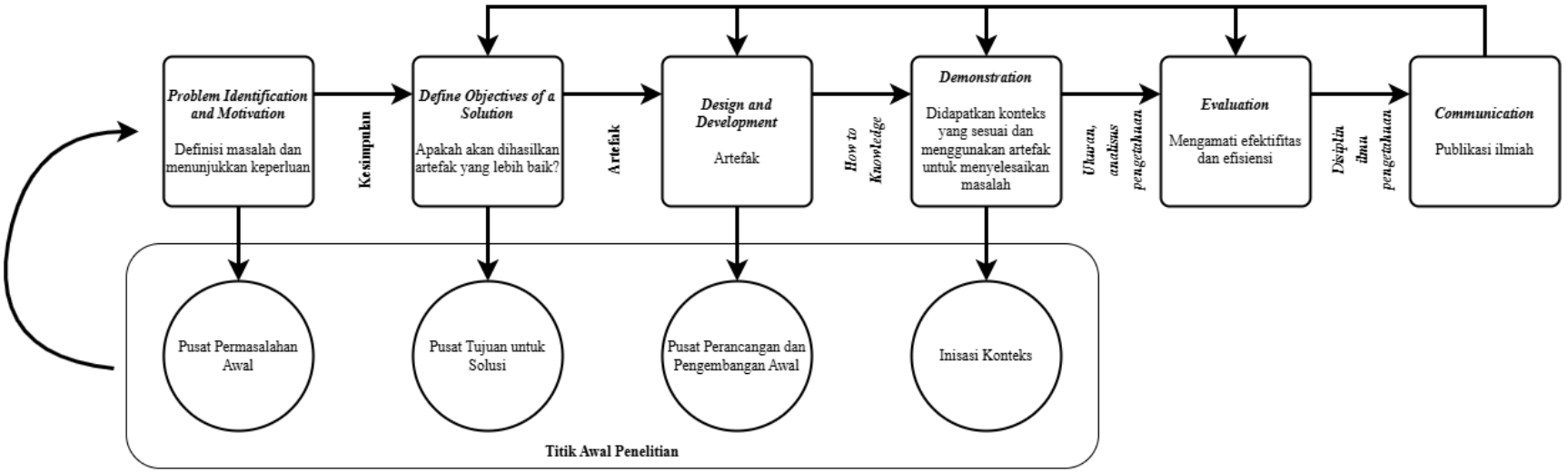
BAB III
METODE PENELITIAN

Bab ini menguraikan secara rinci mengenai metodologi yang diterapkan dalam pelaksanaan penelitian. Pembahasan mencakup tipe penelitian yang digunakan, desain penelitian yang menjadi acuan, instrumen yang diperlukan hingga prosedur penelitian yang dijalankan. Pemilihan setiap elemen metodologi ini didasarkan pada relevansinya terhadap tujuan penelitian untuk mengembangkan model *deep reinforcement learning* dengan sistem persepsi terintegrasi untuk optimalisasi perencanaan lintasan pada sistem parkir otonom.

3.1 Jenis Penelitian

Jenis penelitian yang digunakan adalah *Design Science Research Method* (DSRM). DSRM merupakan sebuah kerangka kerja metodologis yang sistematis dan terstruktur, umum digunakan untuk merancang dan mengevaluasi solusi inovatif terhadap masalah nyata (*real-world problem*), khususnya dalam ranah teknologi informasi. Metode ini menekankan pada penciptaan artefak domain teknologi informasi. Metode ini menekankan pada penciptaan artefak baru seperti perangkat lunak, algoritma, model ataupun *framework* yang dapat menyelesaikan masalah secara efektif dan berbasis bukti (*evidence-based*).

Pemilihan DSRM didasarkan pada kemampuannya untuk memandu proses penelitian mulai dari identifikasi masalah hingga komunikasi hasil solusi secara komprehensif. Metode DSRM ini telah terbukti relevan dan banyak diaplikasikan di berbagai bidang, termasuk medis, farmasi, sosial, manajemen dan teknik yang menunjukkan fleksibilitas dan ketangguhannya [28]. Dengan demikian, DSRM akan menjadi panduan utama dalam setiap tahapan pengembangan model *deep reinforcement learning* dengan sistem persepsi terintegrasi untuk optimalisasi perencanaan lintasan pada sistem parkir otonom. Adapun ilustrasi dari DSRM adalah Gambar 3.1.



Gambar 3.1 Diagram Alir DSRM

Dari Gambar 3.1, *Design Science Research Method* (DSRM) merupakan proses yang terdiri dari enam tahap berurutan untuk menciptakan dan mengevaluasi artefak solusi. Tahap ini dimulai dengan identifikasi dan motivasi masalah untuk mendefinisikan ruang lingkup dan justifikasi penelitian. Selanjutnya adalah mendefinisikan tujuan solusi yang ingin dicapai secara rasional. Tahap ketiga adalah desain dan pengembangan, di mana artefak (berupa model, metode, atau konstruksi) dibuat berdasarkan tujuan yang telah ditetapkan.

Setelah itu, dilakukan demonstrasi untuk menunjukkan kemampuan artefak dalam menyelesaikan contoh masalah. Tahap kelima adalah evaluasi, di mana kinerja artefak diukur dan dibandingkan dengan tujuan awal, seperti yang dicontohkan melalui pengujian pada simulator CARLA dengan parameter spesifik. Proses ini diakhiri dengan tahap komunikasi, yaitu penyebaran hasil penelitian—termasuk masalah, artefak, dan efektivitasnya—kepada audiens yang relevan.

3.2 Deskripsi Umum Penelitian

Pengembangan model *deep reinforcement learning* dengan sistem persepsi terintegrasi untuk optimalisasi jalur lintasan pada sistem parkir otonom adalah pengembangan terbaru dari sistem parkir otonom dengan memanfaatkan data dari kamera sebagai sistem persepsi. Data dari kamera dijadikan sebagai sistem persepsi yang diintegrasikan dengan algoritma TD3. Dengan memanfaatkan kamera sebagai sensor yang digunakan pada sistem persepsi, sistem parkir dengan model *deep reinforcement learning* ini dapat membuat proses *training* model akan lebih aman dikarenakan mengindari adanya tabrakan dengan objek di sekitar kendaraan.

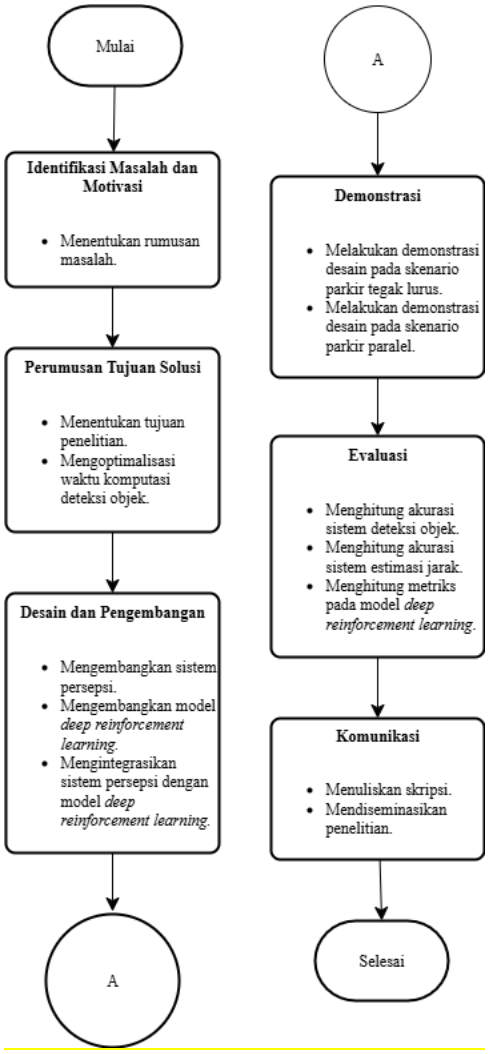
Model *Deep Reinforcement Learning* (DRL) pada sistem parkir ini sebelumnya telah dikembangkan oleh Shidqi [5]. Model DRL menggunakan algoritma TD3 sebagai algoritma DRL. Kelebihan dari TD3 adalah lebih stabil dibandingkan algoritma terdahulunya, DDPG. Namun, adapun kekurangan dari TD3 ataupun algoritma DRL lainnya sebagai bagian dari *end-to-end framework* adalah tidak adanya sistem persepsi yang digunakan pada pada pemrosesan.

Penelitian ini dilakukan dengan dua skenario, yaitu skenario parkir tegak lurus dan parkir paralel pada simulator CARLA. Komponen yang digunakan pada simulator adalah kamera. Semua data seperti data dari sensor ataupun model

dinamis seperti dinamika kendaraan (akselerasi, rem, *steering* dan suspensi), spesifikasi kendaraan (dimensi, massa, pusat massa) dan interaksi lingkungan (deteksi tabrakan, friksi). Hal ini menandakan bahwa simulasi mempertimbangkan dinamika yang realistis.

3.3 Prosedur Penelitian

Prosedur penelitian ini mengikuti tahapan-tahapan sistematis yang ditetapkan dalam kerangka *Design Science Research Method* (DSRM). Setiap tahapan dilaksanakan secara berurutan dan iteratif untuk memastikan pengembangan solusi yang *robust* dan valid. Berdasarkan diagram alir DSRM pada Gambar 3.1, adapun diagram alir DSRM pada penelitian ini adalah sebagai Gambar 3.2.



Gambar 3.2 Diagram Alir DSRM pada Penelitian

3.3.1 Identifikasi Masalah dan Motivasi (*Problem Identification and Motivation*)

Pada tahap awal ini, ditentukan dan dijelaskan masalah yang akan menjadi fokus penelitian serta hal-hal yang perlu dibenahi. Dilakukan tinjauan pustaka dari penelitian sebelumnua dan terbaru yang berkaitan dengan pengembangan model *deep reinforcement learning* yang terintegrasi dengan sistem persepsi untuk memaksimalkan optimalisasi jalur lintasan pada sistem parkir otonom.

Kegiatan identifikasi masalah ini juga mencakup pemahaman terhadap urgensi dan motivasi di balik penyelesaian masalah tersebut. Keterbatasan yang ada pada teknologi saat ini, seperti sensitivitas terhadap kondisi lingkungan yang beragam dan kebutuhan respons sistem yang cepat menjadi pendorong utama penelitian ini. Hasil dari tahap ini adalah rumusan masalah yang jelas, terukur dan justifikasi kuat mengenai kontribusi yang diharapkan dari solusi yang dikembangkan.

3.3.2 Perumusan Tujuan Solusi (*Objectives of a Solution*)

Setelah masalah teridentifikasi dengan jelas, tahap selanjutnya adalah menentukan secara rinci tujuan dari solusi yang diusulkan untuk mengatasi masalah-masalah tersebut. Potensi dari solusi yang ada diperhitungkan dan dievaluasi berdasarkan tinjauan literatur atau teknologi yang sudah dikenal dan matang.

Selain itu, waktu komputasi dari proses deteksi objek dengan menggunakan model berbasis *Convolutional Neural Network* (CNN) akan dioptimalisasi. Tujuan krusialnya adalah mengintegrasikan sistem persepsi berbasis *computer vision* secara sinergis dengan model *deep reinforcement learning* untuk perencanaan lintasan parkir otonom, sehingga menghasilkan sistem navigasi yang tidak hanya aman dan efisien, namun juga adaptif di berbagai kondisi dinamis pada mobil otonom.

3.3.3 Desain dan Pengembangan (*Design and Development*)

Pada tahap ini, peneliti merancang secara rinci dan kemudian mengembangkan artefak solusi yang diusulkan sebagai jawaban atas permasalahan yang telah dirumuskan sebelumnya. Solusi utama yang dikembangkan dalam penelitian ini adalah sebuah sistem persepsi komprehensif berbasis *computer vision* yang

terintegrasi dengan algoritma *deep reinforcement learning* untuk kapabilitas perencanaan lintasan pada mobil otonom. *Framework* pengembangan perangkat lunak seperti Pytorch dimanfaatkan secara ekstensif untuk membangun, melatih dan menguji model-model *deep reinforcement learning* yang dikembangkan.

Proses pengembangan juga melibatakan optimasi komputasi yang signifikan, terutama dengan memanfaatkan kapabilitas pemrosesan paralel dari *graphics processing unit* (GPU). Penggunaan GPU ini bertujuan untuk mempercepat proses pelatihan model *deep reinforcement learning* yang umumnya membutuhkan sumber daya komputasi besar dan waktu yang lama. Iterasi desain dan pengembangan dilakukan secara berkelanjutan berdasarkan hasil evaluasi awal untuk memastikan solusi yang dihasilkan semakin *robust* dan efektif.

3.3.4 Demonstrasi (*Demonstration*)

Setelah artefak solusi berhasil dirancang dan dikembangkan, tahap selanjutnya adalah mendemonstrasikan kapabilitas solusi tersebut dalam menyelesaikan masalah yang telah diidentifikasi. Demonstrasi yang dilakukan melibatkan dua skenario utama, yaitu skenario parkir tegak lurus dan parkir paralel yang disimulasikan pada beberapa tempat menggunakan perangkat lunak simulasi CARLA. Pada skenario simulasi perangkat lunak model *deep reinforcement learning* yang telah dibangun dilatih secara intensif menggunakan *framework* Pytorch dengan dukungan akselerasi dari GPU untuk mempercepat proses pelatihan dan iterasi.

Tujuan dari demonstrasi ini adalah untuk menunjukkan fungsionalitas inti dari sistem yang dikembangkan, termasuk kemampuan deteksi objek, pemahaman lingkungan dan pembuatan keputusan navigasi. Simulasi menggunakan CARLA memungkinkan pengujian dalam berbagai kondisi lingkungan dan skenario lalu lintas yang terkontrol dan aman.

3.3.5 Evaluasi (*Evaluation*)

Pada tahap evaluasi, kinerja dari solusi berupa sistem diukur secara kuantitatif dan kualitatif berdasarkan tujuan-tujuan yang telah ditetapkan pada tahap awal. Proses evaluasi difokuskan pada beberapa metrik kunci, terutama yang berkaitan dengan keakuratan sistem deteksi objek, seperti *mean Average Precision*

dan aspek krusial lainnya yang dievaluasi adalah kecepatan respons sistem secara keseluruhan, mulai dari akuisisi data sensor hingga *output* berupa aksi kontrol.

Pengukuran metrik-metrik tersebut dilakukan melalui serangkaian skenario pengujian yang terstruktur dalam lingkungan simulasi. Hasil evaluasi ini dibandingkan dengan *baseline* atau solusi yang sudah ada untuk menunjukkan keunggulan dan inovasi dari sistem yang diusulkan. Temuan dari tahap evaluasi menjadi dasar untuk penyempurnaan lebih lanjut atau validasi akhir terhadap efektivitas solusi.

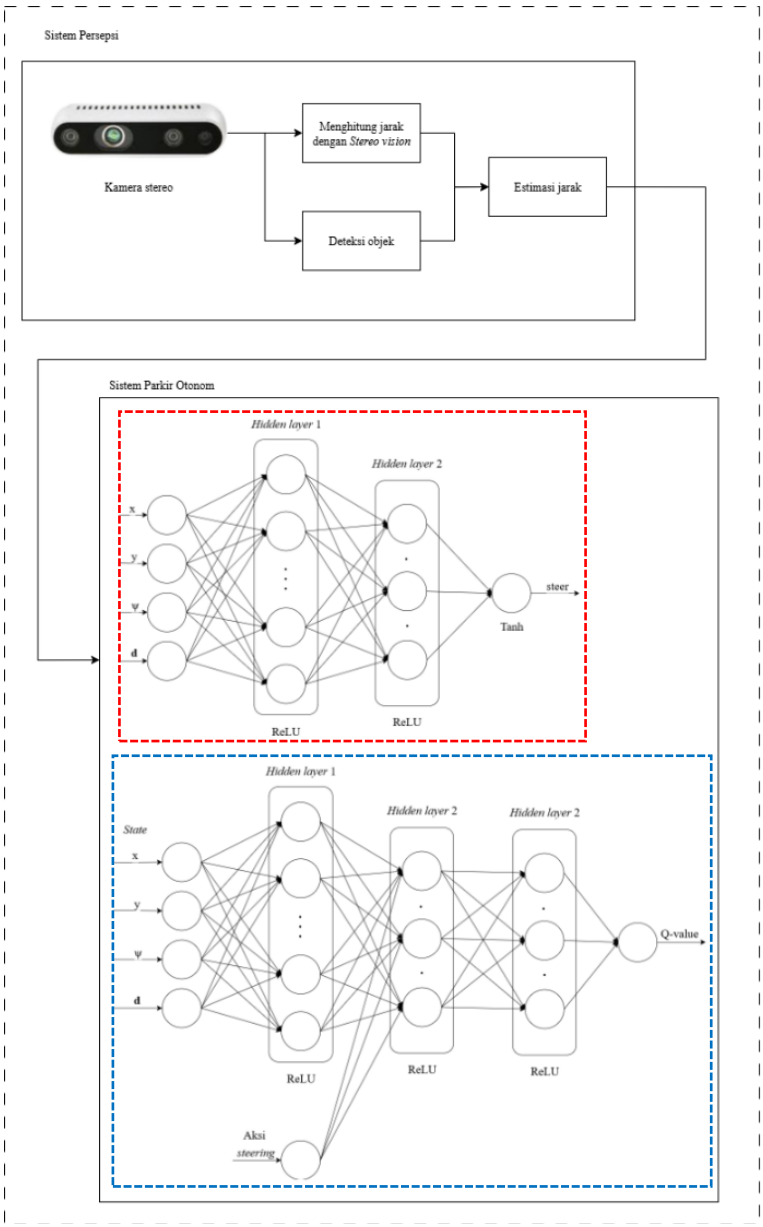
3.3.6 Komunikasi (*Communication*)

Tahap komunikasi merupakan fase terakhir dalam siklus DSRM, dimana hasil-hasil penelitian yang telah diperoleh akan disampaikan dan didiseminasikan kepada audiens yang relevan. Temuan penelitian ini akan dikomunikasikan secara komprehensif, baik dari sisi metodologi, proses pengembangan maupun hasil evaluasi yang dihasilkan. Bentuk utama komunikasi hasil riset ini adalah melalui skripsi.

Publikasi ini akan disusun mengikuti standar kualitas dan etika publikasi ilimiah yang ditetapkan, termasuk Pedoman Penulisan Karya Ilmiah Universitas Pendidikan Indonesia Edisi 2024. Selain menyelesaikan kepenulisan skripsi, hasil penelitian ini akan diseminasi melalui presentasi pada konferensi ilmiah internasional yang relevan dengan bidang kajian. Tujuan dari tahap komunikasi ini adalah untuk memberikan kontribusi terhadap khazanah ilmu pengetahuan serta memfasilitasi adopsi dan pengembangan lebih lanjut dari solusi yang ditawarkan.

3.4 Arsitektur Sistem

Arsitektur sistem pada Gambar 3.3 menunjukkan integrasi antara sistem persepsi dengan model *deep reinforcement learning*. Bagian yang pertama adalah sistem estimasi jarak berbasis *stereo vision* yang menggunakan dua kamera sebagai sensor. Sistem ini menghasilkan estimasi jarak antara mobil dengan objek di sekitarnya. Nilai estimasi jarak ini akan digunakan sebagai salah satu *state space* pada model DRL.



Gambar 3.3 Arsitektur Sistem Parkir Otonom

Bagian kedua adalah sistem DRL, sistem ini adalah implementasi dari model TD3 yang telah dikembangkan. Nilai estimasi jarak dari sistem estimasi jarak dijadikan salah satu *state space* yang digunakan pada model. *Output* dari sistem ini adalah *Q-value* dan juga lintasan jalur parkir. *Q-value* adalah nilai *reward* yang didapatkan oleh agen setelah melakukan *training*.

Pada Gambar 3.3, terdapat dua blok pada sistem parkir otonom. Blok berwarna hijau adalah *actor network* dan blok berwarna biru adalah *critic network*. Seperti

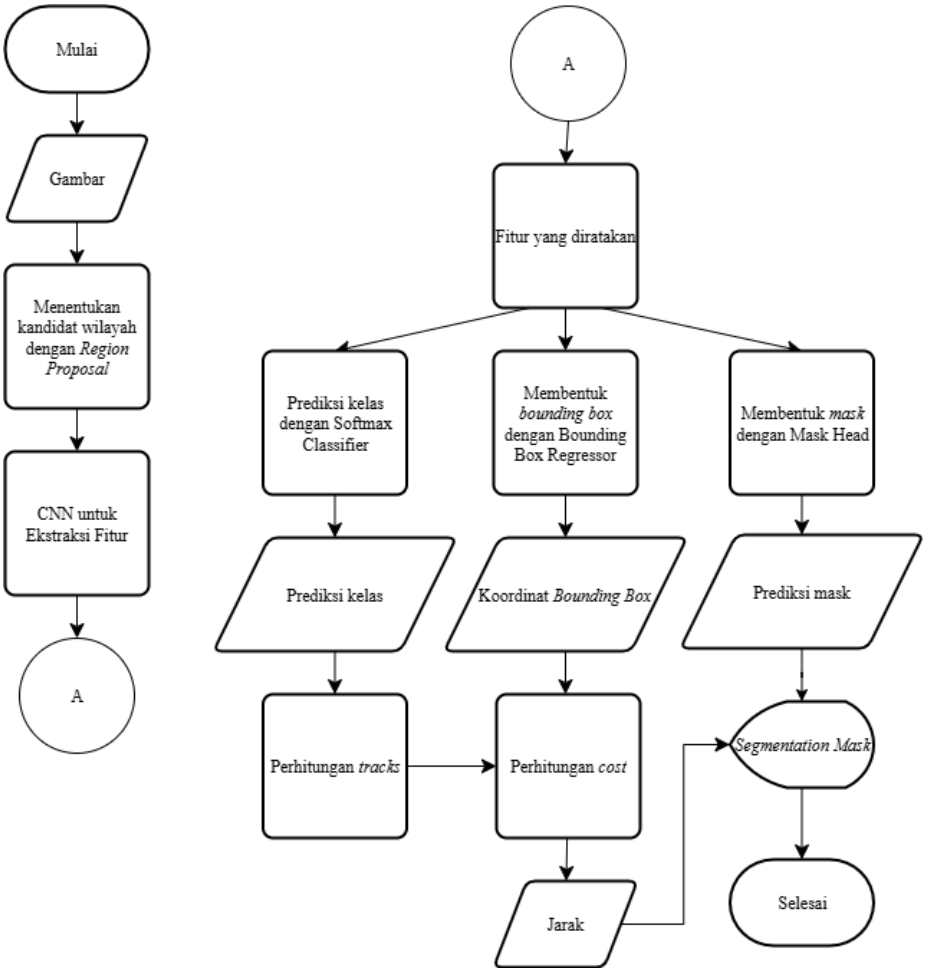
yang telah disebutkan pada sub sub bab 2.4.4 mengenai metode *actor critic*, peran *actor* adalah memilih aksi yang sesuai dengan *policy*, sedangkan *critic* mengevaluasi tindakan tersebut dengan memperkirakan *value function* [21]. Pada *actor-critic*, agen adalah kombinasi dari *actor* dan *crtitic*. Hal ini dapat dianalogikan dengan agen sebagai manusia, a*ctor* menjadi otot fan *critic* menjadi otaknya.

3.5 Perancangan Sistem Persepsi

Sistem persepsi adalah sistem yang bertanggung jawab untuk melihat dan memahami keadaan lingkungan di sekitar mobil. Sistem persepsi pada penelitian ini bertujuan untuk mengetahui jarak antara kendaraan dengan objek di sekitarnya. Ada dua sistem yang diteliti oleh peneliti pada penelitian ini, yaitu sistem estimasi berbasis Koordinat Bounding Box dan juga sistem estimasi berbasis peta disparitas.

3.5.1 Sistem Estimasi Jarak Berbasis Koordinat *Bounding Box*

Sistem estimasi jarak yang pertama didasarkan oleh penelitian yang dilakukan oleh Zaremba [29]. Sistem estimasi jarak ini menggunakan koordinat *bounding box* untuk perhitungan jarak dan MaskRCNN sebagai algoritma deteksi objek [29]. *Flowchart* dari sistem estimasi jarak ini dapat kita lihat pada Gambar 3.4.



Gambar 3.4 *Flowchart S*istem Estimasi Jarak Pertama

Untuk mencari nilai dari jarak, hal pertama yang dilakukan adalah mencari nilai parameter model seperti $tan\theta$ dan f_l. Langkah pertama dari mencari nilai parameter model adalah dengan mendeteksi gambar dari kedua kamera dengan menggunakan MaskRCNN. MaskRCNN adalah sebuah *framework* PyTorch yang dapat digunakan untuk deteksi objek, segmentasi objek dan juga pelabelan. MaskRCNN dipilih karena ia dapat melakukan *instance segmentation* yang meningaktkan keakuratan dari deteksi objek [30].

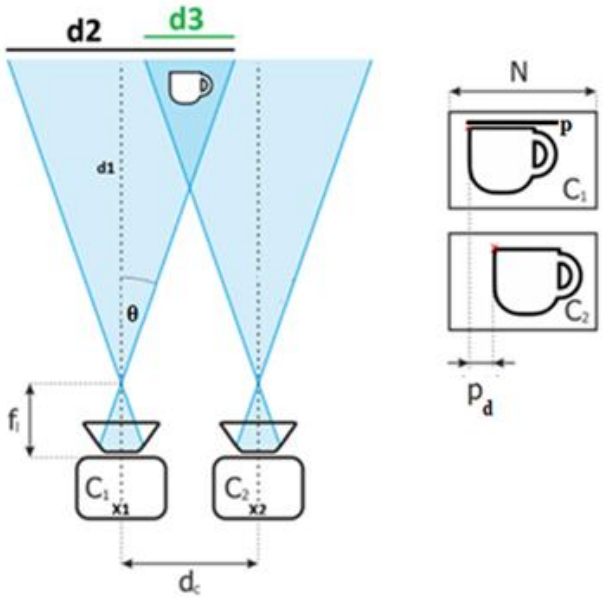
Hasil dari deteksi objek yang dilakukan oleh MaskRCNN adalah *bounding box* yang memiliki nilai koordinat pixel dari sudut-sudutnya seperti *top left* x (tlx), *top left* y (tly), *bottom right* x (brx), dan *bottom right* y (bry). Dari Gambar 3.5 dapat diketahui bahwa tlx adalah titik X11, tly adalah Y1, brx adalah X2, dan bry adalah Y2.

Gambar 3.5 Koordinat *Bounding Box* Hasil dari *MaskRCNN*

Lalu, nilai koordinat pixel digunakan untuk mencari *center*, *top left*, *top right* dan *area* dari *bounding box*. Hasil dari perhitungan ini digunakan untuk mencari nilai *horizontal distance* antara *top left* dan *top right*, *vertical distance* dan *area different* dari *bounding box* yang berasal dari gambar pada kamera kanan dan kamera kiri. Setelah mendapatkan nilai *top left* dan *bottom right*, kedua nilai tersebut digunakan untuk mencari nilai *distance to center from top left* dan *distance to center to bottom right*.

Posisi kamera dirancang sejajar antara kamera kanan dan kiri, sehingga dibuatlah fungsi *cost* sebagai "nilai hukuman" apabila ada pergerakan vertikal dari kedua *bounding box*, perbedaan *area* kedua *bounding box*, pergerakan horizontal *bounding box* dan perbedaan label pada array label. Fungsi *cost* juga digunakan untuk mencocokan gambar dengan metode *linear sum assignment*.

Lalu yang selanjutnya adalah mencari nilai *pixel distance* (p_d) dengan membandingkan nilai nilai *distance to center from top left* (tl to center) dari *bounding box* kamera kanan dan *distance to center from bottom right* (br to center) dari *bounding box* kamera kanan. Apabila nilai dari tl to center lebih kecil dari br to center, maka nilai p_d adalah nilai *horizontal distance top left* (dist_tl). Namun, jika tl to center lebih besar dari br to center, maka nilai p_d adalah nilai *horizontal distance bottom right* (dist_br)



Gambar 3.6 Posisi Kamera Stereo pada Pendekatan Pertama

Adapun persamaan geometri pada salah satu kamera (C_1) dari Gambar 3.6 adalah sebagai berikut.

$$distance = f_l + d_1 \tag{3.1}$$

$$\frac{x_1}{f_l} = \frac{d_2}{d_1} \rightarrow d_1 = \frac{f_l d_2}{x_1} \tag{3.2}$$

$$tan\theta = \frac{d_2/2}{d_1} \rightarrow d_1 = \frac{d_2}{2tan\theta} \tag{3.3}$$

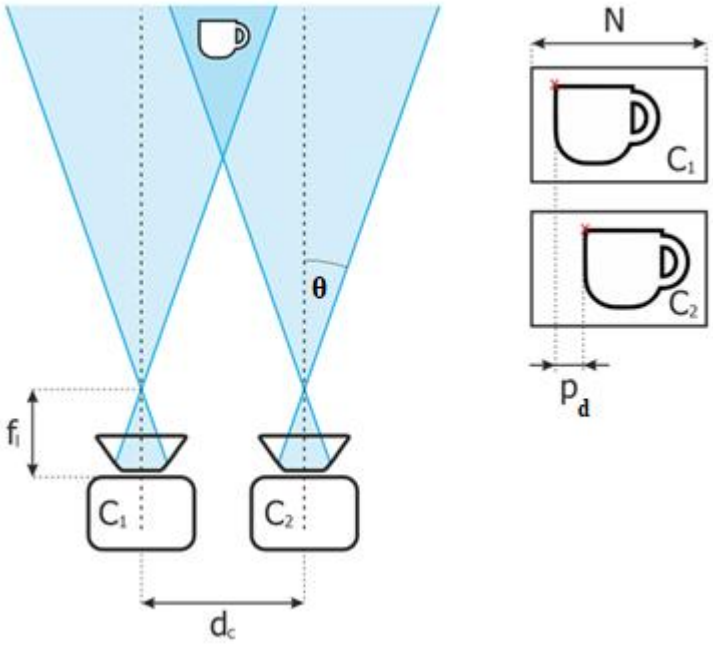
$$distance = \frac{d_2}{2tan\theta} + f_l \tag{3.4}$$

Dari Gambar 3.6 kita dapat mengetahui bahwa p adalah jumlah piksel dan N adalah ukuran lebar piksel gambar dan dapat membuat persamaan geometri sebagai berikut.

$$\frac{d_3}{d_2} = \frac{p}{N} \rightarrow d_2 = \frac{N d_3}{p} \tag{3.5}$$

$$distance = \frac{N d_3}{2ptan\theta} + f_l \tag{3.6}$$

Langkah selanjutnya adalah dengan menambahkan kamera, yaitu C_2. Dari Gambar 3.6, d_c adalah jarak antar kamera dan d_3 adalah lebar objek (piksel).



Gambar 3.7 Posisi Kamera

Dari Gambar 3.7, cara termudah untuk menghitung jarak adalah dengan menganggap jarak dari kamera sebanding dengan $\frac{N}{p_d}$. Maka daripada itu, $distance = a \times \left(\frac{N}{p_d}\right) + b$. Kita dapat melakukan kalibrasi dimana kita mengetahui *distance* dan menghitung nilai a dan b yang mana kedua nilai ini adalah konstanta kamera. Ketika melakukan deteksi, kita dapat nilai p_d (*pixel distance*) dan nilai N yang dapat digunakan untuk menghitung jarak.

$$distance = \frac{Nd_c}{2p_d tan\theta} + f_l \tag{3.7}$$

$$a_1 = \frac{Nd_c}{2p_{d_1} tan\theta} + f_l \rightarrow f_l = a_1 - \frac{Nd_c}{2p_{d_1} tan\theta} \tag{3.8}$$

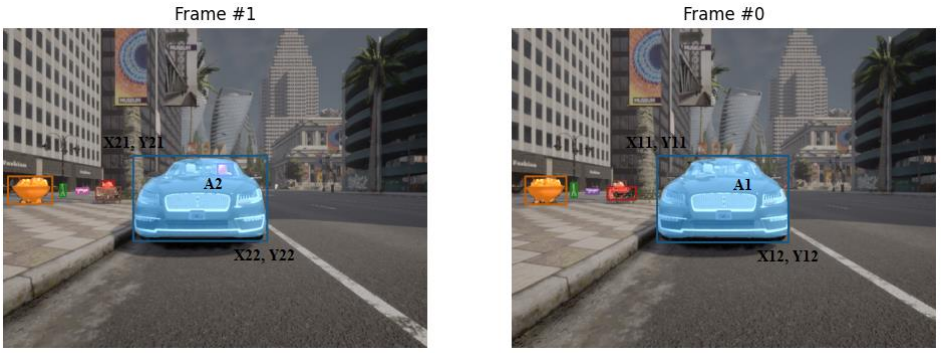
$$a_2 = \frac{Nd_c}{2p_{d_2} tan\theta} + f_l \rightarrow f_l = a_2 - \frac{Nd_c}{2p_{d_2} tan\theta} \tag{3.9}$$

Dari Gambar 3.6 dan Gambar 3.7, kita asumsikan $\frac{d_3}{p} = \frac{d_c}{p_d}$. a_1 adalah jarak yang diketahui dari percobaan 1, a_2 adalah jarak yang diketahui dari percobaan 2, p_{d_1} adalah *pixel distance* dari percobaan 1 dan p_{d_2} adalah *pixel distance* dari percobaan 2. Dari Persamaan 3.8, nilai yang telah diketahui adalah a_1, a_2, N dan

d_c. Sedangkan untuk p_{d_1}, p_{d_2}, $tan\theta$ dan f_l belum diketahui. Berikut adalah nilai parameter a_1 dan a_2 yang digunakan untuk mancari p_{d_1}, p_{d_2}, $tan\theta$ dan f_l yang dapat kita lihat pada Tabel 3.1.

Tabel 3.1 Nilai Parameter pada Sistem Estimasi Jarak Berbasis Koorinat *Bounding Box*

a_1	a_2	p_{d_1}	p_{d_2}	$tan\theta$	f_l
50	60	72,52	95,98	-2,16	90,91
80	100	71,28	53,31	1,51	20,67
120	140	47,44	41,86	0,90	-30,04
160	180	38,95	32,29	1,69	63,03
200	220	29,88	29,45	0,16	-1169,77
240	260	27,06	22,54	2,37	140,27
280	300	20,84	21,59	-0,53	855,73
300	320	18,71	17,14	1,57	101,66



Gambar 3.8 Titik-titik Koordinat *Bounding Box* dari Hasil Deteksi MaskRCNN

Dari Gambar 3.8, kita dapat membuat persamaan sebagai berikut.

$$distance = \frac{d_c}{2} \times N \times \frac{1/_{tan\theta}}{p_d} + f_l \tag{3.10}$$

$$f_l = \frac{(a_2 p_{d_2}) - (a_1 p_{d_1})}{(p_{d_2} - p_{d_1})} \tag{3.11}$$

$$tan\theta = \frac{1}{a_2 - f_l} \times \frac{d_c}{2} \times \frac{N}{p_{d_2}} \tag{3.12}$$

$$pixel\ distance\ (p_d) = tl\ to\ center\ \vee\ br\ to\ center \tag{3.13}$$

$$dist\_tl = tl[0] - tl[1] \tag{3.14}$$

$$dist\_br = br[0] - br[1] \tag{3.15}$$

Adapun keterangan dari Persamaan 3.13 sampai dengan 3.15 adalah sebagai berikut.

tl to center = Jarak kiri atas ke tengah gambar (X11 ke A1) [px]

br to center = Jarak kanan bawah ke tengah (X12 ke A1) [px]

dist_tl = Jarak antara kiri atas gambar kanan (X11) dengan kiri atas gambar kiri (X21) [px]

dist_br = Jarak antara kanan bawah gambar kanan (X12) dengan kanan bawah gambar kiri (X22) [px]

dist_tl = X11 – X21 [px]

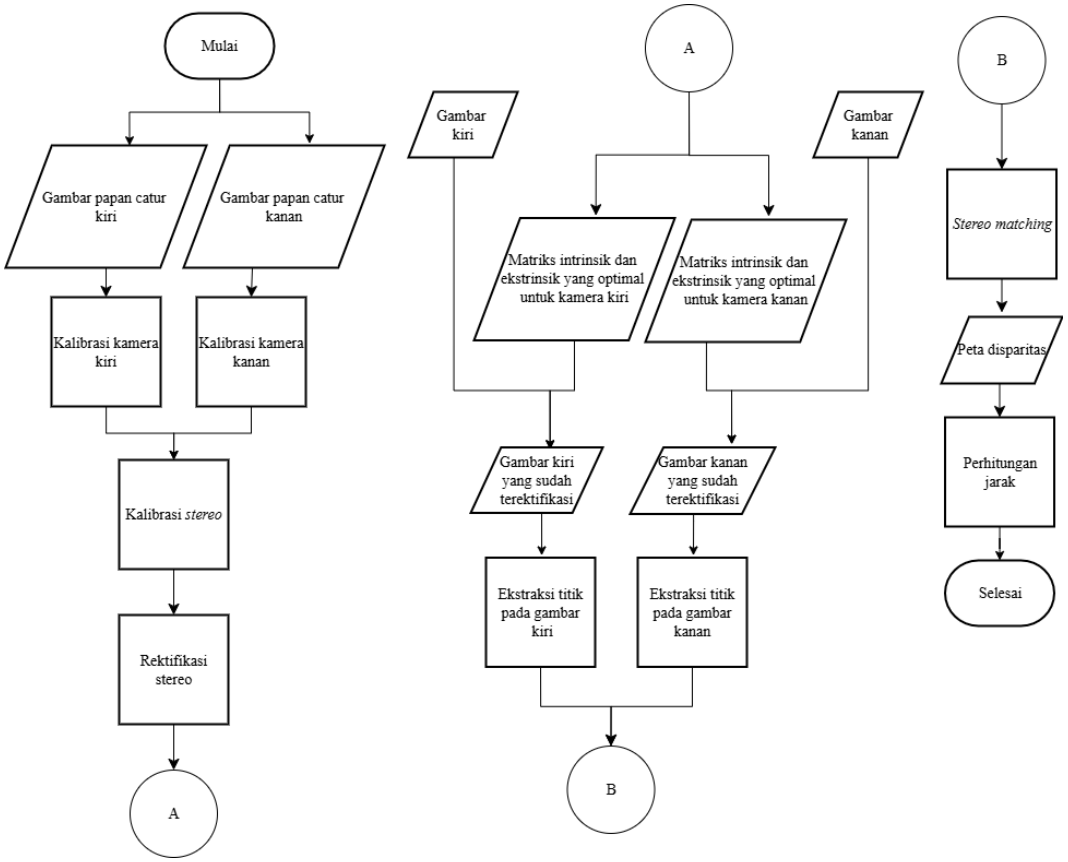
dist_br = X12 – X22 [px]

Adapun persamaan dari *pixel distance* adalah sebagai Persamaan 3.16.

$$pixel\_distance = \begin{cases} dist\_tl & \text{tl\_to\_center} < \text{br\_to\_center} \\ dist\_br & \text{tl\_to\_center} > \text{br\_to\_center} \end{cases} \quad (3.16)$$

3.5.2 Sistem Estimasi Jarak Berbasis Peta Disparitas

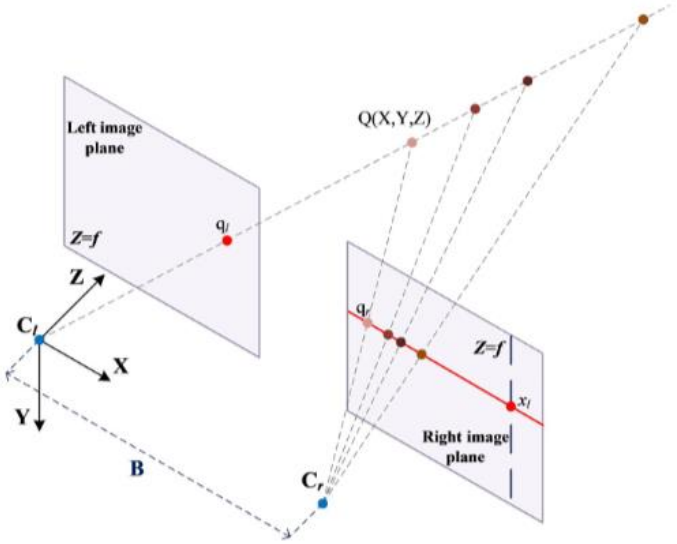
Sistem estimasi jarak kedua didasarkan dari penelitian yang dilakukan oleh Adil pada tahun 2022. Sistem estimasi jarak ini menggunakan pendekatan peta disparitas pada perhitungan jarak dan YOLOv8 sebagai algoritma deteksi objek. YOLOv8 dipilih dikarenakan keandalannya dalam deteksi *real*-time pada berbagai macam kondisi mengemudi [31]. Sistem ini terdari empat tahapan, kalibrasi stereo, rektifikasi stereo, pencocokan stereo (*stereo matching*) dan perhitungan jarak. *Flowchart* dari sistem estimasi jarak ini dilihat pada Gambar 3.9.



Gambar 3.9 Sistem Estimasi Jarak dengan Pendekatan Peta Disparitas

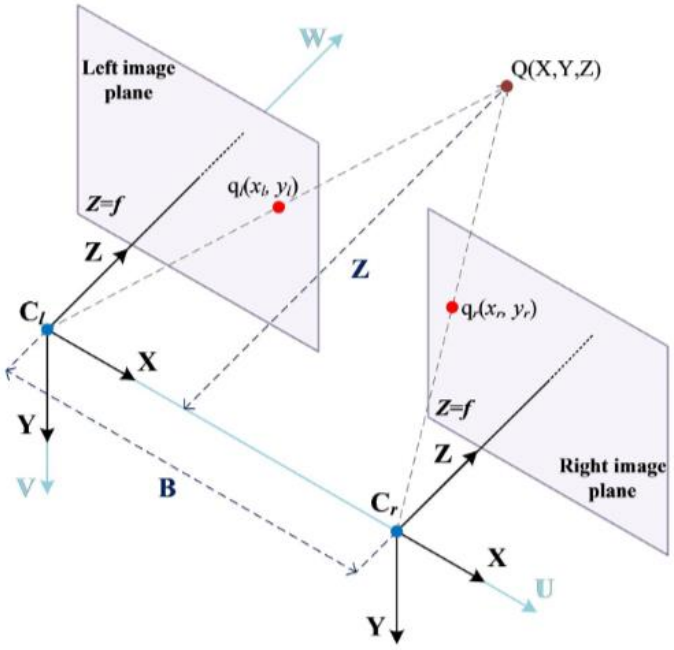
Tahap pertama dari *stereo vision* adalah kalibrasi stereo. Kalibrasi stereo dilakukan pada kamera untuk mendapatkan nilai intrinsik dan ekstrinsik dari masing-masing kamera [32]. Masing-masing kamera dikalibrasi dengan mencari posisi dari sudut-sudut papan catur dari 35 gambar. Kalibrasi kamera ini diperlukan untuk menghasilkan parameter untuk rektifikasi stereo.

Tahap kedua dari *stereo vision* adalah rektifikasi stereo. Rektifikasi stereo adalah proses mencocokkan kedua gambar dari dua sisi diproyeksikan dalam satu bidang gambar. Gambar yang telah diproyeksikan dilakukan penyelarasan garis epipolar sepanjang sumbu horizontal. Rektifikasi stereo menyederhanakan proses pencocokan stereo secara signifikan.



Gambar 3.10 Geometri Epipolar: *Parallel Stereo* [32]

Seperti pada gambar Gambar 3.11, semua garis epipolar berada pada garis horizontal gambarnya, sehingga proses rektifikasi membantu proses pencocokan stereo menjadi lebih mudah. Alih-alih mencari titik-titik yang sesuai di seluruh bingkai gambar kanan, kita dapat membatasi pencarian hanya pada satu baris horizontal [33].



Gambar 3.11 Triangulasi Stereo [32]

Gambar yang didapatkan dari proses rektifikasi stereo digunakan untuk membuat peta disparitas. Pada gambar Gambar 3.11, jarak antara titik pusat kamera kiri dengan pusat kamera kanan adalah B (*baseline*). $q_l(x_l, y_l)$ dan $q_r(x_r, y_r)$ adalah proyeksi Q pada bidang gambar kanan dan kiri. Z adalah jarak dari titik Q ke kamera. Setelah proses rektifikasin stereo selesai, kita mendapatkan koordinat y pada kedua gambar $y = y_l = y_r$. Dengan menggunakan segitiga yang sama, kita dapat mendapatkan Persamaan 3.17 sebagai berikut:

$$\frac{B}{Z} = \frac{B-(x_l - x_r)}{Z-f} \rightarrow \frac{B}{Z} = \frac{B-d}{Z-f} \rightarrow d = \frac{Bf}{Z} \tag{3.17}$$

Tahap ketiga dari *stereo vision* adalah pencocokan stereo. Pencocokan stereo adalah teknik *computer vision* untuk menentukan informasi *depth* dari dua atau lebih gambar dengan objek yang sama namun dengan perbedaan sudut pandang, sama seperti bagaimana manusia memperoses penglihatan dari kedua bola mata. Informasi *depth* yang diterima yaitu disparitas (perpindahan horizontal) antara titik di gambar kiri dan kanan. Algoritma yang biasa digunakan untuk pencocokan stereo adalah *semi-global matching* [34]. Hasil dari pencocokan stereo ini menghasilkan peta dispartitas yang bisa kita gunakan pada tahap perhitungan jarak. Adapun parameter dari pencocokan stereo yang digunakan adalah sebagi Tabel 3.2.

Tabel 3.2 Nilai Parameter SGBM

Parameter	**Nilai**
minDisparity	3
numDisparity	127
blockSize	3
uniquenessRatio	10
speckleWindowsSize	100
speckleRange	32
Disp12MaxDiff	5
P1	2304
P2	36864

Lalu tahap keempat sebagai langkah terakhir pada *stereo vision* adalah perhitungan jarak. Perhitungan jarak dilakukan dengan cara mengubah nilai disparitas menjadi jarak dengan cara mencari nilai disparitas pada saat jarak yang diketahui [35]. Lalu, data tersebut dihitung nilainya dengan menggunakan persamaan polinomial orde 6 dengan Persamaan 3.18.

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ ... + \beta^{\varepsilon} X^e + \varepsilon \quad (3.18)$$

Dari Persamaan 3.17 di atas, dapat diketahui bahwa persamaan di atas memiliki keterangan pada Tabel 3.3

Tabel 3.3 Keterangan dari Simbol pada Persamaan Polinomial Orde 6

No	Simbol	Keterangan
1	Y	Variabel prediksi
2	X	Variabel prediktor
3	β_0	Konstanta (intersep)
4	$\beta_1, \beta_2, \dots, \beta_e$	Koefisien regresi untuk masing-masing derajat polinomial ($X, X^2, \dots, X^e$)
5	ε	Error
6	e	Derajat polinomial

3.6 Perancangan Model *Deep Reinforcement Learning*

3.6.1 Pengaturan Lingkungan Simulasi

Perancangan model dilakukan di simulator CARLA versi 0.9.15 dan di tempat bernama "Town10HD". Simulator CARLA adalah perangkat lunak *open-source* yang dikembangkan untuk mendukung *training*, *prototyping* dan validasi dari model kendaraan otonom, termasuk persepsi dan juga kontrol dari kendaraan [36]. Skenario parkir yang dilakukan ada dua, yaitu tegak lurus dan paralel. Adapun tampilan dari tempat parkir pada kedua skenario adalah Gambar 3.12.

Gambar 3.12 Parkir tegak lurus (kiri) dan parkir paralel (kanan)

Gambar 3.12 adalah tampilan menyeluruh dari lingkungan tempat parkir dijalankan. Titik hijau adalah titik koordinat awal mobil, (-63.70, 178.20) pada skenario parkir tegak lurus dan (-13.30, 130.20) pada skenario parkir paralel. Sedangkan titik oranye adalah titik koordinat akhir mobil, (-68.0, 172.80) pada skenario parkir tegak lurus dan (0.50, -71.0) pada skenario parkir paralel. Lalu, kotak hijau adalah tempat parkir yang dituju. Mobil akan dianggap terparkir apabila berada di dalam kotak ini.

Mobil Tesla Model 3 dipilih dikarenakan persepsi yang digunakan oleh Tesla Model 3 hanya menggunakan kamera yang mana cocok untuk dijadikan *prototype* pada penelitian ini. Selain itu, mobil Tesla Model 3 adalah mobil listrik yang mana memiliki kelebihan torsi yang instan sehingga memudahkan pada saat kontrol kecepatan. Dan Tesla Model 3 ini telah tersedia pada *blueprint library* CARLA. Sensor tabrakan pun dipasang untuk mendeteksi tabrakan dengan kendaraan atau objek lainnya. Adapun spesifikasi kendaraan dari Tesla Model 3 dapat dilihat pada Tabel 3.4.

Tabel 3.4 Spesifikasi Tesla Model 3

Spesifikasi	Dimensi	Satuan	Sumber
Panjang	4695	mm	Buku Panduan Tesla Model 3 [37]
Tinggi	2088	mm	

Sudut maksimal *steering*	70	°	CARLA *Library* [38]
Massa	1845	kg	
Pusat massa	(0.45, 0, -0.3)	m	
Koefisien gesek	0.15	-	
Friksi ban	3.5	-	
Tingkat da*mping* ban	0.25	Ns/m	

Dari Tabel 3.4, kendaraan Tesla Model 3 yang disediakan oleh CARLA adalah model dinamis dikarenakan menyediakan beberapa parameter penting dalam model dinamis seperti massa, koefisien gesek, friksi ban, tingkat *damping* ban dan lokasi pusat massa. Hal ini menunjukkan bahwa kendaraan mempertimbangkan inersia, aerodinamika dan traksi ban yang membuktikan bahwa simulasi ini mempertimbangkan dinamika kendaraan yang realistis.

3.6.2 *State Space* dan *Action Space*

State space adalah semua keadaan yang dapat diamati oleh agen di dalam lingkungan dan juga *input* untuk agen. Lalu *action space* adalah semua tindakan yang mungkin dilakukan oleh agen di lingkungan dan juga *output* dari agen. Penentuan *state space* dan *action space* dirancang untuk membuat agen mempelajari skenario parkir secara efisien dan efektik. Pada Tabel 3.5 menunjukkan *state space* dan *action space* pada penelitian ini.

Tabel 3.5 *State space* dan *action space*

State Space	**Rentang**	**Satuan**	***Action Space***	**Rentang**	**Satuan**
x	[-69.00, -61.00][a]	m	*Steering*	[-0.65, 0.65]	-
y	[172.00, 180.00][b]	m	*Throttle*	0.3	-
ψ	[-180, 180]	°	*Reverse*	*True*	-

[a][-16.00, -3.30] untuk parkir paralel

[b][126.40, 131.80] untuk parkir paralel

Pergerakan yang agen lakukan dibatasi pada x dan y pada Tabel 3.5. Sudut dari kepala menentukan orientasi agen terhadap lingkungan. Aksi agen seperti *steering*, *throttle* dan *reverse* juga dibatasi. *Input* dari *steering* pada CARLA dibatasi di rentang [-1, +1] yang bernilai [-70°, +70°] pada rentang *steering*. Pada penelitian ini, *input steering* dibatasi pada rentang [-0.65, +0.65] yang bernilai [-45°, +45°]. Selain itu, nilai *throttle* konstan pada nilai 0.3, yang mana menghasilkan kecepatan 5 km/j dan gigi *reverse*/mundur selalu aktif.

3.6.3 *Reward Function*

Reward function (fungsi imbalan) adalah seperangkat aturan yang memberitahu sebuah agen apakah tindakan yang dilakukannya itu "baik" atau "buruk". *Reward function* diperlukan agar agen menyelesaikan tugasnya, agen mungkin tidak akan menyelesaikan *training*-nya jika *reward*nya sedikit. Adapun *reward function* yang ada adalah pengembangan dari penelitian yang dilakukan oleh Shidqi [5]. Agen dikenai penalti pada jarak dan perbedaan sudut yaw dari tempat parkir. Selain itu, agen juga dikenai penalti bila menabrak kendaraan ataupun objek lain. *Reward function* yang digunakan pada penelitian ini terdiri dari lima komponen seperti yang ada pada Persamaan 3.19.

$$reward = (1 - w)P_d + wP_\psi + P_c + P_b + R_p + R_d. \quad (3.19)$$

Reward function didesain untuk mengurangi jarak antara agen dan koordinat target P_d dan perbedaan sudut *yaw* P_ψ. Perbedaan sudut *yaw* adalah deviasi dari orientasi agen dan orientasi agen ketika parkir yang telah ditentukan. Ada pula *weight* w yang ditambahkan untuk menekankan mana yang lebih penting antara jarak dan perbedaan sudut *yaw*. Dalam skenario ini, *weight* yang dipilih adalah 0.65. Agen akan diberi hukuman apabila bertabrakan dengan objek lainnya P_c, seperti kendaraan, gerbang, tiang lampu atau objek lainnya. Selain itu, agen diberikan hukuman ketika mengeksplor di luar area yang telah ditentukan pada P_b. Agen diberikan *reward* apabila berhasil parkir R_p. Adapun persamaan dari semua

variabel *reward function* ada pada Persamaan 3.20 sampai dengan Persamaan 3.25, dimana *subscript* p adalah kondisi terparkir, lalu *subscript* i adalah kondisi awal.

$$P_d = -\sqrt[2]{\frac{(x - x_p)^2 + (y - y_p)^2}{(x_i - x_p)^2 + (x_i - x_p)^2}} \tag{3.20}$$

$$P_\psi = -\frac{arctan2[\sin(\psi_p - \psi), \cos(\psi_p - \psi)]}{\pi} \tag{3.21}$$

$$P_c = \begin{cases} 0, & \text{atau,} \\ -1, & \text{jika terjadi tabrakan} \end{cases} \tag{3.22}$$

$$P_b = \begin{cases} 0, & \text{atau,} \\ -4, & \text{jika mengeksplor di luar area} \end{cases} \tag{3.23}$$

$$R_p = \begin{cases} 3, & \text{jika terparkir,} \\ 0, & \text{atau} \end{cases} \tag{3.24}$$

$$R_d = \begin{cases} -2 & \text{jika jarak} < 120, \\ -4 & \text{jika jarak} < 100 \\ 0 & \text{jika jarak} > 120 \end{cases} \tag{3.25}$$

Ada beberapa perubahan yang dilakukan pada skenario parkir paralel berdasarkan kebiasaan agen. Perhitungan jarak dirubah dengan perpindahan pada garis y untuk mendukung agen untuk berhasil menuju ke tempat parkir. Alasan dari perubahan itu adalah dikarenakan agen belajar untuk mengurangi jarak dengan cara berjalan mundur pada garis lurus. Agen diberikan *reward* untuk mengurangi perpindahan pada garis y untuk mendukung kebiasaan.

Selain itu, agen juga diberikan *reward* untuk melakukan gerakan belok ke kanan sampai titik tertentu R_{steer}, dimana itu adalah garis batas dan diberikan hukuman ketika agen memasuki trotoar P_{side}. Hukuman ketika agen memasuki trotoar bisa dilihat dengan melihat nilai koordinat z dari agen. Yang terakhir, *weight* disesuaikan berdasarkan koordinat u dan perpindahan pada garis y. Adapun *reward function* pada skenario parkir paralel adalah pada Persamaan 3.26 sampai dengan Persamaan 3.30.

$$reward = (1 - w)d_y + wP\psi + P_c + P_b + R_p + R_d + R_{steer} + P_{side}, \quad (3.26)$$

yang mana detail dari beberapa variabel ini adalah Persamaan 3.22.

$$d_y = \left| \frac{y - y_p}{y_i - y_p} \right| \quad (3.27)$$

$$R_{steer} = \begin{cases} +0.1, & \textit{steering} > 0 \text{ jika y} > 64.5 \\ 0, & \text{atau lainnya,} \end{cases} \quad (3.28)$$

$$P_{side} = \begin{cases} 0, & \text{atau lainnya,} \\ -1, & \text{z} > 0.002 \end{cases} \quad (3.29)$$

$$R_p = \begin{cases} 1 - d_y, & \text{y} \leq 63.5, \\ 0, & \text{atau lainnya} \end{cases} \quad (3.30)$$

3.6.4 *Terminal State*

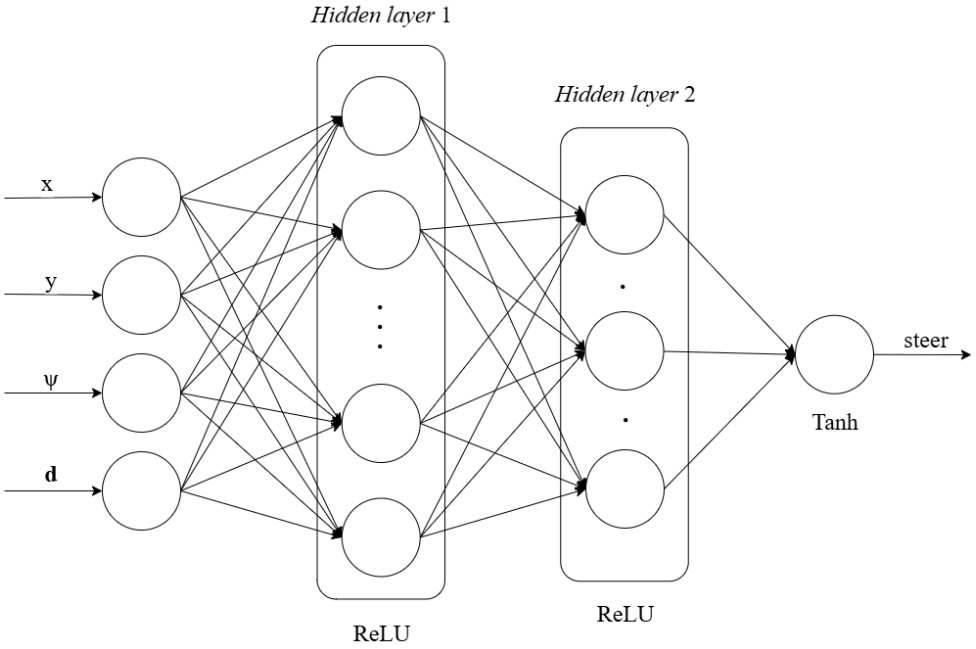
Terminal state adalah kondisi dimana episode berakhir jika agen mencapai para beberapa kondisi spesifik. Peneliti menerapkan beberapa kondisi untuk mencapai *terminal state*, yaitu: terparkir, tabrakan, mengeksplor di luar area batas dan mendekati objek sekitar pada rentang 50cm. Hal ini dilakukan agar *training* dilakukan oleh agen secara efisien, yang mana adalah metode memberhentikan *training* dengan memberhentikan proses episode dan berlanjut ke episode selanjurnya yang terbukti dapat meningkatkan performa [5].

Agen dianggap terparkir ketika berada di dalam kotak pada gam dan perbedaan sudut *yaw* dibawah 5°. Episode akan berakhir ketika agen melakukan tabrakan, mengeksplor di luar rentang yang ditentukan dan mendekati objek pada rentang 50cm. Hal ini dilakukan untuk membuat agen belajar dengan efisien dan mendukung agen untuk terparkir.

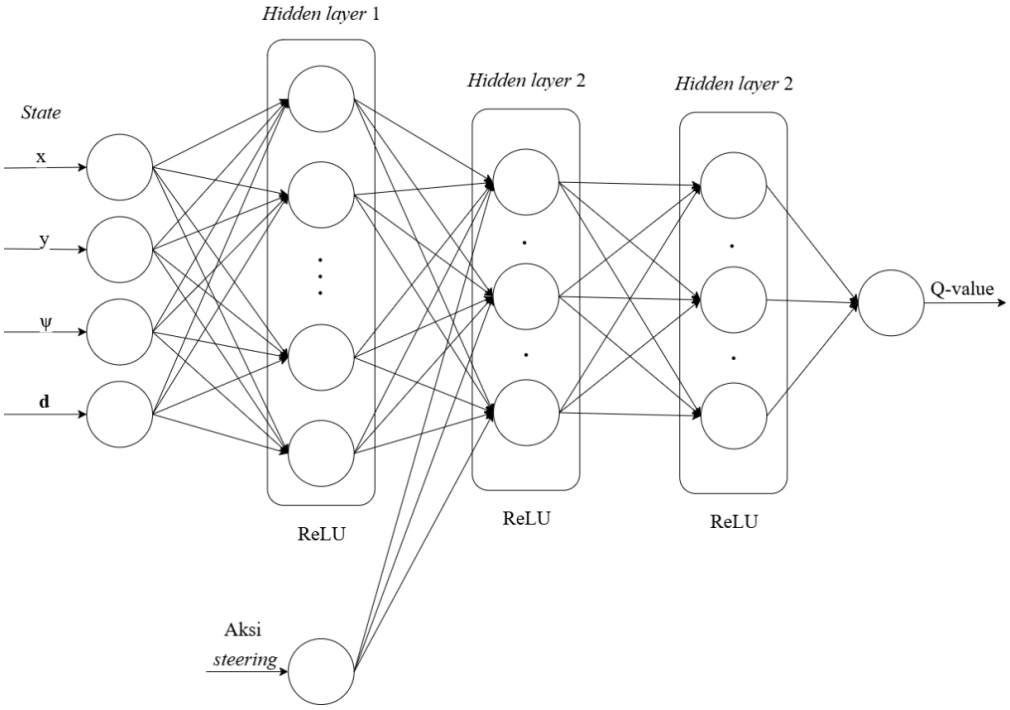
3.6.5 Arsitektur *Neural Network*

Desain dari *neural network* yang digunakan berdasarkan penelitian yang dilakukan oleh Fujimoto dkk [39]. Jaringan dibagi menjadi dua, yaitu *actor network* dan *critic network*. Kedua jaringan memiliki dua *hidden layer*, dengan *hidden layer* yang pertama terdiri dari 400 neuron dan kedua terdiri dari 300 neuron. Namun

dikarenakan agen sulit untuk untuk mencapai keadaan konvergen, maka ditambahkan *hidden layer* pada *critic network*. Adapun ilustrasi dari *actor network* dapat dilihat pada Gambar 3.13 dan *critic network* pada Gambar 3.14.



Gambar 3.13 *Actor network*



Gambar 3.14 *Critic network*

Input dari *actor network* adalah *tuple* dari keadaan posisi, oritentasi dan jarak dengan benda di sekitar agen (x, y, ψ, d). Lalu *input* ini akan dimasukkan ke dalam *hiden layer* pertama untuk diitung dan dilakukan non linearisasi menggunakan fungsi ReLU. Setelah iru, nilai akan dimasukkan kepada *hidden layer* kedua. *Output* dari *hidden* kedua ini kemudian dimasukkan pada *layer output* yang diaktivasi menggunakan Tanh untuk membatasi *output* menjadi [-1, +1]. *Output* dari *layer output* kemudian diskalakan agar sesuai dengan *state space* untuk *input steering* dengan mengalikannya dengan *ibout steering* maksimum.

Di sisi lain, *input critic network* adalah *state* dan *output action* dari *actor network*. Berbeda dengan *state*, *action* diteruskan langsung ke *hidden layer* kedua. *Hidden layer ketiga* adalah konfigurasi yang sama dengan *hidden layer* kedua. Lapisan ini ditambahkan agar agen dapat mempelajari tugas yang lebih sulit, seperti parkir paralel. *Output*nya dalah *Q-value* yang menentikan sebarapa baik antara *state-action*.

3.6.6 *Hyperparameter*

Pada Tabel 3.6 menunjukkan *hyperparameter* yang digunakan untuk model TD3.

Tabel 3.6 Spesifikasi *hyperparameter*

Hyperparameter	**Deksripsi**
Tingkat pembelajaran *actor*	0.0003
Tingkat pembelajaran *critic*	0.0003
Ukuran *batch*	64
S*oft update factor*	0.005
Discount rate	0.99
Optimizer	Adam
Frekuensi *actor update*	2
OU *noise*	$\theta = 0.15$, $\mu = 0$, $\sigma = 0.3$
Regularisasi *actor* L2	0.0001
Regularisasi *critic* L2	0.0001
Ukuran *replay buffer*	1000000
Nilai maksimum *noise factor*	0.0001
Nilai minimum *noise factor*	0.0001
Noise factor decay	0.1
Frekuensi *noise factor decay*	Setiap 20000 langkah waktu

Pada tahap awal, *hyperparameter* yang digunakan sama dengan TD3 *stable baselines* dari OpenAI [40]. *Stable baseline* terbukti sebagai nilai permulaan yang baik. Hal yang peneliti ubah dari *stable baseline* adalah ukuran *batch* dan regularisasi L2 untuk *actor* dan *critic*. Awalnya, ukuran *batch* adalah 100. Tidak

ada aturan untuk menentukan nilai yang betul untuk ukuran *batch*, beberapa penelitian menunjukkan bahwa ukuran *batch* yang kecil menghasilkan performa yang lebih baik [41].

Regularisasi L2 pun ditambahkan untuk menyetabilkan proses *training* dengan cara menghindari *update weight* yang besar. *Noise* yang dihasilkan menggunakan *noise* OU dan dikalikan dengan *noise factor*. *Noise factor* akan meluruh setiap 20000 langkah sekali, kira-kira 200 episode untuk memastikan transisi dari fase eksplorasi dan eksploitasi.

Agen mengalami *loss* yang sangat besar. Fenomena ini terjadi ketika agen lupa tentang apa yang telah ia pelajari dari pengalaman sebelumnya. Fenomena ini terjadi pada skenario parkir tegak lurus dimana agen dapat melakukan parkir namun nilai *cumulative reward* masih tidak membaik.

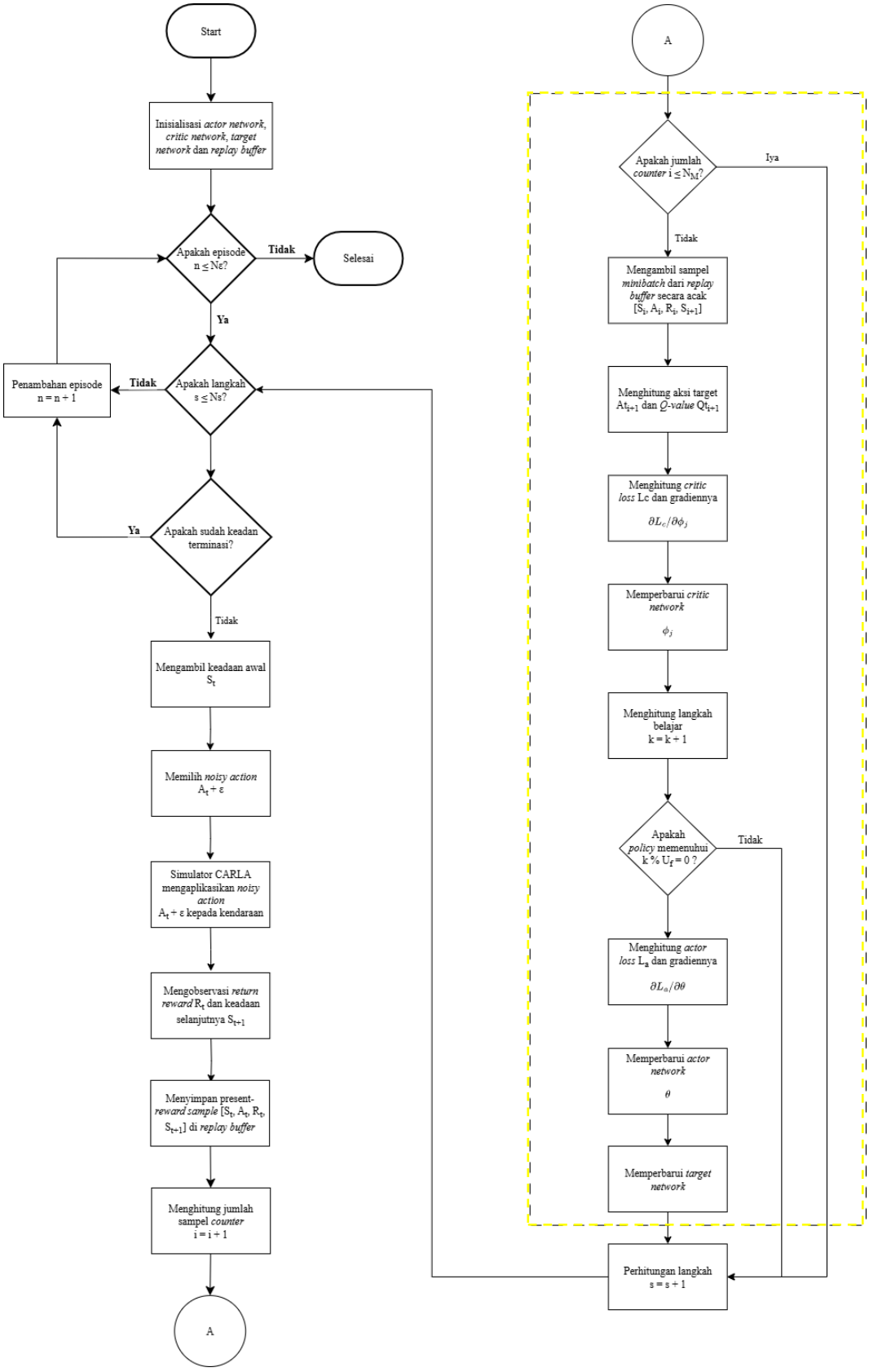
3.6.7 Perulangan *Training*

Pada Gambar 3.15 mengilustrasikan perulangan *training*. Perulangan dibagi menjadi dua bagian, yaitu perulangan simulasi dan perulangan belajar. Perulang simulasi dimulai dengan menginisialisasi *actor network*, *critic network*, *target network* dan *replay buffer*. Perulangan ini akan terus berulang sama nilai maksimum episode tercapai.

Episode juga berhenti apabila mencapai keadaan *terminal state* atau *time step* maksimal, yang mana adalah 1000. Tahap simulasi adalah perulangan RL dimana agen mengobservasi kondisi awal, melakukan aksi dengan *noise* ε yang mana diaplikasikan pada simulator CARLA, mendapatkan *reward* dari aksinya dan mengobservasi keadaan baru. Perulangan dilakukan sebanyak jumlah episode yang telah ditentukan, yaitu 1000. Semua informasi ini disimpan di *replay buffer*.

Apabila nilai *reward* lebih atau sama dengan ukuran *batch*, maka proses akan masuk kepada tahap perulangan belajar. Pembelajaran dimulai dengan *sampling* random informasi dari *replay buffer* dan menghitung aksi target dan *Q-value* target. *Q-value* target digunakan untuk menghitung *critic loss*. Akibatnya, gradien *critic*

loss dihitung untuk memperbarui *critic weight* ϕ. Setiap dua kali *critic network* diperbarui, *actor network* memperbarui dirinya sekali. Hal ini dilakukan dengan mengambil gradien dari *actor loss* dan menguranginya. Selain itu, *target network*, *target network*, *target actor*, target *critic* pertama dan kedua juga diperbarui menggunakan *soft update*.



Gambar 3.15 *Flowchart* perulangan *training*

3.7 Integrasi Sistem Persepsi dengan *Deep Reinforcement Learning*

Integrasi antara sistem persepsi dengan *deep reinforcement learning* (DRL) bertujuan agar informasi persepsi (gambar RGB) diubah menjadi nilai jarak dan diubah menjadi representasi *state space* yang dapat dipahami oleh algoritma DRL sehingga agen dapat melakukan aksi (*steering* dan *throttle*) secara *real-time* dan mendapatkan *reward*. Adapun tahap-tahapnya adalah sebagai berikut.

3.7.1 Alur Sistem

Alur sistem adalah langkah atau tahapan yang menjelaskan bagaimana suatu sistem atau aplikasi beroperasi, mulai dari *input* sampai dengan menghasilkan *output*. Alur sistem melibatkan data, informasi dan/atau instruksi dalam sistem. Adapun alur sistem pada penelitian ini dapat dilihat pada Gambar 3.16 berikut ini.

Gambar 3.16 Alur sistem

1. Akuisisi Data Sensor

Pada tahap ini, kamera stereo mengambil gambar dari lingkungan pada simulator CARLA. Gambar pertama digunakan sebagai kalibrasi stereo. Setelah kamera dikalibrasi, gambar akan di *preprocessing* untuk dilanjutkan dengan rektifikasi. Selain itu, data pun diambil dari odometer berupa lokasi (koordinat kendaraan) dan orientasi (rotasi pada setiap sumbu) yang ada pada kendaraan.

2. Ekstraksi Fitur Persepsi

Setelah dilakukan rektifikasi pada tahap akuisisi data, gambar akan dijadikan bahan untuk mengestimasi kedalaman dengan menggunakan *semi-global block matching* dengan *output* peta disparitas. Dari peta disparitas ini kita dapat mengetahui jarak dari suatu objek dengan kendaraan. Nilai jarak inilah yang akan dikirimkan sebagai salah satu *state space* DRL. Selain untuk menghitung jarak, gambar juga digunakan untuk mendeteksi objek oleh sistem deteksi objek.

3. Mengirimkan *State* ke DRL

Nilai jarak yang sebelumnya dihasilkan pada tahap ekstraksi fitur persepsi akan dikirimkan sebagai salah satu *state space*.

4. Pelatihan Agen DRL

Pelatihan dilakukan dengan algoritma TD3 sebagai algoritma DRL yang digunakan pada proses pelatihan (*training*).

5. Evaluasi dan Validasi

Pada tahap ini akan dilakukan evaluasi dengan cara menginterpretasikan data dari sistem persepsi dengan metriks akurasi seperti RMSE dan juga beberapa grafik penilaian model seperti *cumulative reward*, *actor loss* dan *critic loss*. Lalu untuk validasi akan dilakukan interpretasi dari bentuk jalur lintasan yang dihasilkan dari proses *training*.

3.7.2 Tools yang Digunakan

Berikut ini dalah *tools* yang digunakan untuk merealisasikan integrasi antara sistem persepsi dengan model *deep reinforcement learning* seperti dijelaskan pada Tabel 3.7.

Tabel 3.7 Tools yang Digunakan dalam Penelitian

Komponen	***Library/Framework***	**Keterangan**
Simulator	CARLA	Simulator kendaraan otonom
Persepsi (Jarak)	OpenCV	StereoSGBM
Persepsi (Deteksi)	YOLOv8	Model YOLOv8x *pretrain*
DRL	PyTorch	Implementasi TD3 dengan *stable baseline*
Visualisasi	Matplotlib	Plot kurva *reward* dan *loss training*