Path 1 : 1-2-4
- Path 2 : 1-3-4

5. Fungsi *Steer Calculation*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.4.2**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.5. Berikut penjelasan tiap nomor node:

1. Menghitung nilai relative target berdasarkan posisi target relatif terhadap posisi mobil. Set Degree Steer Angle dengan nilai 0.
2. Jika UseTargetSteering bernilai true, maka menuju node 3.
3. Hitung sudat antara arah mobil dan arah target, kemudian tambahkan ke Degree Steer Angle. Tambahkan nilai div dengan 1
4. Node penghubung.
5. Jika UseAckermanSteering bernilai true, maka menuju node 6.
6. Hitung ackerman steering geometry, kemudian tambahkan ke Degree Steer Angle. Tambahkan nilai div dengan 1.
7. Node penghubung.
8. Jika nilai div lebih besar dari 0, maka menuju node 9.
9. Hitung Target Steer Angle berdasarkan Degree Steer Angle dibagi dengan div.
10. Jika tidak, maka nilai Target Steer Angle adalah 0.



Gambar 5.5 : Flowchart dan Flowgraph Fungsi Steer Calculation

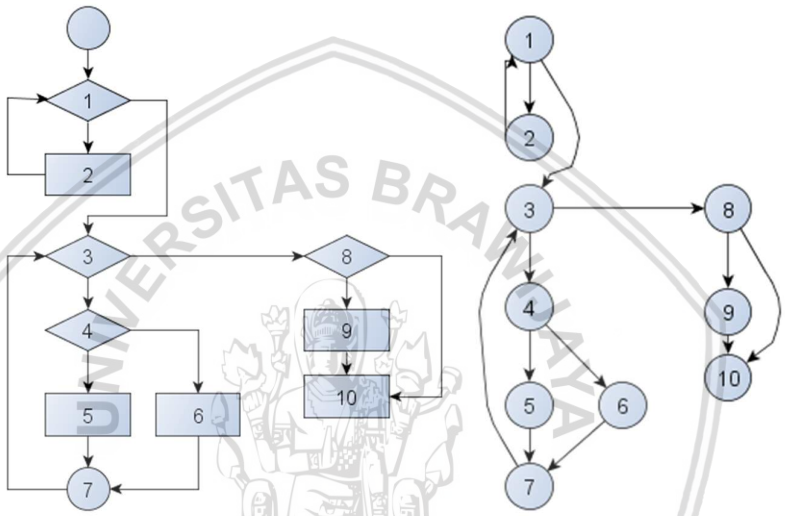
Berdasarkan flowgraph pada Gambar 5.5, jumlah node adalah 11 dan edge adalah 12, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=12-

11+2=3. Dari hasil perhitungan tersebut didapatkan 5 basis jalur independen, yaitu:

- Path 1 : 1-2-4-5-7-8-10-11
- Path 2 : 1-2-3-4-5-7-8-9-11
- Path 3 : 1-2-3-4-5-6-7-8-9-11

6. Fungsi *Magnitude Calculation*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.4.3**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.6.



Gambar 5.6 : Flowchart dan Flowgraph Fungsi Magnitude Calculation

Berikut penjelasan tiap nomor node:

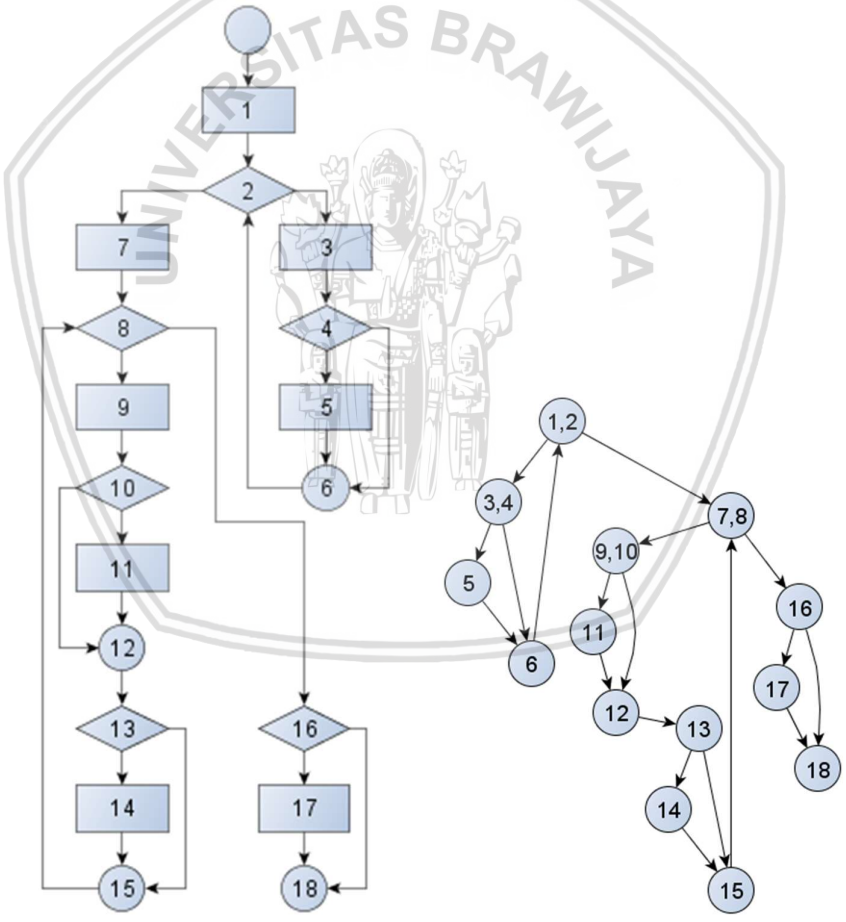
1. Jika Jumlah Predict Point lebih dari 0 dan dot product antara forward progresspoint dengan posisi progress point terhadap nilai head, maka menuju node 2. Jika tidak, menuju node 3.
2. Dequeue data yang pertama dimasukkan atau head.
3. Jika Jumlah Predict Point kurang dari predict length, maka menuju node 4. Jika tidak, menuju node 8.
4. Jika predict point berjumlah 0, maka menuju node 5. Jika tidak, menuju node 6.
5. Enqueue data pertama ke dalam daftar predict point.
6.
7. Node penghubung. Menuju node 3.
8. Jika ada perubahan pada predict point dan jumlah predict point sama dengan predict length, maka menuju node 9. Jika tidak, menuju node 10.
9. Perhalus data magnitude di predict point dengan iterasi.
10. Terapkan nilai Target Magnitude berdasarkan data pada head predict point..

Berdasarkan flowgraph pada Gambar 5.6, jumlah node adalah 10 dan edge adalah 13, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=13-10+2=5. Dari hasil perhitungan tersebut didapatkan 5 basis jalur independen, yaitu:

- Path 1 : 1-3-4-5-7-3-8-9-8-10
- Path 2 : 1-3-4-6-7-3-8-9-8-10
- Path 3 : 1-2-1-3-4-6-7-3-8-9-8-10
- Path 4 : 1-2-1-3-4-5-7-3-8-9-8-10
- Path 5 : 1-2-1-3-8-10

7. Fungsi *Get Closest Point*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.5.1**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.7.



Gambar 5.7 : Flowchart dan Flowgraph Fungsi Get Closest Point

Berikut penjelasan tiap nomor node:

1. Set nilai distance menjadi 0.

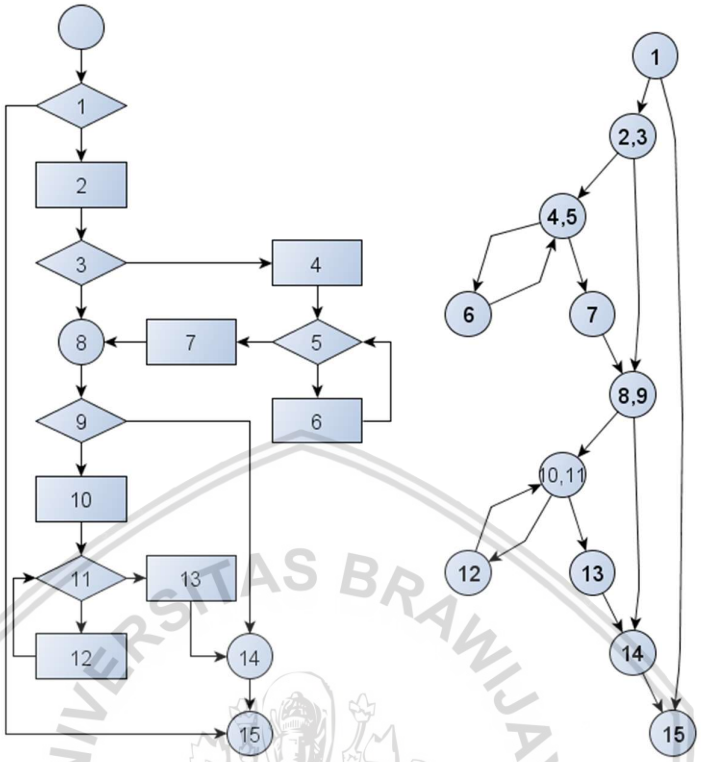
2. Lakukan perulangan pada data waypoints, selama kondisi memenuhi, menuju node 3.
3. Hitung jarak antara posisi point dengan world position.
4. Jika distance lebih kecil, maka menuju node 5.
5. Set nilai distance dan indeks point;.
6. Node penghubung.
7. Hitung nilai chunk bersdasarkan res. Set nilai distF, distB dan closest menjadi nilai maksimal. Ambil nilai posisi dari waypoint dan masukkan ke p0, p1, p2, p3, p4.
8. Lakukan iterasi sebanyak nilai res.
9. Mendapatkan nilai distF dari perhitungan jarak antara world position dengan nilai posisi hasil perhitungan catmull rom. Mendapatkan nilai distB dari perhitungan jarak antara worls position dengan nilai posisi hasil perhitungan catmull rom.
10. Jika nilai distF lebih kecil dari nilai closest, maka menuju node 11.
11. Dapatkan nilai t dari chunk dikalikan indeks iterasi. Masukkan nilai distF ke variabel closest.
12. Node penghubung.
13. Jika nilai distB lebih kecil dari nilai closest, maka menuju node 14.
14. Dapatkan nilai t dari chunk dikalikan indeks iterasi. Masukkan nilai distB ke variabel closest.
15. Node penghubung.
16. Jika nilai t kurang dari 0, maka menuju node 17.
17. Tambahkan nilai t dengan 1 dan kurangi indeks point dengan 1.
18. Terapkan nilai indeks point setelah menggunakan fungsi indeks repeater.

Berdasarkan flowgraph pada Gambar 5.7, jumlah node adalah 14 dan edge adalah 19, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=19-14+2=7. Dari hasil perhitungan tersebut didapatkan 7 basis jalur independen, yaitu:

- Path 1 : 1-2-7-8-16-18
- Path 2 : 1-2-3-4-8-2-7-8-16-18
- Path 3 : 1-2-3-4-5-8-2-7-8-16-17-18
- Path 4 : 1-2-3-4-5-8-2-7-8-9-10-11-12-13-14-15-8-16-17-18
- Path 5 : 1-2-3-4-5-8-2-7-8-9-10-12-13-15-8-16-17-18
- Path 6 : 1-2-3-4-5-8-2-7-8-9-10-11-12-13-15-8-16-17-18
- Path 7 : 1-2-3-4-5-8-2-7-8-9-10-12-13-14-15-8-16-17-18

8. Fungsi *Move Point*

Flowchart dan flowgraph dibawah didapat dari struktur implementasi fungsi Apply Steer pada sub bab **4.2.5.2**. Penterjemahan baris kode ke node flowchart dapat dilihat pada Gambar 5.8.



Gambar 5.8 : Flowchart dan Flowgraph Fungsi Move Point

Berikut penjelasan tiap nomor node:

1. Jika nilai absolut distance kurang atau sama dengan 0, maka menuju node 15.
2. Persiapkan variabel destRemain, distTrough dan distNow.
3. Jika distance lebih dari 0, maka menuju node 4.
4. Hitung distRemain dan distTrough.
5. Jika distRemain lebih kecil dari distNow, maka menuju node 6.
6. Hitung nilai distNow, indexPoint distRemain dan distTrough.
7. Hitung nilai t.
8. Node penghubung.
9. Jika distance lebih dari 0, maka menuju node 10.
10. Hitung distRemain dan distNow.
11. Jika distRemain< dari minus distNow, maka menuju node 12.
12. Hitung nilai distNow, indexPoint distRemain dan distTrough.
13. Hitung nilai t.

Berdasarkan flowgraph pada Gambar 5.8, jumlah node adalah 11 dan edge adalah 16, sehingga *cyclomatic complexity*nya adalah V(G)=E-N+2=16-11+2=7. Dari hasil perhitungan tersebut didapatkan 7 basis jalur independen, yaitu:

- Path 1 : 1-15
- Path 2 : 1-2-3-8-9-14-15

- Path 3 : 1-2-3-4-5-6-5-7-8-9-14-15
- Path 4 : 1-2-3-4-5-6-5-7-8-9-10-11-12-11-13-14-15
- Path 5 : 1-2-3-4-5-7-8-9-10-11-12-13-14-15
- Path 6 : 1-2-3-4-5-7-8-9-10-11-13-14-15
- Path 7 : 1-2-3-4-5-6-5-7-8-9-10-11-13-14-15

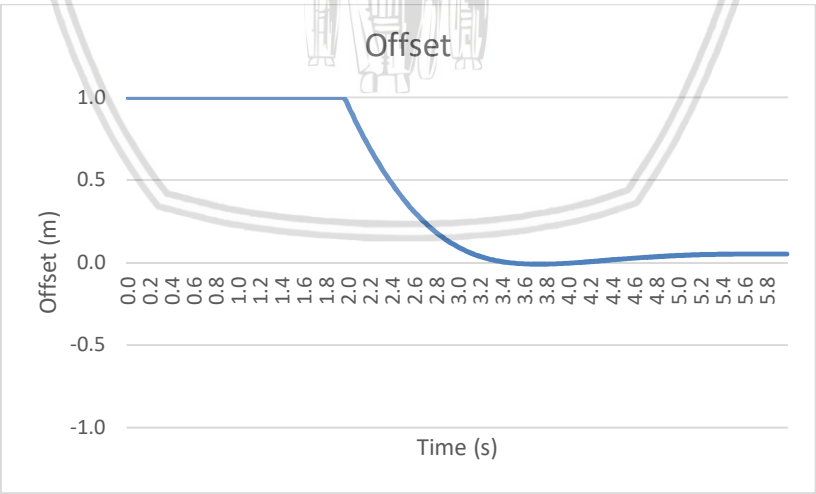
5.2.2 Hasil Pengujian Akurasi dan Presisi

Hasil pengujian akurasi dan presisi yang didapat berupa data *Offset*, *Steer* dan perbandingan *Magnitude*. Data yang terkumpul adalah hasil uji dari *Autonomous Vehicle* dengan atribut yang seperti pada Tabel 5.1.

Tabel 5.1 : Tabel Atribut *Autonomous Vehicle*

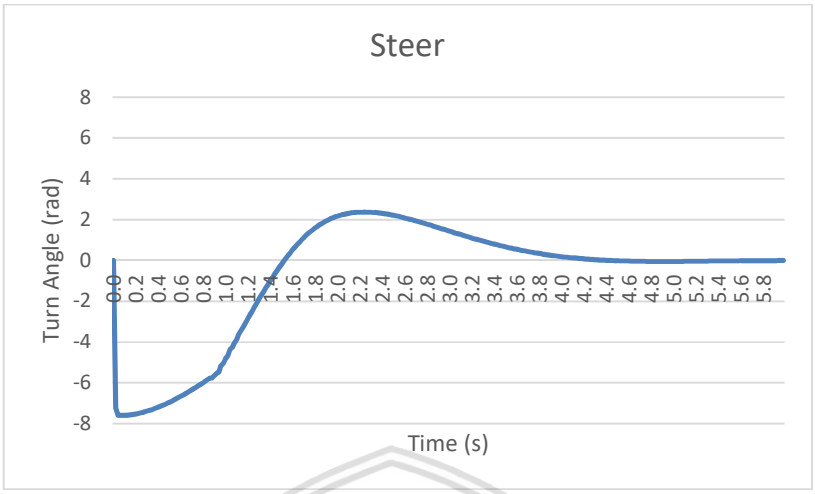
Atribut	Nilai Atribut
Accel rate	5
Decel rate	10
Taeget Distance	10
Tire Friction	0.5
Prediction Distance	2
Prediction Length	200

Data pertama yang dihasilkan adalah ketika mobil hendak berjalan. Data menunjukkan *Offset*, *Steer* dan *Magnitude*. Posisi awal mobil diposisikan dengan offset awal 1 meter dari *waypoint*. Data yang dikumpulkan tiap framenya selama 6 detik.



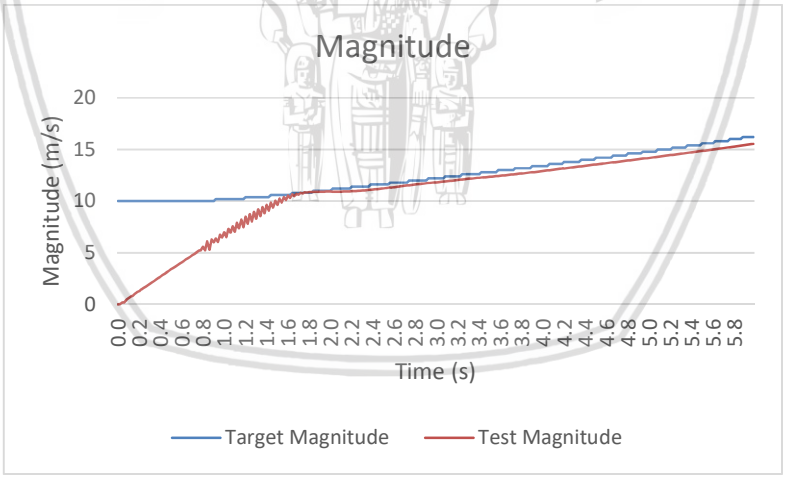
Gambar 5.9 : Grafik *Offset Autonomous Vehicle* 6 Detik Pertama

Gambar 5.9 menunjukkan grafik *offset* mobil selama 6 detik pertama. Dapat dilihat *offset* awal bernilai 1 karena posisi awal mobil di letakkan dengan *offset* 1. Kemudian seiring waktu mobil melakukan akselerasi, *offset* mulai berubah menuju 0 .



Gambar 5.10 : Grafik Steer *Autonomous Vehicle* 6 Detik Pertama

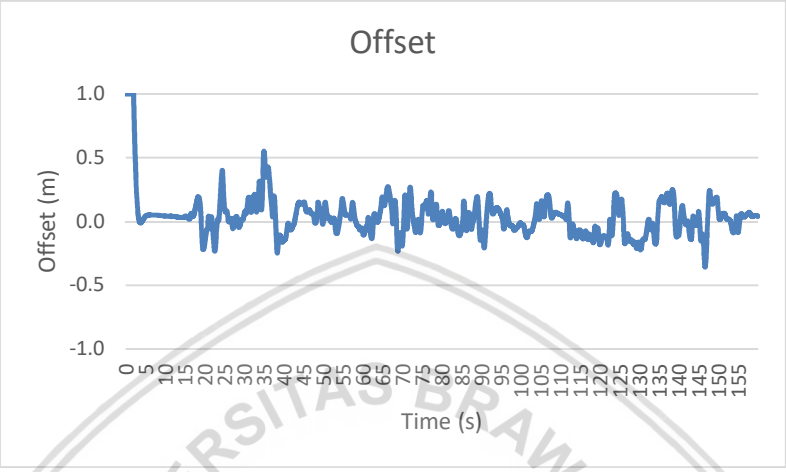
Gambar 5.10 menunjukkan grafik nilai sudut belokan *steer* atau *turn angle*. *Turn angle* diawali nilai kurang dari -7 radian yang berarti membelok kekiri maksimal. Hal ini dikarenakan posisi mobil yang berada pada offset 1 meter. Kemudian seiring dengan bertambahnya kecepatan atau *magnitude* mobil, nilai *offset* mulai mengarah ke 0 yang di ikuti perubahan nilai *turn angle* menuju ke 2 radian. Kemudian ketika nilai *offset* sudah dekat dengan 0, nilai *turn angle* juga berubah menuju 0. Sehingga pada saat detik ke 6, nilai offset dan steer sudah mendekati 0 atau dapat diartikan dengan posisi mobil sudah pada jalur *waypoint*.



Gambar 5.11 : Grafik *Magnitude Autonomous Vehicle* 6 Detik Pertama

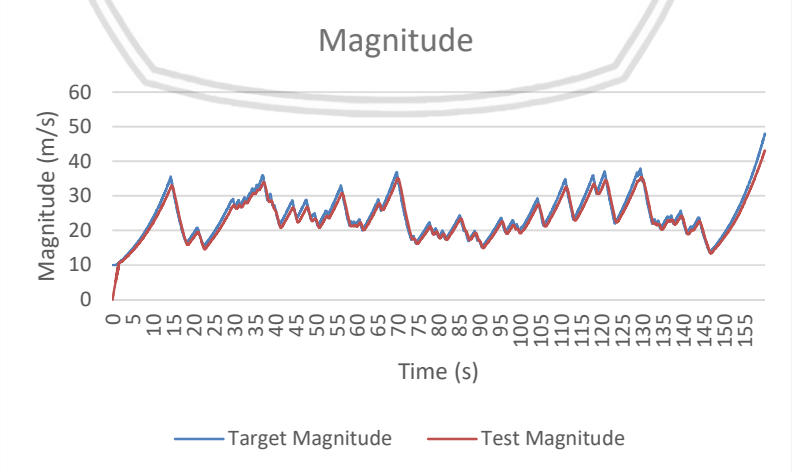
Gambar 5.11 menunjukkan Grafik *Target Magnitude* dan *Test Manitude*. Nilai awal *Target Magnitude* adalah 10 m/s sebagai acuan agar mobil melakukan percepatan hingga *magnitude* mobil mencapai nilai *Target Magnitude*. Setelah *Test Magnitude* mendekati *Target Magnitude* pada detik ke 1.6, kemudian *Target Magnitude* akan meningkatkan kecepatannya sesuai dengan perhitungan pada *magnitude calculation*. Sehingga mulai detik ke 1.6 hingga detik ke 6 nilai *Test Magnitude* sejajar dengan pertambahan nilai *Target Magnitude* hingga 16 m/s.

Data kedua yang dihasilkan adalah seluruh data yang dikumpulkan dari simulasi mobil berjalan sejak awal hingga melalui satu lap sirkuit. Data yang terkumpul dari dari pengujian ini berjumlah 8000 data yang masing-masing berisi nilai *offset*, *target magnitude*, *test magnitude* dan *time*. Data *offset* dapat dilihat dalam bentuk grafik seperti pada Gambar 5.12.



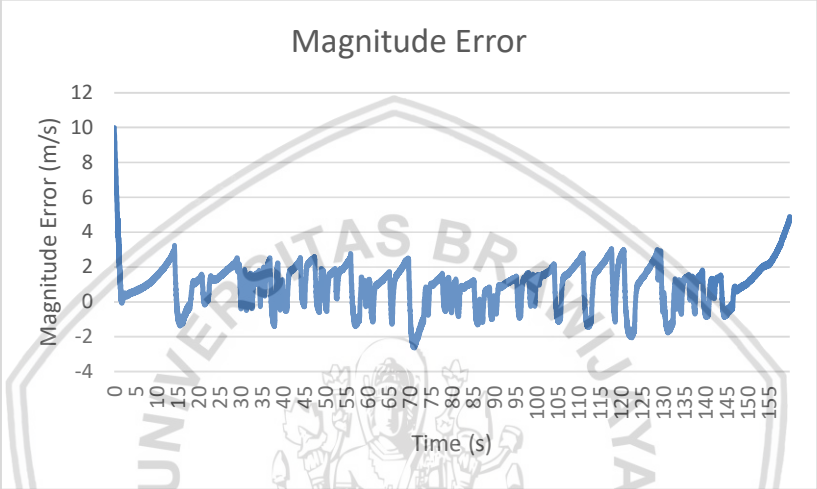
Gambar 5.12 : Grafik *Offset Autonomous Vehicle*

Data *offset* pada Gambar 5.12 adalah nilai *offset* dari posisi mobil ke titik *progress waypoint*. Semakin besar nilai *offset*, maka semakin jauh posisi mobil dari titik *progress waypoint*. Nilai rata-rata dari seluruh data *offset* adalah 0.03693 meter. Kemudian nilai standar deviasi dari seluruh data *offset* berdasarkan diferensial terhadap mean sebesar 0.162278379 meter. Sehingga jika di hitung jumlah data yang nilai absolut *offset* kurang dari standar deviasi, maka didapatkan nilai akurasi sebesar 82.8%. Sedangkan jika perhitungan standar deviasi dengan nilai mean 0, maka nilai deviasinya adalah 0.166427546 meter. Sehingga jika di hitung jumlah data yang nilai absolut offset kurang dari deviasi, maka didapatkan nilai akurasi sebesar 97.9%.



Gambar 5.13 : Grafik *Magnitude Autonomous Vehicle*

Data *Magnitude* pada Gambar 5.13 terdiri dari dua jenis, yaitu *Target Magnitude* dan *Test Magnitude*. *Target Magnitude* merupakan nilai data hasil dari perencanaan yang dilakukan pada fungsi *magnitude calculation*. Data *Target Magnitude* tersebut sudah melalui proses *smoothing* agar perubahan nilai tidak terlalu ektrim. Nilai perubahan tersebut percepatan yang dibatasi berdasarkan dua nilai input yaitu *accelRate* dan *decelRate*. Data *Test Magnitude* adalah nilai *feedback* magnitude mobil dari hasil proses kontrol PID dari *Car Controller*. Untuk melihat detil dari perbandingan kedua data tersebut, maka didapat nilai *error* yang dapat dilihat pada Gambar 5.14.



Gambar 5.14 : Grafik *Magnitude Error Autonomous Vehicle*

Rata-rata nilai seluruh *magnitude error* dari data pada Gambar 5.14 adalah 0.982264613 m/s. Nilai standar deviasi dari seluruh magnitude *error* berdasarkan diferensial terhadap nilai mean adalah sebesar 1.269508892 m/s. Sehingga jika di hitung jumlah data yang nilai absolut errornya kurang dari standar deviasi, maka nilai akurasinya adalah 51.4%. Sedangakan jika nilai perhitungan standar deviasi dengan nilai mean 0, maka nilai deviasi yang di dapat adalah sebesar 1.605146908 m/s. Sehingga jika di hitung jumlah data yang nilai absolut errornya kurang dari nilai deviasi, maka nilai akurasinya adalah sebesar 68.3%

5.2.3 Hasil Pengujian Performa

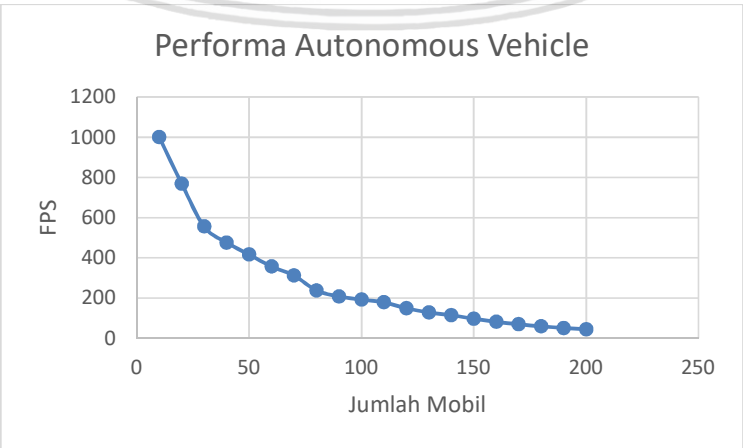
Hasil pengujian performa dilakukan dengan menjalankan program dengan jumlah mobil yang berbeda tiap kasus ujinya. Kondisi yang terapkan pada pengujian adalah tanpa menyertakan proses render. Berikut tabel hasil uji performa *Autonomous Vehicle.*

Tabel 5.2 : Hasil pengujian terhadap performa

Jumlah Mobil	Rata-rata Fps	Rata-rata tims(ms)
10	1000	1
20	769.2308	1.3

30	555.5556	1.8
40	476.1905	2.1
50	416.6667	2.4
60	357.1429	2.8
70	312.5	3.2
80	238.0952	4.2
90	208.3333	4.8
100	192.3077	5.2
110	178.5714	5.6
120	149.2537	6.7
130	128.2051	7.8
140	114.9425	8.7
150	97.08738	10.3
160	81.96721	12.2
170	69.93007	14.3
180	59.52381	16.8
190	51.28205	19.5
200	44.44444	22.5

Berdasarkan Tabel 5.1, didapatkan nilai waktu yang diperlukan untuk mengeksekusi program tiap framenya. Data yang didapat merupakan nilai rata-rata tiap framenya. Rata-rata waktu dari hasil pengujian kemudian di konversikan dalam bentuk *Frame per Second*, sehingga didapatkan nilai FPS seperti pada Tabel 5.1. Untuk melihat mempermudah pehaman data, maka hasil pengujian akan ditampilkan dalam bentuk chart seperti pada Gambar 5.12.



Gambar 5.15 : Grafik perbandingan jumlah mobil terhadap performa (FPS)

Dari data Gambar 5.12, dapat dilihat FPS menurun seiring dengan bertambahnya jumlah mobil yang diuji. Untuk keterangan lebih jelas dapat dilihat pada subbab analisis..

5.3 Analisis Hasil Pengujian

Analisis Hasil Pengujian dilakukan berdasarkan data yang telah didapat pada pengujian fungsional dan pengujian non fungsional. Berikut hasil Analisa hasil pengujian tersebut.

5.3.1 Analisis Hasil Pengujian White Box

Berdasarkan hasil pengujian White box didapat dengan melihat kesesuaian fungsi dari implementasi tiga jenis class dengan perancangan *Autonomous Vehicle* sebelumnya. Seluruh pengujian unit yang dilakukan beserta jalur independen pada setiap fungsi telah sesuai dengan perhitungan *cyclomatic complexity*. Kasus-kasus uji yang dibuat telah sesuai dengan jumlah jalur independen yang telah diuji dan masing-masing jalur independen telah dijalankan paling tidak satu kali. Oleh karena itu, dapat diambil kesimpulan bahwa unit modul dari program sudah memenuhi kebutuhan fungsional.

5.3.2 Analisis Hasil Pengujian Akurasi dan Presisi

Berdasarkan hasil uji pada pengujian akurasi dan presisi selama 6 detik eksekusi, telah dijelaskan proses mobil bergerak mendekati *waypoint* dengan *magnitude* yang telah di tentukan. Di jabarkan bahwa nilai *offset* mobil semakin kecil sering dengan berjalannya waktu dan nilai *magnitude* mobil mendekati nilai *target magnitude* di setiap waktunya. Sehinnga dalam pengujian tersebut telah terbukti bahwa mobil dapat merespon perubahan nilai *offset* dan *target magnitude*.

Langkah selanjutnya adalah menentukan nilai akurasi dan presisi berdasarkan hasil pengujian terhadap *offset*. Berdasarkan hasil uji presisi, terdapat dua macam pendekatan. Pendekatan pertama adalah ketika dihitung berdasar standar deviasi dan pendekatan kedua adalah ketika perhitungan standar deviasi *offset* menggunakan nilai mean 0. Sehingga didapatkan kesimpulan bahwa nilai presisi berdasarkan pendekatan pertama kurang lebih 0.162278379 meter dan nilai presisi pada pendekatan kedua kurang lebih 0.166427546 meter.

Untuk menghitung akurasi *offset*, nilai dari kedua pendekatan standar deviasi diatas diperlukan untuk menghitung jumlah data *offset* yang berada dalam lingkup sebaran *offset*. Berdasarkan hasil pengujian akurasi, didapatkan nilai akurasi pada pendekatan standar deviasi dan nilai akurasi pada pendekatan standar deviasi *error* dengan nilai mean 0. Sehingga dapat disimpulkan bahwa nilai akurasi berdasarkan kedua pendekatan tersebut adalah 82.8% dan 97.9%.

Pengujian *magnitude* dilakukan dengan membandingkan antara *Target Magnitude* dengan *Test Magnitude*. Karena dalam tampilan grafik data telihat berdekatan, maka perhitungan dilakukan pada nilai *error* dari kedua data. Pengujian presisi pada *magnitude error* juga menggunakan dua pendekatan.

Berdasarkan hasil pengujian, di dapatkan nilai standar deviasi pada pendekatan pertama dan kedua adalah kurang lebih 1.269508892 m/s dan 1.605146908 m/s.

Pengujian akurasi pada magnitude error didapatkan dari hasil perhitungan jumlah data *magnitude error* yang berada dalam lingkup sebaran *magnitude error*. Berdasarkan hasil pengujian akurasi, didapatkan nilai akurasi pada pendekatan standar deviasi dan nilai akurasi pada pendekatan standar deviasi *error* dengan nilai mean 0. Sehingga dapat disimpulkan bahwa nilai akurasi berdasarkan kedua pendekatan tersebut adalah 51.4% dan 68.3%.

5.3.3 Analisis Hasil Pengujian Performa

Pengujian performa dilakukan dengan mengukur kemampuan program dalam menjalankan tugasnya dalam ukuran waktu. Pengujian performa yang dilakukan pada *Autonomous Vehicle* dilakukan dengan kondisi tanpa render. Pengukuran performa dilakukan dengan mengukur waktu yang diperlukan program dalam menjalankan *Autonomous Vehicle* pada jumlah yang telah didefinisikan. Berdasarkan hasil pengujian didapatkan data waktu eksekusi dengan jumlah mobil yang berbeda. Jika dilihat dari hasil pengujian, program dapat menjalankan program dengan *Frame per Second* di atas 60 jika jumlah *Autonomous Vehicle* adalah kurang dari 180. Jika lebih dari itu akan terjadi penurunan performa yang dikarenakan perhitungan iterasi path planning membutuhkan banyak sumber daya. Sehingga dapat disimpulkan performa maksimal yang dapat diberikan *Autonomous Vehicle* adalah dengan menjalankan kurang dari 180 *Autonomous Vehicle* secara bersamaan.

BAB 6 KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan permasalahan yang sudah dirumuskan pada pendahuluan, maka didapat kesimpulan sebagai berikut:

1. Penerapan *Autonomous Vehicle* menggunakan *Steering Behavior* terdiri dari perancangan tiga kelas utama, yaitu *Car Controller*, *Steering Behavior* dan *Waypoint*. Implementasi *Autonomous Vehicle* menggunakan Unity3D dengan bahasa pemrograman c#. Kemudian pengujian menggunakan pengujian *White box*, Akurasi, Presisi dan Performa.
2. Berdasarkan hasil pengujian *White Box*, dapat disimpulkan bahwa seluruh jalur logika pada setiap fungsi telah sesuai dengan perhitungan *cyclomatic complexity* dan setiap setiap jalur independen telah di jalankan paling tidak satu kali. Sehingga hasil yang di capai telah memenuhi kebutuhan fungsional.
3. Berdasarkan hasil pengujian akurasi dan presisi terhadap data *offset*, dapat disimpulkan bahwa jika menggunakan pendekatan standar deviasi, maka nilai presisi *offset* adalah kurang lebih 0.162278379 meter dan nilai akurasi *offset* adalah sebesar 82.8%. Jika perhitungan akurasi dan presisi menggunakan perhitungan standar deviasi dengan nilai mean 0, maka nilai presisi *offset* adalah kurang lebih 0.166427546 meter dan nilai akurasi *offset* adalah sebesar 97.9%.
4. Berdasarkan hasil pengujian akurasi dan presisi terhadap data *magnitude error*, dapat disimpulkan bahwa jika menggunakan pendekatan standar deviasi, maka nilai presisi *magnitude error* adalah kurang lebih 1.269508892 m/s dan nilai akurasi *magnitude error* adalah sebesar 51.4%. Jika perhitungan akurasi dan presisi menggunakan perhitungan standar deviasi dengan nilai mean 0, maka nilai presisi *magnitude error* adalah kurang lebih 1.605146908 m/s dan nilai akurasi *magnitude error* adalah sebesar 68.3%.
5. Berdasarkan hasil pengujian performa pada *Autonomous Vehicle*, dapat disimpulkan bahwa program dapat menjalankan simulasi *Autonomous Vehicle* dengan performa terbaik jika jumlah agen yang dijalankan secara bersamaan adalah sebanyak kurang dari 180 agen.

6.2 Saran

Penerapan *Autonomous Vehicle* yang dilakukan oleh penulis masih memiliki banyak Batasan. Sehingga diperlukan pengembangan lanjut dalam beberapa poin dibawah:

1. Untuk pengembangan lebih lanjut, diperlukan pengembangan *Path Planning* agar mobil dapat bergerak maju dan mundur.
2. Pengembanga selanjutnya dengan menambahkan fitur *Path Finding* dengan *Smoothing Path*.

3. Pengembangan kecerdasan buatan untuk mambantu penentuan keputusan menggunakan metode-metode *Learning Machine*.
4. Penambahan fitur mendeteksi *Autonomous Agent* lain untuk menentukan *Dynamic Path Planning*.
5. Pengembangan lain secara bebas untuk meningkatkan fitur dan kinerja *Autonomous Vehicle*

DAFTAR PUSTAKA

Zhao , J.-S., Liu, X., Feng, Z.-J. & Dai, J. S., 2013. Design of an Ackermann-type steering mechanism. *Journal of Mechanical Engineering Science,* 227(11), pp. 2549-2562.

Burnhill, D., 2009. *Ackerman Steering Principle.* [Online] Available at: http://www.rctek.com/technical/handling/ackerman_steering_principle.html
[Diakses 8 12 2017].

Dewantoro, M. F., 2015. RANCANG BANGUN PERMAINAN SIMULASI UJIAN BERKENDARA 3D. *Repositori Jurnal Mahasiswa PTIIK UB,* 5(7).

Franklin, S. & Graesser, A., 1996. Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. *International Workshop on Agent Theories, Architectures, and Languages,* pp. 21-35.

Irawan, K. H., 2013. RANCANG BANGUN PERMAINAN SIMULASI BERKENDARA 3D PADA PLATFORM PC. *Jurnal Mahasiswa PTIIK UB,* 1(1).

Lekkas, A. M., 2014. *Guidance and path-planning systems for autonomous vehicles,* s.l.: s.n.

Pressman, R. S., 2005. *Software engineering: a practitioner's approach.* s.l.:Palgrave Macmillan.

Reynolds, C. W., 1999. Steering Behaviors For Autonomous Characters. *Game developers conference,* Volume 1999, pp. 763-782.

Setiawan, I., 2008. *KONTROL PID UNTUK PROSES INDUSTRI.* s.l.:PT Elex Media Komputindo.

Shiffman, D., 2012. *The Nature of Code: Simulating Natural Systems with Processing.* s.l.:Daniel Shiffman.

Sprunk, C., 2008. Planning Motion Trajectories for Mobile Robots Using Splines.

Unity Technologies, 2017. *Unity Documentation.* [Online] Available at: https://docs.unity3d.com/2017.2/Documentation/Manual/ [Diakses 8 Desember 2017].

Wang, H., 2005. Steering Behaviors for Autonomous Vehicles in Virtual Environments. *IEEE,* Volume 2005, pp. 155-162.