**SIMULASI NAVIGASI OTONOM PADA GAME TRACKMANIA NATION MENGGUNAKAN TEKNIK COMPUTER VISION**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS SANATA DHARMA**

**YOGYAKARTA**

**2024**

# KATA PENGANTAR

# Daftar Isi

[KATA PENGANTAR i](#_Toc193278928)

[Daftar Isi 1](#_Toc193278929)

[Daftar gambar 3](#_Toc193278930)

[Daftar tabel 4](#_Toc193278931)

[BAB 1 Pendahuluan 5](#_Toc193278932)

[1.1 Latar Belakang 5](#_Toc193278933)

[1.2 Rumusan Masalah 7](#_Toc193278934)

[1.3 Batasan Masalah 7](#_Toc193278935)

[1.4 Batasan Istilah 7](#_Toc193278936)

[1.5 Tujuan Penelitian 8](#_Toc193278937)

[1.6 Manfaat Penelitian 8](#_Toc193278938)

[1.7 Sistematika Penulisan 8](#_Toc193278939)

[BAB 2 Tinjauan Pustaka 9](#_Toc193278940)

[2.1 Tinjauan Pustaka Hasil Riset 9](#_Toc193278941)

[2.2 Landasan Teori 10](#_Toc193278942)

[2.2.1 Computer Vision 10](#_Toc193278943)

[2.2.2 Matriks Evaluasi 17](#_Toc193278944)

[2.2.3 Tools Tambahan 18](#_Toc193278945)

[BAB 3 Metodologi Pengembangan sistem 20](#_Toc193278946)

[3.1 Flowchart Pengembangan Sistem 20](#_Toc193278947)

[3.2 Pengumpulan Data(why is it image and not video) 21](#_Toc193278948)

[3.3 Image Processing 22](#_Toc193278949)

[3.3.1 Grayscalling 22](#_Toc193278950)

[3.3.2 Noise Reduction 23](#_Toc193278951)

[3.3.3 Canny Edge Detection 25](#_Toc193278952)

[3.3.4 Region of Interest 30](#_Toc193278953)

[3.3.5 Perspective Transform 31](#_Toc193278954)

[3.4 Lane Detection 32](#_Toc193278955)

[3.4.1 Sliding Window 32](#_Toc193278956)

[3.5 Algoritma Mengemudi Berdasarkan Lane 35](#_Toc193278957)

[3.6 Evaluasi dan Analisis Matrik 37](#_Toc193278958)

[3.6.1 Matriks Kualitatif Lane Detection 37](#_Toc193278959)

[3.6.2 Matriks Track-Level 38](#_Toc193278960)

[BAB 4 HASil simulasi dan PEMBAHASAN 40](#_Toc193278961)

[4.1 Pengumpulan Data 40](#_Toc193278962)

[4.2 Image Processing 41](#_Toc193278963)

[4.2.1 Canny Edge Detection 41](#_Toc193278964)

[4.2.2 Region of Interest 41](#_Toc193278965)

[4.2.3 Perspective Wrapping 42](#_Toc193278966)

[4.3 Sliding Window 45](#_Toc193278967)

[4.4 Parameter Tunning 46](#_Toc193278968)

[4.5 Algoritma Mengemudi 47](#_Toc193278969)

[Daftar pustaka 51](#_Toc193278970)

# Daftar gambar

[**Gambar 2.1** Aplikasi gaussian blur pada noise 7](#_Toc184811824)

[**Gambar 3.1** Flowchart metodologi pengembangan sistem 14](#_Toc184811825)

[**Gambar 3.2** Flowchart simulasi 15](#_Toc184811826)

[**Gambar 3.3** Hasil tangkapan layar disimpan ke dalam dua folder 16](#_Toc184811827)

[**Gambar 3.4** Representasi RGB pada gambar 3x4 17](#_Toc184811828)

[**Gambar 3.5** Representasi grayscale pada gambar 3x4 17](#_Toc184811829)

[**Gambar 3.6** Noise pada gambar 4x4 dan 3x3 Gaussian kernel 17](#_Toc184811830)

[**Gambar 3.7** Hasil gambar 3.6 setelah menggunakan gaussian blur 17](#_Toc184811831)

[**Gambar 3.8** Non-Maximum Suppression 19](#_Toc184811832)

[**Gambar 3.9** Contoh NMS pada piksel (1,1) 19](#_Toc184811833)

[**Gambar 3.10** Empatarah gradien 19](#_Toc184811834)

[**Gambar 3.11** Eliminasi tepi yang lemah 20](#_Toc184811835)

[**Gambar 3.12** Implementasi RoI pada hasil Canny edge detection 20](#_Toc184811836)

[**Gambar 3.13** Histogram intensitas piksel untuk mengidentifikasi posisi awal lane 21](#_Toc184811837)

[**Gambar 3.14** Sliding window pada hasil deteksi tepi 22](#_Toc184811838)

[**Gambar 3.15** Polynomial linear (kiri) dan polynomial kuadrat (kanan) 22](#_Toc184811839)

[**Gambar 3.16** Kontrol kemudi berdasarkan jalur 23](#_Toc184811840)

# Daftar tabel

[**Tabel 3.1** Sobel filter horizontal (x) 18](#_Toc184811992)

[**Tabel 3.2** Sobel filter vertikal (y) 18](#_Toc184811993)

[**Tabel 3.3** Hasil kalkulasi *gradient magnitude* 18](#_Toc184811994)

[**Tabel 3.4** Hasil kalkulasi *gradient direction* 18](#_Toc184811995)

[**Tabel 3.5** Hasil operasi NMS 19](#_Toc184811996)

# Pendahuluan

## Latar Belakang

Kecelakaan berkendara merupakan masalah serius yang dapat disebabkan oleh berbagai faktor. Salah satu penyebab utamanya adalah distraksi, seperti penggunaan *handphone* saat berkendara, yang menjadi faktor risiko kecelakaan bagi pengemudi remaja hingga dewasa [1]. Menurut Badan Pusat Statistik, kecelakaan jalan yang terjadi di Indonesia didominasi oleh sepeda motor, diikuti oleh pengendara mobil sedan, dan paling banyak terjadi pada kelompok usia muda, yaitu 15-29 tahun. Peningkatan jumlah kendaraan darat di Indonesia turut memperbesar potensi kecelakaan di jalan raya. Kementerian Komunikasi dan Informatika (Kominfo) menyatakan bahwa rata-rata tiga orang meninggal dunia setiap jamnya akibat kecelakaan lalu lintas di Indonesia. Salah satu upaya yang dapat dilakukan untuk mengurangi angka kecelakaan dan mengatasi distraksi akibat kesalahan manusia (*human error*) adalah dengan mengembangkan teknologi kendaraan otomatis (*autonomous driving*)[2], [3]*.*

Kemudi otomatis (*autonomous driving*) atau mobil otonom (*self-driving*), telah dipelajari dan dikembangkan oleh berbagai universitas, pusat penelitian, perusahaan mobil, dan perusahaan di seluruh dunia sejak pertengahan tahun 1980 [4]. Penelitian ini bertujuan menciptakan kendaraan yang mampu mendeteksi, memahami, dan menavigasi lingkungannya tanpa bantuan manusia. Minat akademis terhadap teknologi ini meningkat karena potensinya untuk mengurangi kecelakaan lalu lintas, meningkatkan efisiensi transportasi, dan mendukung inovasi di berbagai bidang terkait. Untuk mencapai tujuan ini, tugas yang paling penting merupakan mempelajari aturan-aturan dalam berkendara yang mampu memberikan luaran (*output*) berupa kontrol untuk mengemudi (gas, rem, belok kiri dan kanan, berhenti dsb.) berdasarkan *input* lingkungan sekitar [5], [6]. Perkembangan teknologi komputasi seperti sensor, *computer vision*, *machine learning,* dan *hardware acceleration*, serta berbagai jenis perangkat komunikasi untuk mendeteksi masukan pada beberapa tahun belakangan semakin menarik perhatian komunitas *automotif* serta akademisi [7].

Namun, minimnya akses ke *platform* pengujian yang aman dan terjangkau untuk pengembangan sistem mengemudi otomatis menjadi salah satu tantangan utama dalam penelitian dan pengembangan kendaraan otonom. Pengujian model yang komprehensif dan menyeluruh memiliki peran penting dalam melatih model mobil otonomuntuk menangani berbagai skenario yang mungkin terjadi di jalan umum. Pelatihan dan pengujian fisik pada jalan umum seringkali tidak aman, membutuhkan biaya yang besar, dan tidak selalu dapat direproduksi secara konsisten. Melakukan operasi pada sebuah mobil tanpa kemudi sendiri membutuhkan dana dan tenaga manusia yang signifikan. Sebuah mobil tanpa kemudi tidak cukup untuk mengumpulkan data yang mampu mencakupi berbagai skenario untuk kepentingan latihan (*training*) dan validasi (*validation*) [8].

Untuk mengatasi keterbatasan tempat pelatihan yang aman serta biaya yang signifikan dalam memproduksi model mobil tanpa kemudi, pelatihan dan validasi model dapat dilakukan melalui simulasi dengan memberikan lingkungan yang aman dan terkendali sebelum diterapkan di dunia nyata [7], [9], [10]. Simulasi telah digunakan untuk melatih model mengemudi sejak masa awal penelitian mobil otonom [11]. Beberapa tahun terakhir, permainan simulasi balapan dan beberapa permainan komersial telah digunakan untuk mengumpulkan data, melatih, serta mengevaluasi baik model mobil otonom maupun *perception system* [9], [12], [13], [14], [15], salah satunya adalah Trackmania [14]*.* Dalam hal ini, *computer vision* memiliki peran penting dalam *perception system* karena memungkinkan mobil otonom untuk memahami lingkungan di sekitarnya berdasarkan analisis gambar dan video. Melalui teknik-teknik seperti deteksi objek, segmentasi gambar, dan pelacakan objek, *computer vision* dapat membantu mobil otonom mendeteksi jalur, rambu lalu lintas, kendaraan lain, serta objek-objek di sekitar. Simulasi dalam permainan memungkinkan pengujian dan evaluasi teknik-teknik *computer vision* ini dalam lingkungan yang aman dan terkendali [9], [16].

Salah satu pustaka yang populer dalam bidang *computer vision* merupakan *Open Source Computer Vision* (OpenCV) yang menyediakan berbagai fungsi untuk kepentingan memecahkan masalah berkaitan dengan *computer vision* [17]. Dalam konteks simulasi mobil otonom, pustaka ini menyediakan fitur-fitur penting seperti *edge detection* dan *line recognition*, yang memungkinkan sistem untuk mengenali elemen-elemen di sekitar, khususnya garis-garis jalur (*lane lines*) [18], [19]. Kemampuan ini sangat berguna dalam membantu mobil otonom memahami dan menavigasi lingkungannya di dalam simulasi secara *realtime* [15], [19]. Penelitian yang dilakukan oleh (Yang X & Ling Z, 2015) menunjukkan performa OpenCV dalam mendeteksi *lane line* secara *realtime* dengan akurasi 84.5% dan *false alarm rate* mencapai 12.0% yang memanfaatkan *edge detection* serta *Hough transformation.*

Berdasarkan uraian tersebut, OpenCV memiliki performa yang sangat baik untuk memahami lingkungan di sekitar mobil otonom terutama mengenali jalur (*lane*) secara *realtime*. Penelitian ini akan memanfaatkan fungsi-fungsi yang disediakan oleh OpenCV untuk mengenali lingkungan pada permainan Trackmania Nations sebagai *platform* untuk mensimulasikan mobil otonom.

## Rumusan Masalah

Berdasarkan latar belakang tersebut, maka rumusan masalah dalam penelitian ini adalah mengevaluasi kinerja simulasi navigasi otonom pada permainan TrackMania dengan mengintegrasikan teknik visi komputer untuk mendeteksi jalur menggunakan *sliding window* dan algoritma mengemudi berbasis aturan.

## Batasan Masalah

Pada penelitian ini terdapat beberapa batasan, yaitu:

1. Lingkungan permainan (*game environment*) yang digunakan dalam simulasi masih memiliki keterbatasan seperti tidak memiliki sistem lalu lintas, minimnya rintangan, dan hanya terdapat satu kondisi waktu (siang hari).
2. Simulasi berfokus pada melakukan navigasi sepanjang jalur tunggal (*single lane*).

## Batasan Istilah

Adapun beberapa istilah yang digunakan dalam penelitian ini: (explain briefly)

1. *Computer* *Vision*
2. *OpenCV* (*Open Source Computer Vision*)
3. *Grayscalling*
4. *Edge Detection*
5. *Region of Interest*
6. *Perspective Transform*
7. *Sliding Window*
8. Kurva (*curvature*)
9. *PyautoGUI*
10. *PyDirectInput*

## Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah:

1. Mengetahui performa OpenCV dan algoritma *rule-based* dalam simulasi *self-driving* pada *game* Trackmania Nation secara *realtime.*
2. Memberikan alternatif sistem *self-driving* yang mampu bekerja pada perangkat komputer dengan kemampuan komputasi yang rendah.

## Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat baik secara teoritis maupun secara praktis bagi beberapa pihak, dengan uraian sebagai berikut:

1. Manfaat Teoritis
2. Manfaat Praktis

## Sistematika Penulisan

# Tinjauan Pustaka

## Tinjauan Pustaka Hasil Riset

Penelitian yang dilakukan oleh Aditya dalam menganalisis jalur jalan (*lane detection*) menggunakan pustaka OpenCV seperti Canny dan Hough Transform pada jalan terstuktur berhasil mendeksi jalur jalan dengan cukup baik. Akan tetapi, penggunaan Hough Transform hanya terbatas pada jalur yang relatif lurus (*straight lane*). Oleh karena itu, algoritma ini akan kesulitan dalam mendeteksi jalur jalan dengan berbagai skenario seperti jalan yang berbelok [19].

Akash Punagin dan Sahana Punagin dalam penelitiannya juga menganalisis berbagai teknik deteksi jalur menggunakan OpenCV pada jalan terstruktur. Penelitian tersebut menggunakan teknik-teknik seperti Gaussian Blur untuk mengurangi noise, tiga jenis algoritma deteksi tepi seperti Sobel, Laplacian dan Canny Edge Detection untuk mendeteksi tepi lintasan, serta transformasi Hough Line untuk mendeteksi garis. Berdasarkan tiga jenis jenis algortima deteksi tepi yang digunakan, Sobel dan Laplacian mendeteksi lane dengan tidak sempurna karena tepi yang dihasilkan oleh kedua algoritma cenderung tebal. Sebaliknya, Canny merupakan metode yang paling efisien dalam mendeteksi tepi jalan dengan menghilangkan *noise* dan tekstur yang tidak relevan menggunakan *non-maximum suppression* [16].

Wang dan Fan melakukan penelitian deteksi pada jalur yang lebih kompleks menggunakan algoritma *sliding window* dan *polynominal fitting* untuk mengekstrak garis pada jalur, serta menggunakan *edge detection* dan *color filter* dari pustaka OpenCV untuk mendeteksi yang mengindikasikan jalan. Penelitian ini menghasilkan mendapatkan performa yang jauh lebih akurat dibandingkan dengan Hough Transform serta lebih tahan terhadap berbagai skenario pada jalan [20].

Christopher Erdelyi dalam penelitiannya mendemonstrasikan penggunaan fungsi-fungsi OpenCV untuk memproses gambar dari game balapan secara otomatis. Penelitian ini melibatkan pengubahan keluaran visual game menjadi gambar hitam putih, dengan batas-batas (boundary) lintasan ditandai menggunakan algoritma seperti Canny Edge Detection dan Hough Line Transformation. Gambar hasil proses ini kemudian dievaluasi oleh algoritma mengemudi sederhana untuk menentukan apakah batas lintasan yang dihasilkan cukup membantu kendaraan dalam bernavigasi melalui jalur. Hasil penelitian menunjukkan bahwa algoritma berbasis OpenCV dapat menghasilkan data visual yang mendukung pengemudian otomatis. Dalam eksperimen, program berhasil menghindari batas lintasan dan menyelesaikan beberapa putaran tanpa kendali manusia. Hal ini dicapai dengan serangkaian penyempurnaan algoritma hingga 60 kali, mencatat keberhasilan kendaraan saat menghadapi belokan. Pada pengujian awal dengan input acak, kendaraan selalu menabrak sebelum mencapai belokan pertama. Sebaliknya, dengan algoritma OpenCV, kendaraan tidak pernah mengalami kecelakaan sebelum belokan pertama selama 30 kali percobaan [15].

## Landasan Teori

### Computer Vision

*Computer Vision* (CV) adalah bidang antardisiplin yang berfokus pada bagaimana komputer dapat direkayasa untuk memperoleh pemahaman tingkat tinggi dari data berupa gambar digital maupun video. CV merupakan sub-bidang dari *Artificial Intelligence* (AI) yang bertujuan untuk mengotomatisasikan tugas-tugas yang dapat dilakukan oleh sistem penglihatan manusia, seperti pengenalan objek, analisis citra, dan pelacakan gerakan [21].

#### OpenCV

OpenCV (Open Source Computer Vision Library) adalah pustaka perangkat lunak sumber terbuka yang dirancang untuk pengolahan citra secara real-time. Dikembangkan oleh Intel pada tahun 1999, OpenCV menyediakan berbagai algoritma yang mendukung penyelesaian masalah dalam bidang CV. Pustaka ini mencakup algoritma pengolahan citra tingkat rendah, seperti edge detection, serta algoritma tingkat tinggi, seperti *face detection*, *feature matching*, dan *object tracking* [17]*.*

##### Grayscalling

Grayscaling adalah proses konversi gambar dari berbagai color spaces menjadi representasi dalam gradasi abu-abu. Warna abu-abu pada setiap piksel direpresentasikan dengan intensitas yang bervariasi dari 0 (hitam) hingga 255 (putih).

(1)

Proses ini berguna untuk menyederhanakan pengolahan citra, terutama dalam tugas-tugas yang tidak membutuhkan informasi warna penuh, seperti deteksi tepi [22], [23].

##### Gaussian Blurring

Gaussian Blurring adalah metode *smoothing* pada gambar menggunakan fungsi Gaussian. Teknik ini digunakan untuk mengurangi noise dan detail kecil pada gambar, sehingga mempermudah analisis lebih lanjut.

(2)

Dimana:

: nilai dari *Gaussian Blur* pada posisi (*i, j*)

***k****:* : ukuran dari kernel

***ρ*** : standar deviasi untuk *scalling factor*

***σ*** :standar deviasi

**π**: nilai pi untuk normalisasi

Dalam pemrosesan citra digital, Gaussian Blur sering digunakan sebelum proses seperti *edge detection* untuk menghasilkan *output* yang lebih akurat [23].



**Gambar 2.1** Aplikasi gaussian blur pada noise

##### Region of Interest

*Region of Interest* (ROI) adalah bagian tertentu yang diambil dari sebuah gambar untuk keperluan analisis. Proses ini memungkinkan fokus pada area yang relevan dalam sebuah gambar, sehingga pengolahan citra lebih efisien dan terarah [24].

##### Thresholding

*Thresholding* atau nilai ambang batas adalah proses segmentasi citra yang digunakan untuk membagi gambar berdasarkan intensitas piksel. Teknik ini dapat menyederhanakan analisis citra dengan mengisolasi elemen-elemen tertentu dari latar belakang.

1. Simple Thresholding

Pada setiap piksel gambar, nilai intensitas dibandingkan dengan nilai ambang batas tertentu. Jika intensitas piksel lebih kecil dari nilai ambang, piksel akan diubah menjadi nol (hitam). Sebaliknya, jika lebih besar, piksel akan diubah menjadi nilai maksimum (putih). Teknik ini cocok untuk gambar dengan pencahayaan seragam.

1. Double Thresholding

*Double thresholding* digunakan untuk membedakan tepi yang kuat, lemah dan tidak relevan. *Threshold* ini biasanya digunakan pada algoritma deteksi tepi seperti *canny edge detection*.

(3)

1. Adaptive Thresholding

*Adaptive thresholding* digunakan untuk gambar dengan variasi pencahayaan atau intensitas warna yang signifikan. Algoritma ini menentukan nilai ambang secara dinamis berdasarkan rata-rata atau median nilai piksel di sekitarnya. Dengan pendekatan ini, setiap area dalam gambar memiliki nilai ambang yang sesuai dengan kondisinya [25].

##### Image Segmentation

Dalam pemrosesan citra digital dan CV, *image segmentation* (segmentasi citra) atau peruasan citra adalah proses pembagian citra digital ke dalam beberapa bagian. Segmentasi citra digunakan untuk menyederhanakan penggambaran citra ke dalam bentuk yang lebih bermakna dan lebih mudah dianalisis [26]. Tujuan dari segmentasi citra adalah untuk membagi sebuah gambar menjadi beberapa bagian/segmen yang memiliki fitur atau atribut yang sama [23], [27].

##### Canny Edge Detection

*Edge Detection* atau deteksi tepi adalah cara-cara matematis untuk mengenali titik-titik dalam citra digital yang kecerahannya berubah drastis atau, secara formal, memiliki diskontinuitas. Tepi dibentuk oleh perubahan intensitas atau warna dalam sebuah gambar. Dengan melakukan deteksi tepi, data dalam gambar akan dikurangi secara signifikan dan memberikan struktur dari gambar untuk keperluan pemrosesan citra [16], [23], [26], [28].

Salah satu algoritma deteksi tapi yang populer adalah Canny Edge Detector diciptakan oleh John F. Canny pada tahun 1986. Algoritma ini memanfaatkan *multi-stage algorithm* yang memiliki tahapan-tahapan seperti *noise reduction*, *gradient calculation*, *Non-maximum suppression, double threshold,* dan *edge tracking by hysteresis* untuk mendapatkan hasil deteksi yang maksimal [28].

1. Noise Reduction

Algoritma deteksi tepi sangat sensitif terhadap *noise* pada gambar karena bergantung pada derivatif untuk menghitung perubahan berdasarkan variasi input. *Noise* dapat menyebabkan algoritma menginterpretasinya sebagai sebuah tepi. Salah satu cara untuk mengurangi *noise* adalah memanfaatkan Gaussian Blurring dengan memberikan filter pada elemen dengan frekuensi yang tinggi dalam sebuah gambar.

1. Gradient Calculation

Tahapan ini mendeteksi perubahan intensitas piksel yang signifikan pada gambar yang mengindikasikan tepi berdasarkan *gradient magnitude* dan *gradient direction*. Filter mendeteksi gambar pada sumbu x (horizontal) dan y (vertikal) kemudian menandai perubahan intensitas piksel pada kedua arah yang dapat dihitung berdasarkan rumus berikut:

(4)

Dimana:

***G*** : *gradient magnitude*

***Gx​*** : arah horizontal (sumbu *x*)

***Gy​*** : arah vertikal (sumbu *y*)

(5)

Dimana:

***Θ*** : *gradient direction*

***atan2(Gy,Gx)*** : menghitung arah dari gradien

1. Non-maximum suppression

Pada tahapan *gradient calculation*, intensitas tepi bervariasi dalam rentang nilai 0-255 menyebabkan beberapa tepi lebih tebal dari yang lain. Algoritma Non-maximum suppression digunakan untuk membuat hasil tepi lebih seimbang. Algoritma ini menelusuri intensitas setiap piksel dan membandingkannya dengan tetangga di sekitarnya, sehingga hanya tepi dengan nilai intensitas terbesar yang dipertahankan.

1. Double Threshold

Tahapan ini bertujuan untuk mengidentifikasi tepi dengan intensitas kecerahan kuat, lemah, dan tidak relevan. Piksel dengan intensitas kecerahan tinggi memiliki kontribusi paling kuat sebagai tepi, sehingga digunakan sebagai nilai nilai ambang batas atas (*high threshold*). Sementara itu, piksel dengan intensitas rendah digunakan untuk sebagai ambang batas bawah (*low threshold*) untuk mengabaikan intensitas piksel yang tidak relevan.

1. Edge Tracking by Hysteresis

Tahapan ini bertujuan untuk mengubah piksel dengan intensitas rendah menjadi kuat jika piksel-piksel di sekitarnya terdapat piksel dengan intensitas kuat.

##### Perspective Transform

Transformasi perspektif (perspective transform) adalah teknik dalam pengolahan citra yang digunakan untuk mengubah sudut pandang suatu gambar, misalnya dari tampilan biasa menjadi tampilan "bird's eye view" (pandangan dari atas). Dalam konteks deteksi jalur (lane detection), transformasi ini berguna untuk mengubah gambar jalan dari sudut pandang kamera mobil menjadi tampilan atas, sehingga jalur terlihat sejajar dan lebih mudah dianalisis. Proses ini melibatkan pemetaan titik-titik tertentu pada gambar asli ke titik-titik baru menggunakan matriks transformasi [29].

##### Hough Transform

Hough Transform adalah teknik ekstraksi fitur yang digunakan dalam Computer Vision, analisis citra, pengenalan pola, maupun pemrosesan citra digital. Teknik ini dirancang untuk menemukan bentuk tertentu, bahkan jika bentuk tersebut tidak sempurna atau memiliki gangguan. Salah satu aplikasi paling umum dari Hough Transform adalah deteksi garis lurus.

Secara dasar, garis lurus dapat direpresentasikan oleh persamaan*:*

(6)

Dimana:

**y**: sumbu vertikal

***x* :** sumbu horizontal

***m* :** *slope* (kemiringan)

***b* :** y-intercept

Namun, untuk mengatasi masalah seperti kemiringan vertikal di mana menjadi tak terdefinisi, Hough Transform menggunakan representasi polar:

(7)

Dalam persamaan ini:

* ρ adalah jarak tegak lurus dari asal ke garis, dan
* θ adalah sudut antara sumbu x dan garis tegak lurus dari asal ke garis yang dianalisis.

Pendekatan ini memungkinkan deteksi garis menjadi lebih stabil pada gambar berbasis piksel [30].

##### Teknik Sliding Window

Sliding Window adalah teknik untuk memecahkan masalah dengan menginisiasi sebuah jendela tetap (*fixed-window*) pada data input, kemudian menggeser jendela tersebut sepanjang data sambil melakukan operasi pada setiap posisi jendela. Teknik ini menggunakan dua penanda (*pointers*) untuk menunjuk awal dan akhir jendela*.*.

Dalam konteks *lane detection*, teknik ini digunakan untuk menemukan tepi dari garis jalur (*lane lines*). Jendela awal ditempatkan di bagian kiri dan kanan jalur pada area bawah gambar dan digeser ke atas hingga mencapai bagian atas gambar, mencari piksel yang menunjukkan jalur pada setiap posisi jendela. Ketika piksel jalur ditemukan, nilai piksel yang memiliki potensi sebagai jalur disimpan dalam dua daftar terpisah (kiri dan kanan). Nilai rata-rata (*mean*) dari posisi piksel ini kemudian digunakan untuk mengatur pusat jendela berikutnya pada setiap langkah [31], [32].

### Matriks Evaluasi

#### Kualitasi Lane Detection

Deteksi jalur adalah komponen penting dalam sistem mengemudi otomatis, yang berfungsi untuk mengenali dan mengikuti jalur atau garis yang ada di jalan. Teknik ini digunakan untuk membantu kendaraan tetap berada pada jalur yang benar dan menghindari penyimpangan. Berbagai metode dapat digunakan dalam deteksi jalur, mulai dari pengolahan citra dasar hingga penerapan algoritma canggih seperti jaringan saraf tiruan (*neural networks*). Dalam implementasi praktisnya, deteksi jalur menggunakan teknik-teknik seperti pemrosesan citra (*image processing*) untuk menemukan garis jalur yang terlihat di jalanan, dengan menggunakan algoritma seperti Canny *Edge Detection* dan *Hough* *Transform* [32], [33].

##### Lane Detection Consistency

Konsistensi deteksi jalur mengukur sejauh mana sistem dapat mendeteksi jalur secara stabil dan dapat diandalkan dalam berbagai kondisi jalan dan lingkungan (misalnya, perbedaan pencahayaan, cuaca, atau jenis jalan). Sistem deteksi jalur yang konsisten harus mampu mengikuti jalur dengan akurat tanpa sering mengalami kesalahan atau gangguan, bahkan ketika tanda jalur di jalan kurang jelas

##### Mean Square Error (MSE)

*Mean Squared Error* (MSE) adalah metrik yang umum digunakan untuk mengukur akurasi deteksi jalur. MSE menghitung rata-rata perbedaan kuadrat antara posisi jalur yang benar (*ground truth)* dan posisi jalur yang terdeteksi oleh sistem. Semakin kecil nilai MSE, semakin akurat sistem dalam mendeteksi jalur.

(8)

Dimana:

* = Ground truth (nilai jalur sebenarnya)
* = Nilai yang terdeteksi
* = Jumlah titik data

#### Matriks Track-Level (add the formula of curvature calculation)

##### Putaran Berhasil Ditempuh

Metrik ini mengukur kemampuan kendaraan untuk berhasil menavigasi setiap putaran atau belokan di sepanjang trek. Mengingat bahwa trek biasanya terdiri dari kombinasi jalur lurus dan belokan, kemampuan untuk menavigasi belokan dengan benar sangat penting. Metrik ini membantu menilai apakah sistem dapat menyesuaikan arah kendaraan dengan baik ketika menghadapi perubahan jalur [15].

##### Jarak Tempuh Sebelum Menabrak

Metrik ini mengukur sejauh mana kendaraan dapat melaju sebelum terjadi tabrakan atau kesalahan yang menyebabkan sistem keluar jalur. Ini mencerminkan seberapa baik sistem mendeteksi dan menghindari hambatan atau kesalahan. Semakin jauh jaraknya sebelum tabrakan, semakin baik kemampuan sistem untuk mengikuti jalur dengan aman

##### Tingkat Keberhasilan

Rasio keberhasilan mengukur sejauh mana kendaraan dapat menyelesaikan trek tanpa kegagalan. Metrik ini merupakan indikator kinerja keseluruhan, yang mencakup kemampuan deteksi jalur, pengambilan keputusan, dan pemulihan dari kesalahan. Rasio keberhasilan yang tinggi menunjukkan bahwa sistem mampu mengikuti jalur dengan stabil dan menghindari masalah secara konsisten.

**(9)**

### Tools Tambahan

#### PyAutoGUI

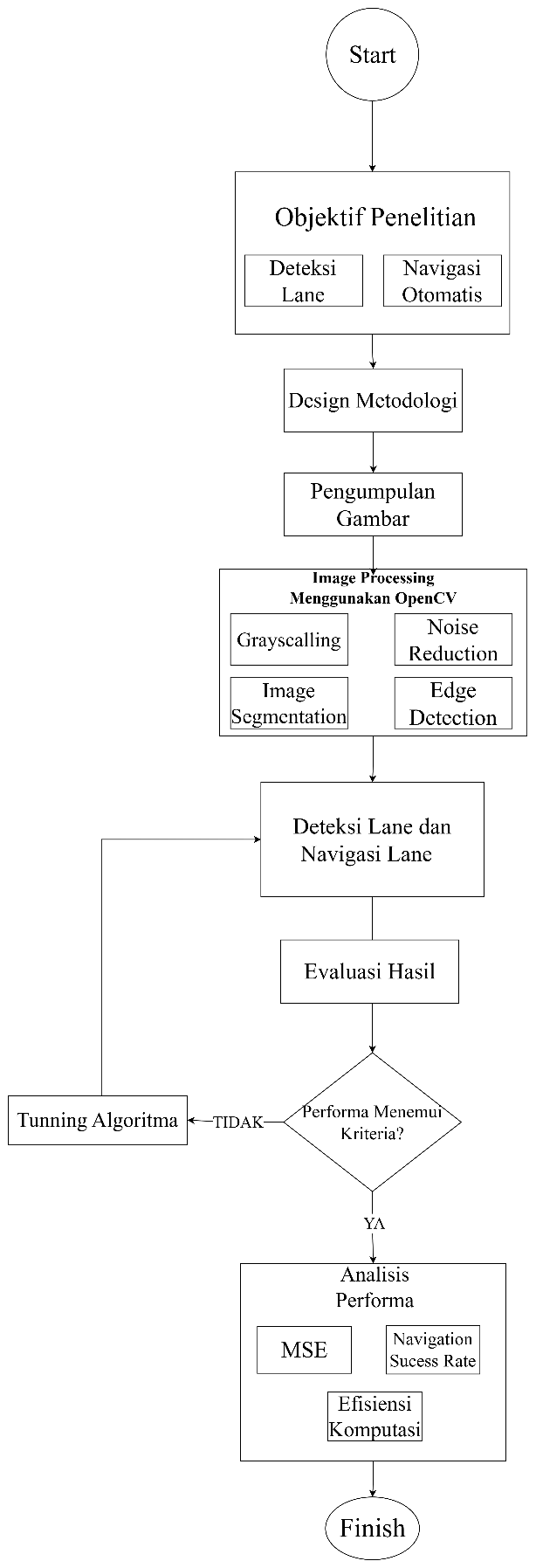
PyAutoGUI adalah pustaka Python yang memungkinkan pengguna untuk mengontrol mouse dan keyboard secara otomatis untuk berinteraksi dengan aplikasi-aplikasi yang berjalan di sistem operasi seperti Windows, macOS, dan Linux. Pustaka ini sangat berguna untuk mengotomatiskan tugas-tugas yang memerlukan interaksi manual, serta berjalan di kedua versi Python 2 dan 3 [34].

#### PyDirectInput

PyDirectInput adalah pustaka Python yang digunakan untuk mengirim input keyboard dan mouse secara langsung ke aplikasi, mengatasi masalah kompatibilitas dengan input standar di beberapa permainan atau aplikasi. Pustaka ini memungkinkan simulasi input perangkat keras lebih akurat, yang penting untuk aplikasi yang memerlukan kontrol presisi tinggi seperti dalam pengujian perangkat lunak atau aplikasi permainan.

# Metodologi Pengembangan sistem

## Flowchart Pengembangan Sistem



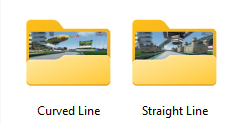
**Gambar 3.1** Flowchart metodologi pengembangan sistem

## Pengumpulan Data(why is it image and not video)



**Gambar 3.2** Flowchart pengumpulan data

Data dikumpulkan dengan merekam layar permainan. Proses ini melibatkan program Python untuk mengambil tangkapan layar dari permainan secara *real-time* dengan resolusi layar permainan 1080p (*windowed*) dan menyimpan gambar-gambar tersebut ke dalam folder khusus. Setiap gambar diberi ID unik untuk mempermudah pengelolaan dan identifikasi di tahap selanjutnya.



**Gambar 3.3** Hasil tangkapan layar disimpan ke dalam dua folder

Gambar hasil tangkapan layar dikelompokkan ke dalam dua folder berdasarkan jenis jalan. Dengan klasifikasi ini, data dapat diolah secara terpisah untuk menghasilkan sistem yang lebih akurat dan efisien:

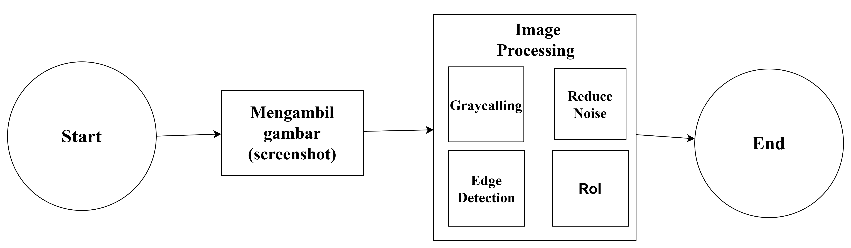
1. Jalan lurus (*straight line*): Pola piksel lebih teratur, sehingga algoritma lebih mudah mengenali tepi dan menentukan jalur.
2. Jalan berbelok (*curved line*): Mengandung lebih banyak variasi bentuk dan intensitas, sehingga membutuhkan pendekatan yang lebih kompleks dalam pendeteksian jalur

Pengumpulan gambar dilakukan secara *real-time* yaitu untuk memastikan bahwa gambar yang didapatkan mampu menggambarkan kondisi *track* atau jalur pada permainan yang sebenarnya.

Data yang diperoleh diambil pada kondisi jalan yang terlihat dengan jelas tanpa adanya *oclusion* seperti kendaraan lain, rambu lalu lintas maupun objek yang umunya terdapat di jalan seperti rambu-rambu lalu lintas. Karena kondisi *game* yang tidak memungkinkan untuk mengatur waktu (siang atau malam), maka data yang diolah terdiri dari satu kondisi waktu saja.

Posisi jalan pada gambar yang diambil juga dapat mempengaruhi kualitas dari hasil simulasi. Semakin jelas gambar jalan, maka hasil deteksi jalur juga akan lebih akurat. Oleh karena itu, posisi pengambilan gambar yang diambil pada simulasi ini adalah dari bagian jalan yang paling mendekati kamera hingga ujung dari jalan sepanjang sumbu y.

## Image Processing



**Gambar 3.4** Flowchart pemrosesan gambar

Setelah data terkumpul, gambar-gambar tersebut dimanipulasi untuk mempersiapkan tahap pemrosesan lebih lanjut seperti yang dapat dilihat pada gambar 3.4. Proses ini memanfaatkan pustaka OpenCV, yang menyediakan berbagai fungsi pemrosesan citra secara efisien. Tahapan manipulasi gambar dijelaskan sebagai berikut:

### Grayscalling

Tahap pertama adalah konversi gambar ke skala abu-abu (*grayscale*). Tujuannya adalah untuk mengurangi kompleksitas gambar dengan menyederhanakan informasi warna. Hal ini mempermudah segmentasi antara jalur jalan dan objek lainnya (perhatikan gambar 3.5).



**Gambar 3.5** Konversi gambar ke *grayscale* mengurangi   
kompleksitas citra

Setiap piksel RGB dikonversi menjadi nilai intensitas Tunggal (0-255) dengan rata-rata piksel pada tiga kanal warna (R, G, B). Untuk setiap piksel pada masing-masing *channel* akan dikenai algoritma *grayscaling* dengan menghitung rata-rata piksel. Contoh pada gambar 3x4 yang direpresentasikan pada gambar 3.6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **255** |  |  |
|  | **100** |  |  |  |
| **255** |  |  |  |  |
|  |  |  |  | **255** |
|  |  |  | **255** |  |
|  |  | **100** |  |  |

**Gambar 3.6** Representasi RGB pada gambar 3x4

Setelah dikenai *grayscalling* akan menghasilkan gambar *single channel* yang terdiri dari piksel dengan intensitasi 0-255 dan menghasilkan gambar 3.7.

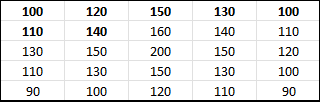
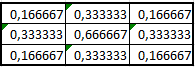
|  |  |  |
| --- | --- | --- |
| **203** |  |  |
|  |  |  |
|  |  |  |
|  |  | **203** |

**Gambar 3.7** Representasi grayscale pada   
gambar 3x4

Gambar 3.7 memiliki jumlah piksel yang jauh lebih sedikit dan memiliki kompleksitas gambar yang jauh lebih rendah jika dibandingkan dengan gambar berwarna (gambar 3.6). Oleh karena itu. *Grayscalling* akan mengurangi komputasi simulasi dengan cukup signifikan

### Noise Reduction

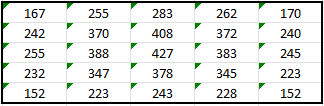
*Noise* adalah piksel-piksel dengan intensitas yang berbeda secara signifikan dibandingkan dengan lingkungan sekitarnya, yang dapat menyebabkan kesalah deteksi tepi (*false edge*). Gambar 3.8merupakan contoh gambar 5x5 pixel dengan distribusi piksel yang tidak seimbang (*noise*).

**** 

**Gambar 3.8** Noise pada gambar 5x5 (kiri) dan 3x3   
(kanan) Gaussian kernel

Salah satu metode yang digunakan untuk mengurangi *noise* adalah menggunakan *gaussian blur* pada gambar 3.8(kanan)yang berfungsi untuk menghaluskan perubahan intensitas piksel dengan rata-rata bergantung pada area sekitar piksel target.

Setelah melakukan operasi konvolusi, yaitu melakukan perkalian kernel Gausian terhadap gambar 3.8, maka hasil yang didapatkan berupa gambar yang memiliki distribusi piksel yang jauh lebih seimbang yang dapat dilihat pada gambar 3.9.



**Gambar 3.9** Hasil gambar 3.6 setelah menggunakan gaussian blur

Dengan menerapkan operasi Gaussian blur pada gambar 3.5, hasil yang didapatkan berupa gambar yang jauh lebih bagus serta minimnya *noise* pada gambar sehinga nantinya akan memudahkan melakukan operasi deteksi tepi.



**Gambar 3.10**  Pengaplikasian *blurring* pada gambar **3.5**

Penggunaan *blur* mampu membuat distribusi piksel yang jauh lebih seimbang sehingga memudahkan algoritma seperti deteksi tepi *Canny* untuk mendeteksi perubahan intensitas piksel.

### Canny Edge Detection

Tahap ini menggunakan algoritma *Canny edge detection,* yang merupakan salah satu metode deteksi tepi yang paling populer dan efektif. Tepi didefinisikan sebagai area dengan perubahan intensitas yang signifikan. Algoritma ini menggunakan gradien intensitas untuk mendeteksi tepi secara persisi.

1. Gradien Calculation

Algoritma menghitung gradien menggunakan operator Sobel untuk menentukan perubahan intensitas pada gambar yang terdiri dari matriks 3x3 yang dapat dilihat pada gambar 3.11 (deteksi tepi horizontal) dan 3.12 (deteksi tepi vertikal).

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

**Gambar 3.11** Sobel filter horizontal (x)

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

**Gambar 3.12** Sobel filter vertikal (y)

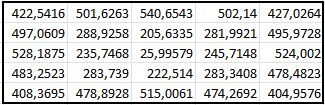
Untuk setiap pixel akan dihitung:

* *Gradient magnitude* mengidentifikasi tingkat atau kedalaman perubahan intensitas dalam gambar.



**Gambar 3.13** Aplikasi *gradient magnitude* (sumber: [*Understanding and Implementing Canny Edge Detection in Native Python | by Pasan Kalansooriya | Medium*](https://medium.com/@pasanSK/understanding-and-implementing-canny-edge-detection-in-native-python-52c296255601)

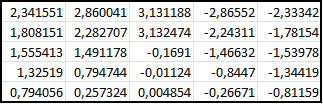
Gambar 3.13menunjukkan aplikasi *gradient magnitude* pada sebuah gambar, yang membantu mendeteksi perubahan intensitas pada piksel secara signifikan. *Gradient magnitude* merupakan komponen penting dalam proses deteksi tepi, karena memberikan informasi mengenai kekuatan perubahan intensitas.



**Gambar 3.14** Hasil kalkulasi *gradient magnitude*

Gambar 3.14di atas ini menunjukkan hasil *kalkulasi gradient calculation* pada gambar di atas. Setiap nilai dalam tabel merepresentasikan tingkat perubahan intensitas pada piksel terkait. Nilai tinggi menunjukkan adanya perubahan intensitas yang tajam, sedangkan nilai rendah mengindikasikan perubahan yang lebih halus.

* *Gradient direction* untuk menentukan arah perubahan intensitas.

**

**Gambar 3.15** Hasil kalkulasi *gradient direction*

Gambar 3.15di atas ini menunjukkan hasil kalkulasi gradient *direction* pada gambar di atas. Setiap nilai dalam tabel merepresentasikan orientasi (vertikal maupun horizontal) tepi yang terdeteksi dalam gambar 3.8.

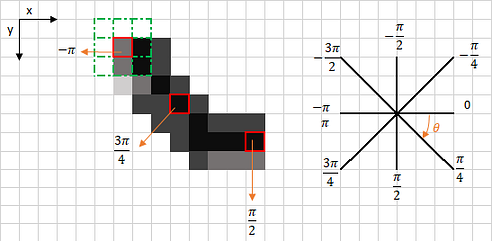
1. Non-Maximum Suppression (NMS)

Operasi NMS mempertahankan piksel dengan gradien tertinggi di sekitar suatu area, sehingga garis tepi menjadi lebih tipis dan presisi.



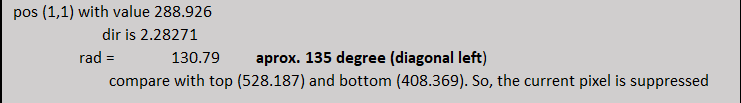
**Gambar 3.16** Aplikasi NMS pada hasil *gradient calculation* (sumber: [Understanding and Implementing Canny Edge Detection in Native Python | by Pasan Kalansooriya | Medium](https://medium.com/@pasanSK/understanding-and-implementing-canny-edge-detection-in-native-python-52c296255601))

Berdasarkan *gradient magnitude* dan *gradient direction* yang diperoleh, setiap piksel akan dibandingkan dengan dua piksel terdekat sesuai arah gradiennya berdasarkan yang dapat dilihat pada gambar 3.17. Hasilnya adalah tepi yang lebih bersih dan akurat.



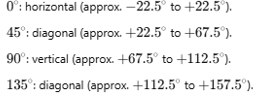
**Gambar 3.17** Non-Maximum Suppression

Berdasarkan *gradient magnitude* dan *gradient direction* yang diperoleh pada gambar 3.14 dan gambar 3.15, operasi NMS berlaku secara lokal dimana, setiap piksel akan dibandingkan dengan dua piksel terdekat berdasarkan nilai *radiance.*



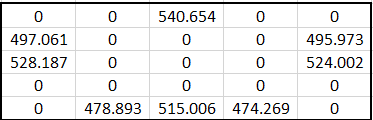
**Gambar 3.18** Contoh NMS pada piksel (1,1)

Untuk menentukan piksel tetangga yang akan dikenai perbandingan, bulatkan *angle* ke 45º terdekat dapat dilihat pada gambar 3.19.



**Gambar 3.19** Empatarah gradien (sumber: **(**[**Sobel Operator Gradient Calculation**](https://chatgpt.com/c/67581755-6f98-800f-84b8-cead2050ba22)**)**

Maka, tepi akhir yang didapatkan dapat dilihat pada gambar 3.20:



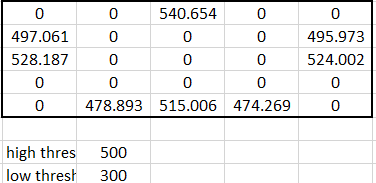
**Gambar 3.20** Hasil operasi NMS pada

Setiap nilai pada gambar 3.20 merupakan intensitas dari piksel pada gambar 3.8. Semakin tinggi nilai intensitas pada suatu koordinat, maka semakin jelas perbedaan intensitas piksel pada koordinat tersebut jika dibandingkan dengan piksel tetangganya.

1. Double Thresholding

Dua ambang batas (*upper* dan *lower*) digunakan untuk mengidentifikasi tepi kuat, tepi lemah dan tepi tidak relevan. Piksel tepi yang tidak memenuhi kriteria ambang akan dihilangkan.

Menggunakan gambar 3.20, *double thresholding* dapat diterapkan dengan menetapkan dua nilai ambang atas dan bawah (perhatikan gambar 3.21).



**Gambar 3.21** Double Thresholding

Untuk setiap piksel pada gambar 3.21 yang memiliki intensitas melebihi batas atas (500) dan tidak kurang dari batas bawah (300) akan dianggap sebagai tepi. Nilai yang tidak memenuhi ambang batas akan dianggap tepi yang tidak relevan (*weak edges*).



**Gambar 3.22** Pemetaan NMS pada gambar asli

Hasil yang didapatkan pada gambar 3.22 merupakan klasifikasi gradient berdasarkan nilai *threshold*. Untuk setiap piksel yang tidak nol akan dipetakan kembali pada gambar aslinya.

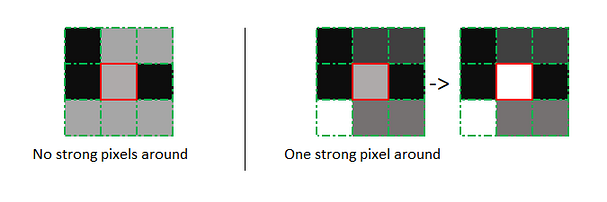


**Gambar 3.23** Contoh Aplikasi *double thresholding*

Gambar 3.23 adalah contoh aplikasi *double threshold* pada sebuah gambar hasil perhitungan NMS (kiri) dan hasil eliminasi tepi tidak relevan (kanan).

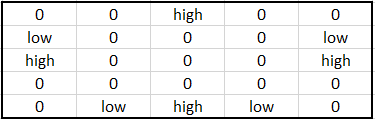
1. Edge tracking by Hysteresis

Hasil dari *thresholding* akan diproses lebih lanjut jika dan hanya jika terdapat minimal satu piksel yang kuat di sekitar piksel yang sedang di proses.



**Gambar 3.24** Eliminasi tepi yang lemah (sumber: [**Canny Edge Detection Step by Step in Python — Computer Vision | by Sofiane Sahir | Towards Data Science**](https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123))

Tahapan ini akan menghasilkan representasi yang hanya menunjukkan tepi-tepi signifikan dari jalur jalan.

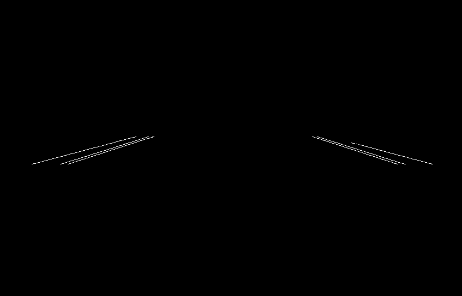
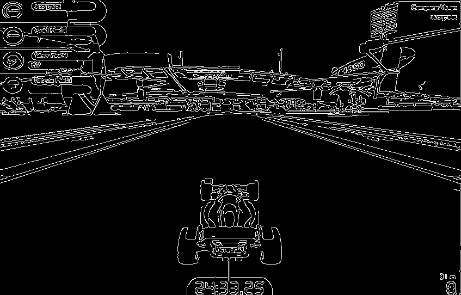


**Gambar 3.25** Distribusi tepi lemah dan kuat

Gambar 3.25 merepresentasikan distribusi tepi yang kuat dan lemah pada gambar 3.22. Informasi ini nantinya dapat digunakan untuk mengambil keputusan apakah sebuah tepi pada koordinat tertentu relevan atau tidak.

### Region of Interest

*Region of Interest* (RoI) digunakan untuk mengisolasi area jalan dari objek lain pada gambar. Proses ini mencakup menentukan area berbentuk *trapezoid* atau poligon yang mencakup jalur jalan dan mengabaikan piksel di luar area ini.

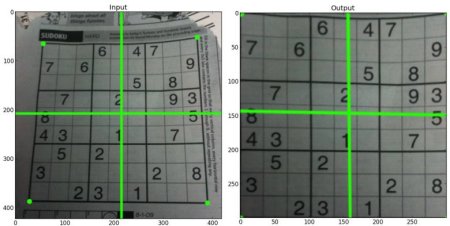
1. 

**Gambar 3.26** Implementasi RoI pada hasil Canny edge detection

Isolasi area jalan membantu meningkatkan akurasi proses berikutnya, seperti pendeteksian garis pada *sliding window*. Dengan hanya memproses jalur jalan, sistem dapat fokus pada objek yang benar-benar penting untuk pengemudian otomatis.

### Perspective Transform

Transformasi perspektif (*perpective transform*) adalah jenis transformasi geometri yang memetakan titik-titik dari satu perspektif (misalnya, tampilan kamera menghadap ke depan) ke perspektif lain (misalnya, tampilan bird's-eye view atau tampilan dari atas). Transformasi ini direpresentasikan oleh matriks transformasi 3x3 dan sering dihitung menggunakan homografi (perhatikan contoh gambar 3.27**)**.

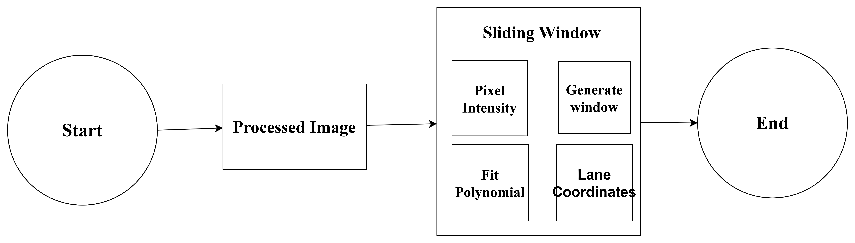


**Gambar 3.27** Transformasi RoI menjadi *Bird’s eye view* (sumber: [OpenCV: Geometric Transformations of Images](https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html))

Dalam deteksi jalur, kamera yang menghadap ke depan menangkap jalan dari sudut pandang yang membuat garis-garis jalur terlihat menyatu di kejauhan (akibat sudut pandang kamera). Transformasi perspektif dapat:

1. Menghilangkan distorsi perspektif: Mengubah gambar menjadi tampilan *bird's-eye view*, di mana garis-garis jalur yang sejajar terlihat paralel.
2. Menyederhanakan deteksi jalur: Memudahkan pendeteksian garis-garis jalur.
3. Meningkatkan akurasi: Memberikan tampilan yang lebih jelas tentang tata letak jalan untuk pemrosesan lebih lanjut.

## Lane Detection



**Gambar 3.28** Flowchart Sliding Window

Algoritma utama dalam simulasi ini adalah mendeteksi piksel pada permainan Trackmania yang mengindikasikan jalur. Informasi ini nantinya akan digunakan sebagai patokan untuk mengemudikan mobil sepanjang jalur jalan yang terdeteksi.

Adapun tahapan-tahapan yang dilakukan untuk mendeteksi jalur jalan yaitu melakukan mengirimkan gambar yang telah diproses secara *real-time* kedalam algoritma *Sliding Window* untuk melakukan ekstraksi jalur jalan.

### Sliding Window

Tahapan ini menggunakan algoritma Sliding Window untuk mendeteksi jalur jalan (*lane*) pada gambar yang sudah melalui proses RoI (*Region of Interest*). *Sliding Window* membantu menelusuri dan menentukan posisi garis secara bertahap dengan pendekatan *window-based*. *Sliding Window* membagi area gambar menjadi beberapa *window* kecil, yang kemudian dipindahkan (*slid*) secara bertahap sepanjang jalur jalan untuk mendeteksi garis.

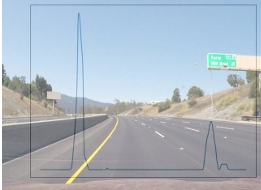
Tahapan penerapan sliding window untuk menentukan jalur jalan adalah sebagai berikut:

#### Histogram Intensitas Piksel

Histogram intensitas piksel digunakan untuk menentukan posisi awal garis. Bagian bawah gambar diambil untuk membuat histogram intensitas horizontal, yang menunjukkan distribusi intensitas piksel di sepanjang lebar gambar. Pada bagian puncak histogram menunjukkan area dengan konsentrasi intensitas tinggi, yang mengindikasikan lokasi jalur.

#### Deteksi Posisi Awal

Posisi awal garis ditentukan berdasarkan histogram intensitas piksel pada bagian bawah gambar. Puncak histogram menunjukkan area dengan intensitas piksel tinggi, yang sering kali merupakan garis jalan.

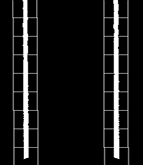


**Gambar 3.29** Histogram intensitas piksel untuk mengidentifikasi posisi awal lane (Sumber: [50220200616-82467-1kw0uel-libre.pdf](https://d1wqtxts1xzle7.cloudfront.net/63648751/50220200616-82467-1kw0uel-libre.pdf?1592341541=&response-content-disposition=inline%3B+filename%3DAnalysis_of_Lane_Detection_Techniques_on.pdf&Expires=1733722188&Signature=BfvdOFHw7xho5YJsNGUQinyjpAsZgr9gj5uHEBvVdFmdhgiRTn8JjFSjJkAKYDHeMCMN7rzZ~E1R3Vuk-FjPFmlCPzW9ihDHa-dB29tD9sBTQiS6KuN5ahdaQlWErdmwBCRvdq1cGNyLe3W177gr5b6cz5aieGBE1Iilu4djbiUZ2PfguFyoVr62rgnQ3AtedjQNyQdMehXlBYoiF0Mo6N8U7r4j-HNA8zvx-n9DK5AyrYi2Q4Ttd4aeYclUzUgY9d75ok2y73qPz8xjTbWe0WxDr3cWRXsjs~PWUJOKp0Jm5ybqlZYK3CdCig-4tOJoR4agiq2BUReUph1UeDXdUA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA), hal: 8)

Pada gambar 3.29, intensitas *lane* kiri dan kanan akan memiliki histogram yang jauh lebih tinggi dibandingkan piksel lainnya.

#### Pergeseran Window

*Window* digeser ke atas berdasarkan deteksi piksel yang termasuk dalam garis. Jika terdapat banyak piksel dalam sebuah *window* yang sesuai dengan jalur, posisi *window* diperbarui.



**Gambar 3.30** Sliding window pada hasil deteksi tepi

Pada kasus tertentu jika piksel jalan tidak ditemukan, posisi terakhir dianggap sebagai titik referensi untuk *window* berikutnya.

#### Menggambar Garis

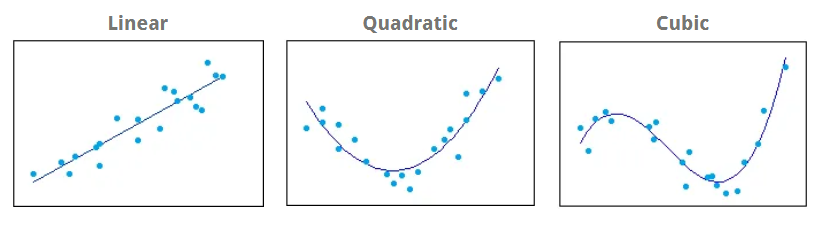
Garis jalan digambar berdasarkan posisi *window* yang terdeteksi pada setiap langkah berdasarkan koordinat yang diperoleh dari fungsi *polynominal curve*.

(7)

Dimana:

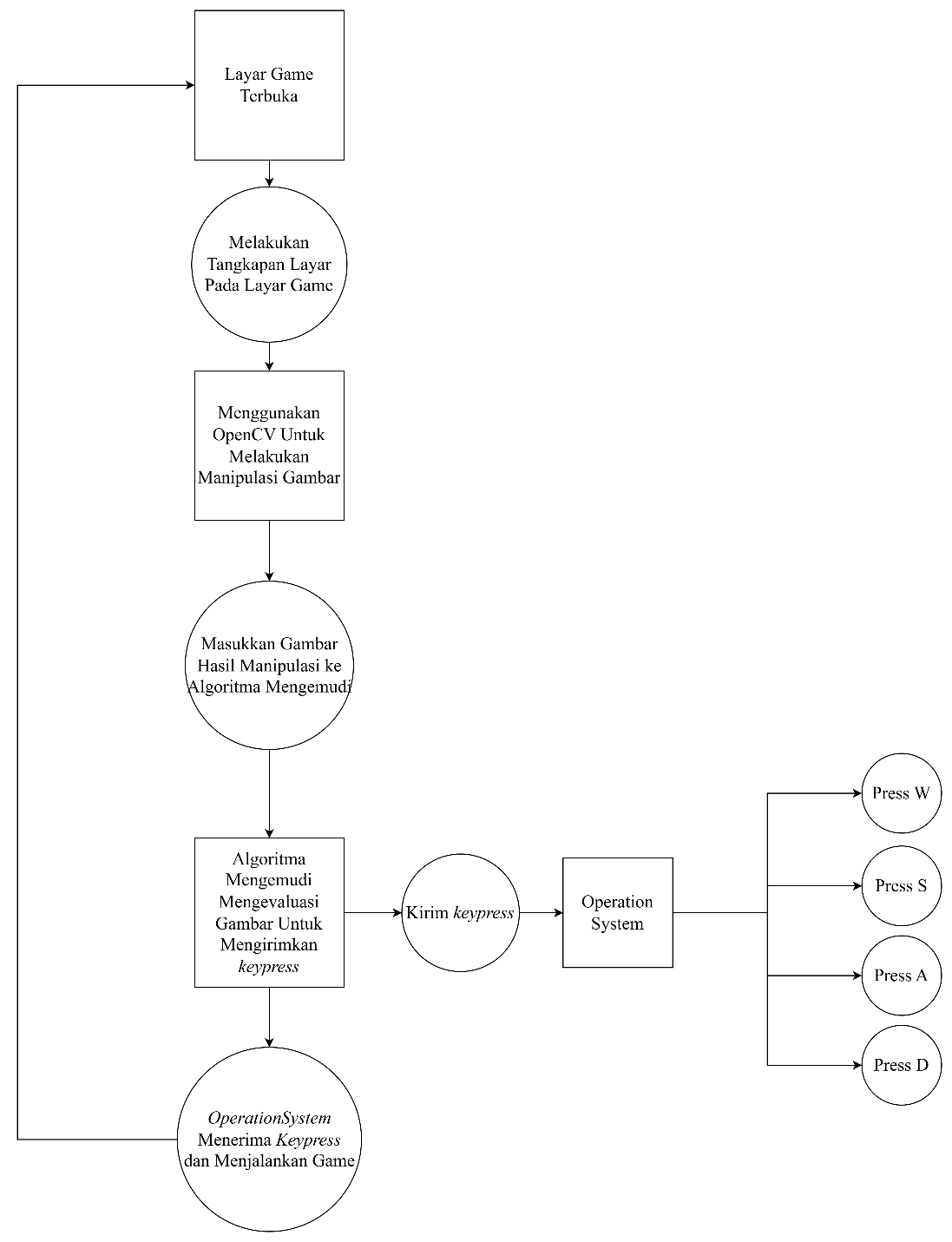
* : Mendeteksi *curvature* pada garis
* : Representasi *slope* dari garis
* : y*-intercept* (dimana garis dimulai pada y-axis pada bagian bawah gambar)

Sliding window akan mengembalikan nilai berupa koordinat x dan y untuk masing-masing garis (kiri dan kanan). Sliding window membantu melacak jalur jalan secara akurat meskipun terdapat lengkungan atau perubahan arah. Algoritma ini juga efektif dalam mengurangi *noise* karena hanya fokus pada area dalam *window*.



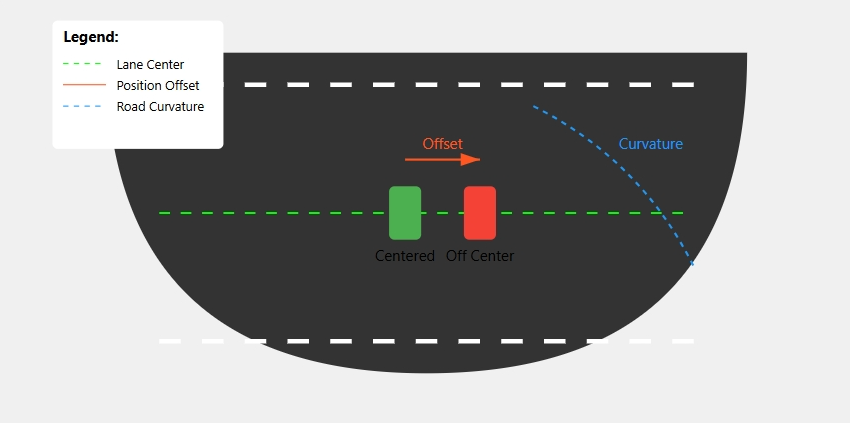
**Gambar 3.31** Polynomial linear (kiri) dan polynomial kuadrat (kanan) (Sumber: <https://sklc-tinymce-2021.s3.amazonaws.com/2020/09/mceclip1_1600436474.png>)

## Algoritma Mengemudi Berdasarkan Lane



**Gambar 3.32** Flowchart simulasi

Tahapan ini merupakan inti dari pengemudian otomatis berbasis jalur (*lane*). Algoritma ini menggunakan informasi dari jalur yang terdeteksi untuk menentukan aksi pengemudian dengan mengirimkan *keypress* berupa tombol-tombol pada *keyboard*.



**Gambar 3.33** Kontrol kemudi berdasarkan jalur

Algoritma akan dirancang untuk menavigasi kendaraan dengan mempertimbangkan posisi jalur relatif terhadap mobil. Jalur digunakan sebagai panduan untuk menentukan arah dan kecepatan pengemudian.

Tahapan ini meliputi:

1. Mengidentifikasi Posisi Jalur

Posisi jalur relatif terhadap kendaraan dihitung berdasarkan koordinat hasil Sliding Window dengan menghitung rata-rata kurvatur. Sistem menggunakan pendekatan kuantitatif untuk menerjemahkan posisi jalur:

1. *Off-center* ke kanan (nilai *offset* positif)

Jika mobil mendekati jalur bagian kanan maka belok ke kiri.

1. *Off-center* ke kiri (nilai *offset* negatif)

Jika mobil mendekati jalur bagian kiri maka belok ke kanan.

1. Mendekati jalur lurus (rata-rata kurvatur = 0):

* Mobil bergerak lurus
* Minimal koreksi kemudi
* Mengirim *key press* tombol W pada keyboard.

1. Mendekati belok kiri (rata-rata kurvatur < 0)

* Belok ringan ke kiri (sudut kecil)
* Koreksi gradual untuk menjaga posisi di tengah jalur
* Mengirim *key press* tombol A pada keyboard.

1. Mendekati belok kanan (rata-rata kurvatur > 0):

* Belok ringan ke kanan (sudut kecil)
* Koreksi gradual untuk menjaga posisi di tengah jalur
* Mengirim *key press* tombol D pada keyboard.

1. Kurvatur ekstrem: Jika melebihi nilai *threshold* yang ditetapkan, mobil diarahkan untuk belok tajam dengan penyesuaian kecepatan untuk keamanan.
2. Logika Pengambilan Keputusan

Algoritma mengimplementasikan hierarki keputusan berbasis prioritas:

1. Prioritas Utama: Menjaga mobil di dalam jalur
2. Prioritas Kedua: Meminimalisasi gerakan kemudi
3. Prioritas Ketiga: Mempertahankan kecepatan stabil
4. Eksekusi dengan PyAutoGUI dan PyDirectInput

Pustaka Python seperti PyAutoGUI dan PyDirectInput digunakan untk mengirim input kontrol ke game secara otomatis. Algoritma mengirimkan perintah berupa *keypress* berdasarkan jalur sebagai berikut:

1. Mengirim perintah keyboard
2. Mensimulasikan input *steering*
3. Mengontrol akselerasi

Dengan pendekatan ini, kendaraan dapat bergerak secara mandiri sesuai dengan jalur yang terdeteksi, baik pada jalur lurus maupun melengkung. Implementasi ini memastikan navigasi yang presisi dengan penyesuaian real-time terhadap kondisi jalur.

## Evaluasi dan Analisis Matrik

Evaluasi kinerja algoritma dilakukan berdasarkan matriks yang telah disebutkan pada Bab 2. Matriks ini mencakup pengukuran kualitatid dan kuantitaf dari hasil deteksi jalur serta performa navigasi kendaraan pada simulasi TrackMania.

### Matriks Kualitatif Lane Detection

#### Konsistensi Lane Detection

Konsistensi deteksi jalur dinilai dengan menghitung seberapa sering algoritma berhasil mendeteksi jalur secara kontinu dalam berbagai kondisi, termasuk tikungan, jalur lurus, dan area dengan *noise* tinggi. Evaluasi dilakukan dengan menganalisis jumlah *frame* di mana jalur terdeteksi dibandingkan dengan total *frame* dalam satu putaran simulasi.

#### Mean Square Error (MSE)

*Mean Square Error* digunakan untuk mengukur akurasi koordinat jalur yang terdeteksi terhadap jalur *ground truth.* Nilai MSE dihitung menggunakan persamaan berikut:

Dimana:

* : Koordinat jalur yang terdeteksi
* : Koordinat jalur ground truth
* : Jumlah total koordinat yang dibandingkan

MSE yang lebih rendah menunjukkan bahwa jalur yang terdeteksi lebih akurat mendekati *ground truth*.

### Matriks Track-Level

#### Putaran Berhasil Ditempuh

Metode evaluasi ini menghitung jumlah putaran penuh yang berhasil ditempuh kendaraan tanpa keluar jalur atau mengalami tabrakan. Matriks ini digunakan untuk menilai stabilitas algoritma pada navigasi jalur.

#### Jarak Tempuh Sebelum Menabrak

Jarak tempuh dihitung sebagai total panjang jalur yang dilalui kendaraan sebelum terjadi tabrakan atau kendaraan keluar dari jalur. Evaluasi ini memberikan gambaran tentang efektivitas algoritma dalam menjaga kendaraan tetap pada jalur dalam kondisi simulasi yang berbeda.

#### Rasio Keberhasilan

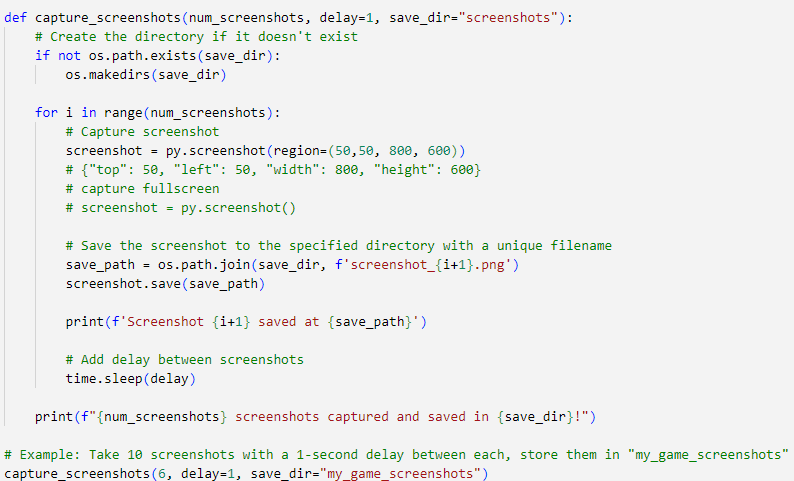
Rasio keberhasilan didefinisikan sebagai persentase dari total simulasi di mana kendaraan berhasil menyelesaikan putaran tanpa tabrakan atau keluar jalur. Rasio keberhasilan memberikan indikasi keseluruhan efektivitas algoritma pada berbagai skenario.

(add the parameter that is affect the lane detection)

# Hasil Simulasi Dan Pembahasan

## Pengumpulan Data

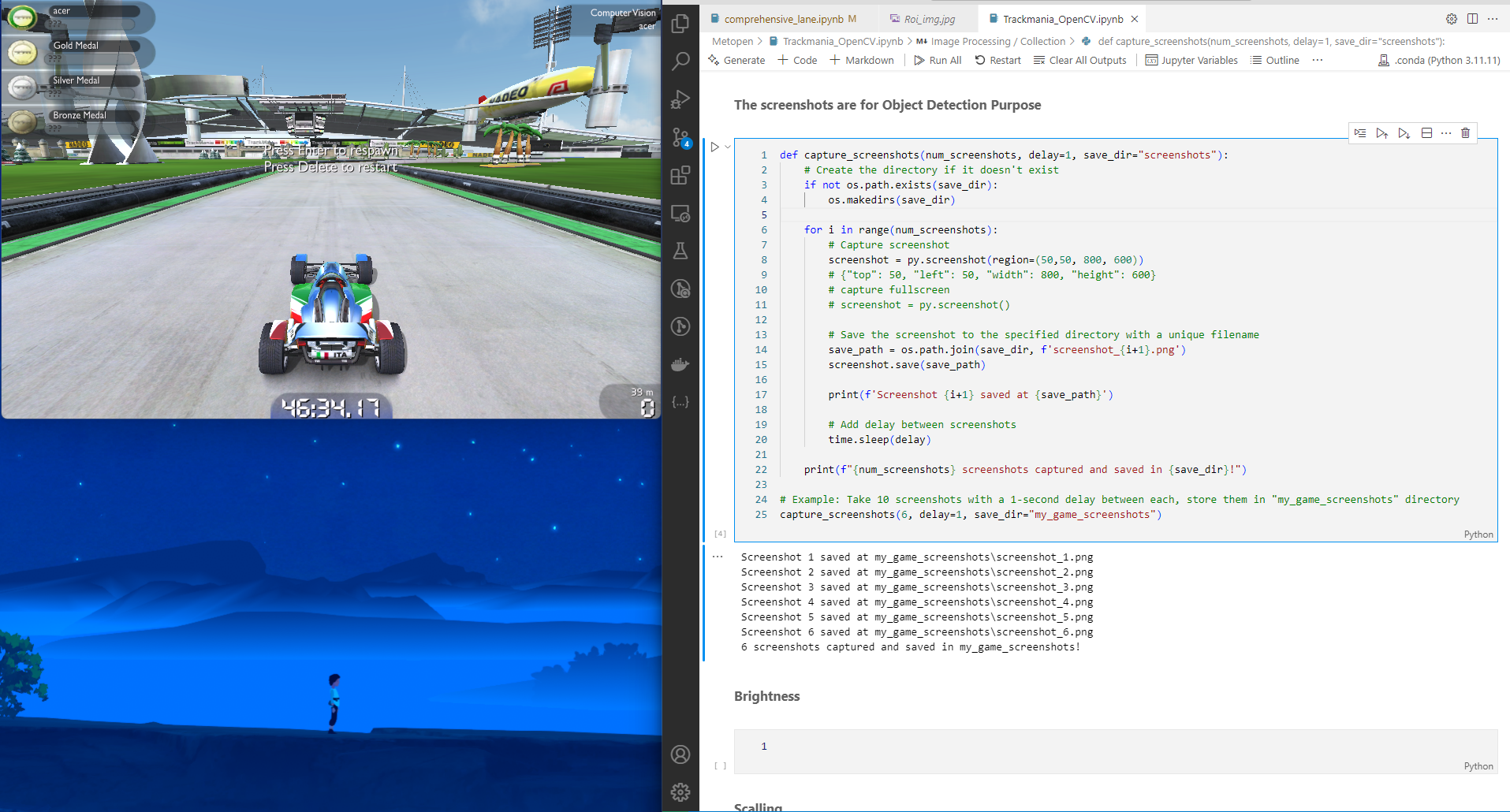
Data dikumpulkan dengan cara mengambil tangkapan layar pada *frame* permainan secara otomatis menggunakan pustaka pyAutoGUI yang bisa dilihat pada gambar 4.1.

**

**Gambar 4.1** Kode mengambil tangkapan layar pada   
posisi tertentu di monitor

Kode tersebut terlebih dahulu membuat sebuah folder bernama *screenshots* yang nantinya digunakan untuk menyimpan hasil tangkapan layar. Kode akan melakukan perulangan sebanyak parameter *num\_screenshots* yang merupakan jumlah tangkapan yang hendak diambil. Posisi gambar yang hendak diambil kemudian ditentukan berdasarkan letaknya di dalam layar monitor (1920 x 1080). Posisi ditentukan berdasarkan posisi gambar di sumbu x dan y, serta tinggi (600) dan lebar (800) dari gambar yang diambil.

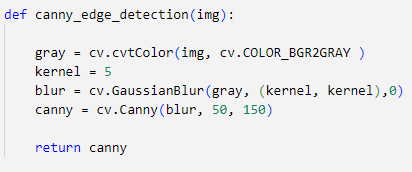
Kode ini akan dijalankan bersebelahan dengan layar permainan yang bisa dilihat pada gambar 4.2. Gambar yang telah ditangkap kemudian akan disimpan dengan ukuran yang telah ditentukan.



**Gambar 4.2** Proses penangkapan gambar dengan ukuran 800 x 600 pixel

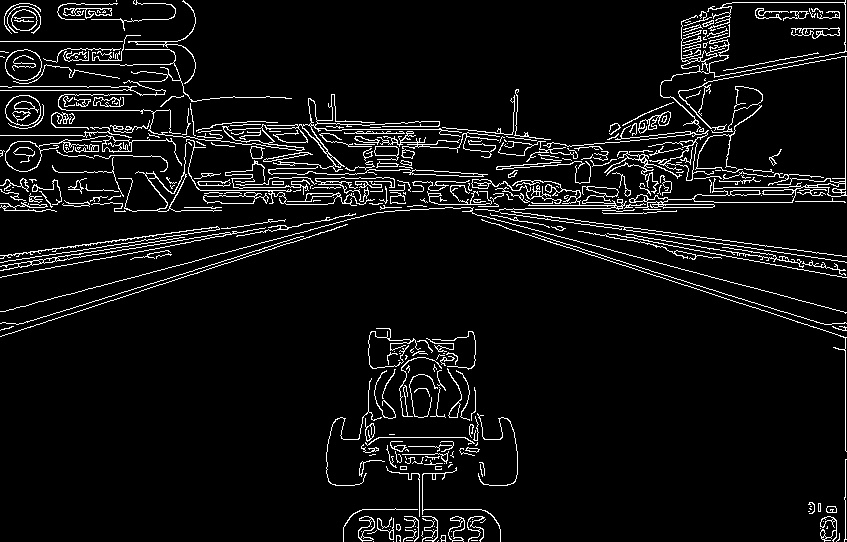
## Image Processing

Setelah gambar-gambar dikumpulkan, langkah selanjutnya adalah gambar-gambar tersebut dikenai operasi deteksi tepi Cannyselama simulasi berjalan secara *real-time.* Operasi deteksi tepi Canny membutuhkan gambar yang hanya terdiri dari 1 *channel* seperti abu-abu. Hal ini dikarenakan mendeteksi tepi hanya membutuhkan intensitas dari piksel. Kode untuk operasi deteksi tepi Canny dapat dilihat pada gambar 4.3.



**Gambar 4.3** Fungsi deteksi tepi *Canny* menggunakan OpenCV

**What is kernel and blur?**



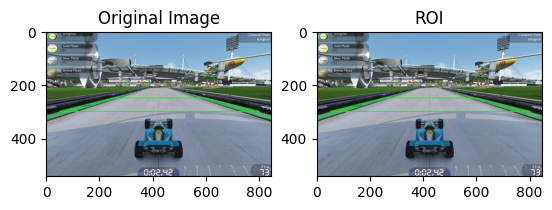
Pada tahap awal pengembangan sistem, deteksi tepi Canny digunakan untuk mengidentifikasi marka jalan. Proses ini melibatkan konversi citra ke skala abu-abu (grayscale) menggunakan cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY), diikuti dengan penerapan algoritma Canny dengan parameter ambang batas bawah dan atas (cv2.Canny(gray, 50, 150)). Untuk meningkatkan hasil deteksi, dilasi diterapkan menggunakan cv2.dilate(edges, None, iterations=1) guna mempertebal tepi yang terdeteksi. Namun, pendekatan ini menghasilkan banyak noise dan tidak dapat secara konsisten mendeteksi marka jalan berwarna hijau pada simulasi. Oleh karena itu, metode ini digantikan dengan pendekatan berbasis HSV (Hue, Saturation, Value) yang lebih spesifik untuk mendeteksi warna hijau, seperti yang akan dijelaskan pada bagian berikutnya.

### Region of Interest



Gambar 4.4 RoI pada gambar dengan ukuran 800x600

Gambar akan diisolasi sehingga hanya bagian jalan saja yang akan diproses dengan menentukan koordinat RoI pada gambar.



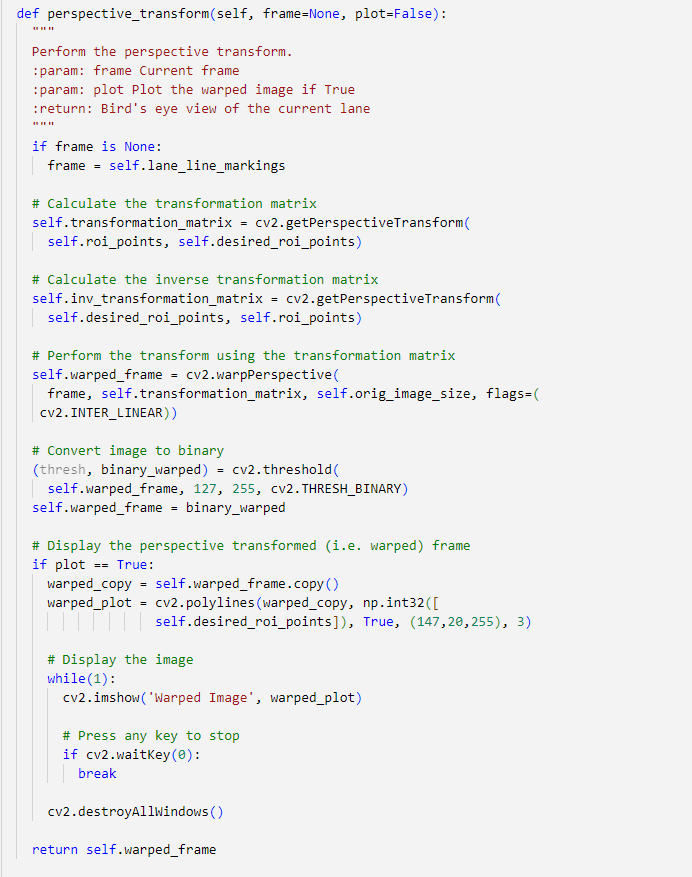
Region of Interest (ROI) didefinisikan untuk membatasi area pengolahan citra hanya pada bagian jalan yang relevan, sehingga mengurangi noise dari area di luar marka jalan. Awalnya, ROI ditetapkan untuk resolusi 800x600 dengan titik-titik berikut: (274, 254) (kiri atas), (517, 254) (kanan atas), (748, 313) (kanan bawah), dan (13, 313) (kiri bawah). Namun, ROI ini kurang optimal untuk menangkap marka jalan pada tikungan. Setelah iterasi, ROI diperlebar menjadi (200, 254) hingga (10, 313) untuk meningkatkan visibilitas marka jalan. Ketika resolusi citra dikurangi menjadi 400x300 untuk efisiensi, titik ROI disesuaikan menjadi (137, 127) (kiri atas), (258, 127) (kanan atas), (374, 156) (kanan bawah), dan (6, 156) (kiri bawah). Titik-titik ini digunakan untuk membentuk poligon yang digambar pada citra menggunakan cv2.polylines, memastikan hanya area jalan yang diproses.

### Perspective Wrapping

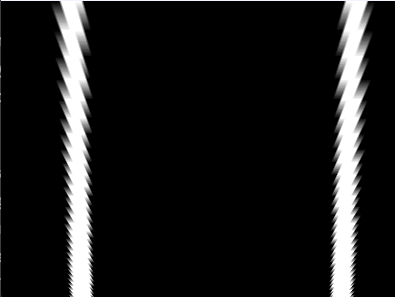
Transformasi perspektif diterapkan untuk mengubah pandangan citra dari perspektif kamera menjadi pandangan atas (*bird’s-eye view*), yang mempermudah deteksi marka jalan. Proses ini menggunakan fungsi cv2.warpPerspective dengan matriks transformasi yang dihitung dari *roi\_points* dan *desired\_roi\_points*. Untuk resolusi 400x300, *desired\_roi\_points* ditetapkan sebagai (50, 0) (kiri atas), (350, 0) (kanan atas), (350, 300) (kanan bawah), dan (50, 300) (kiri bawah), menghasilkan citra keluaran berukuran (400, 300).



Interpolasi cv2.INTER\_LINEAR digunakan untuk menjaga kualitas citra setelah transformasi. Langkah ini memastikan marka jalan sejajar dalam ruang yang dihasilkan, mempermudah perhitungan histogram dan deteksi *sliding window*.



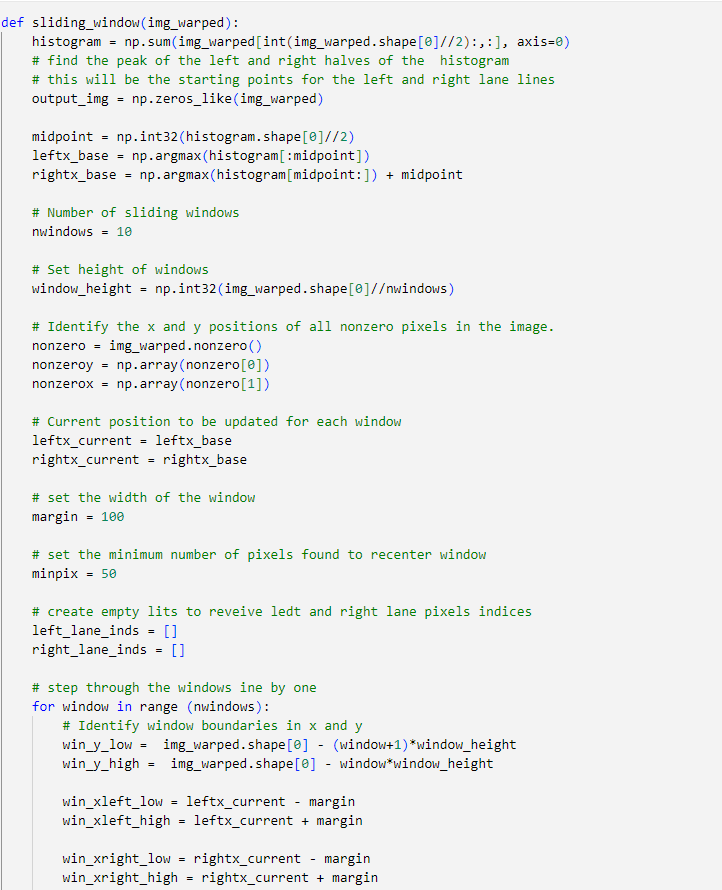
Berikut adalah hasil yang didapatkan

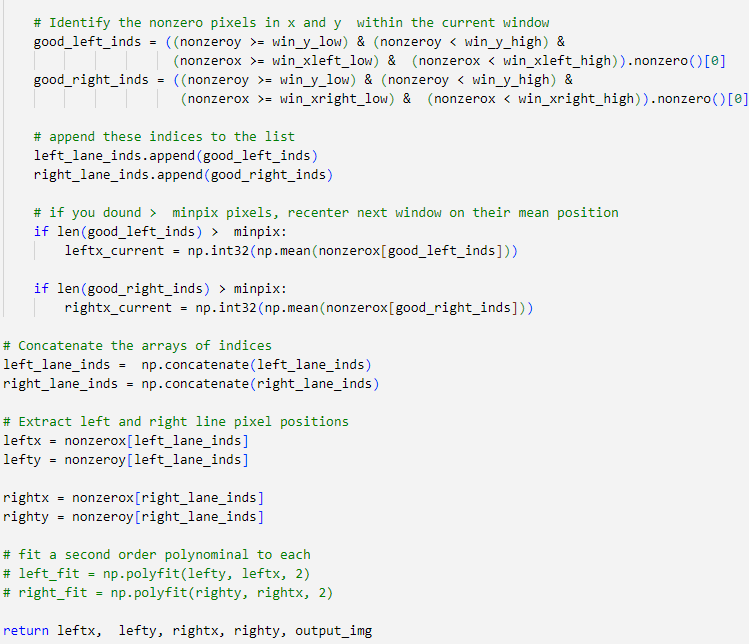


#### Optimasi Pipeline

Untuk meningkatkan efisiensi pengolahan, ukuran citra dikurangi dari 800x600 menjadi 400x300, mengurangi jumlah piksel yang diproses sebesar 75% (dari 480.000 piksel menjadi 120.000 piksel). Hal ini secara signifikan mempercepat operasi seperti konversi HSV dan transformasi perspektif. Selain itu, teknik skipping frames diterapkan dengan parameter skip\_frames = 4, sehingga hanya setiap frame ke-5 yang diproses. Pendekatan ini mengurangi beban komputasi sambil tetap mempertahankan stabilitas deteksi marka jalan, meskipun frame rate pada pipeline penuh tetap rendah (~1.9 FPS).

## Sliding Window



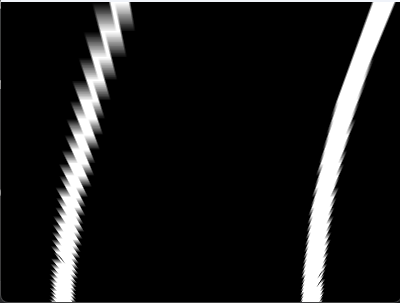


Metode *sliding window* digunakan untuk mendeteksi marka jalan secara akurat dengan fungsi *get\_lane\_line\_indices\_sliding\_windows*. Proses ini dimulai dengan menghitung histogram intensitas piksel pada citra yang telah di-*warp* (*calculate\_histogram*), diikuti dengan identifikasi posisi awal marka jalan kiri dan kanan. Jendela geser kemudian digunakan untuk melacak marka jalan secara vertikal, menghasilkan koefisien *polinomial* (*left\_fit* dan *right\_fit*) untuk setiap sisi. Untuk menjaga stabilitas deteksi antar frame, smoothing diterapkan: *left\_fit = 0.8 \* prev\_left\_fit + 0.2 \* left\_fit*. Fungsi tambahan *get\_lane\_line\_previous\_window* digunakan untuk memperbarui posisi jendela berdasarkan deteksi sebelumnya, memastikan kontinuitas pada tikungan.

### Debugging dan Visualisasi

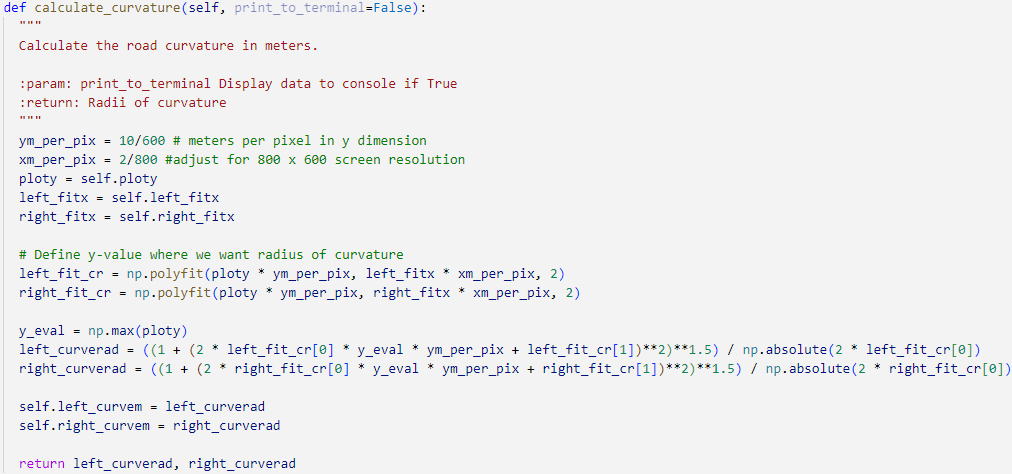
Selama pengembangan, visualisasi digunakan untuk memverifikasi hasil deteksi marka jalan. Citra-citra seperti Lane Detection, ROI Image, dan Warped Frame ditampilkan menggunakan cv2.imshow. Citra Warped Frame menunjukkan marka jalan yang telah disejajarkan, sedangkan ROI Image menampilkan poligon ROI yang digambar pada citra asli. Visualisasi ini membantu mengidentifikasi masalah seperti ROI yang terlalu sempit atau deteksi marka yang tidak akurat, sebelum akhirnya dinonaktifkan untuk meningkatkan frame rate.







## Menghitung Kurva Jalan



Kurva jalan dihitung menggunakan rumus radius kelengkungan:

*((1 + (2 \* fit[0] \* y\_eval + fit[1]) \*\* 2) \*\* 1.5) / np.abs(2 \* fit[0])*

Awalnya, perhitungan ini menghasilkan nilai Radius: 0.0 m karena koefisien *polinomial* yang tidak valid. Setelah memperbaiki deteksi marka jalan, koefisien polinomial menjadi akurat, dan konversi piksel ke meter diterapkan:

(XM\_PER\_PIX = 2 / 400, YM\_PER\_PIX = 10 / 300)

Evaluasi dilakukan pada posisi bawah citra (*y\_eval = ploty[-1]*), menghasilkan nilai radius yang realistis, seperti ~500 m pada jalan lurus dan <300 m pada tikungan.

##### Jalan Lurus



##### Belokan tajam



##### Belokan Ringan



## Menghitung Offset Mobil



*Offset* mobil dihitung untuk menentukan posisi relatif mobil terhadap pusat jalur menggunakan fungsi *calculate\_car\_position*. Pusat jalur dihitung sebagai berikut:

*lane\_center = (self.left\_fitx[-1] + self.right\_fitx[-1]) / 2*

*Lane\_center* dibandingkan dengan pusat mobil (*car\_center = warped\_frame.shape[1] / 2*). Offset dikonversi ke sentimeter menggunakan XM\_PER\_PIX, dengan smoothing diterapkan: smoothed\_offset = 0.8 \* last\_center\_offset + 0.2 \* center\_offset. Hasilnya menunjukkan offset yang bervariasi, misalnya -10.0 cm saat mobil berada di sebelah kiri jalur, yang kemudian digunakan untuk menghitung nilai kemudi.

##### Mobil berada di Tengah jalan



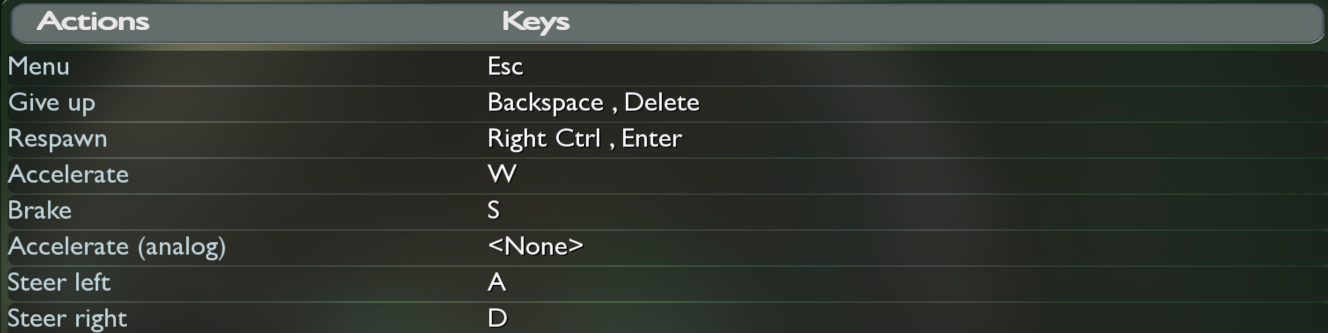
##### Mobil di pinggir kiri jalan

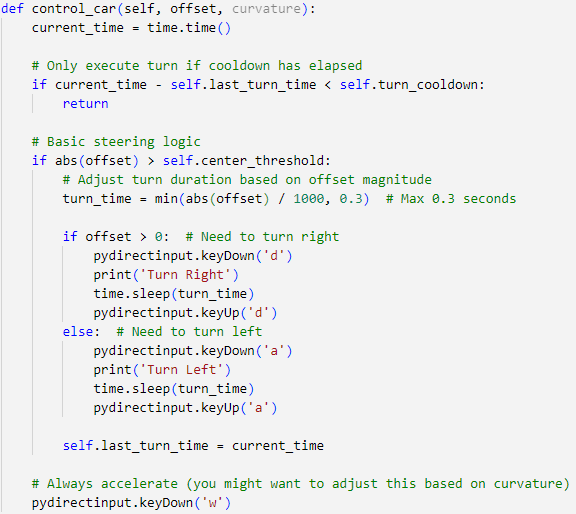


##### Mobil berada di pinggir kanan jalan



## Algoritma Mengemudi





Algoritma mengemudi dikembangkan untuk mengontrol kemudi dan kecepatan mobil berdasarkan offset dan radius kelengkungan. Fungsi control\_steering menghitung nilai kemudi mentah (raw\_steering = center\_offset / 100.0 \* steering\_sensitivity), dengan smoothing: steering\_value = 0.9 \* last\_steering + 0.1 \* raw\_steering. Penyesuaian untuk tikungan diterapkan dengan faktor curve\_factor \* 0.2. Fungsi control\_speed mengurangi kecepatan pada tikungan (speed\_factor = min(1.0, curve\_radius / 300) \*\* 2), menghasilkan kecepatan seperti 0.05 pada tikungan tajam. Kontrol diterapkan menggunakan pydirectinput dengan press\_duration = min(0.05, abs(steering\_value) \* 0.3), memungkinkan mobil berbelok ke kiri (keyDown('a')) atau kanan (keyDown('d')).

## Parameter Tunning (Penyesuaian Parameter)

Penyesuaian parameter dilakukan untuk mengoptimalkan performa sistem. Parameter steering\_sensitivity awalnya diatur ke 0.7, tetapi diturunkan menjadi 0.4 untuk mengurangi overcorrecting. control\_delay dikurangi menjadi 0.05 untuk responsivitas yang lebih baik, dan press\_duration multiplier disesuaikan menjadi 0.3. Faktor penyesuaian tikungan dikurangi menjadi 0.2, dan speed\_control ditetapkan pada 0.1. Smoothing diterapkan pada offset dan kemudi untuk stabilitas, memastikan mobil dapat berbelok dengan lebih halus.

**Kenapa memilih ukuran 800 x 600 ?**

**Kenapa tidak memilih ukuran 1920 x 1080 atau yang lainnya?**

**Kenapa memilih roi tersebut**

**Kenapa memilih destination roi tersebut**

**Kenapa perlu menghitung kurvatur**

Tabel 4.1 Parameter ukuran gambar

|  |  |  |
| --- | --- | --- |
| **Ukuran Gambar** | **FPS** | **Performa Kemudi** |
| **400 x 300** |  |  |
| **800 x 600** |  |  |
| **1920 x 1080** |  |  |

Tabel 4.2 Parameter RoI

|  |  |  |
| --- | --- | --- |
| **ROI** | **Destination ROI** | **Performa Lane** |
|  |  |  |
|  |  |  |
|  |  |  |

**Tabel 4.3** Konversi Warna

|  |  |
| --- | --- |
| **Konversi Warna** | **Performa** |
| **Grasyscale** |  |
| **HSV** |  |

**Tabel 4.4** Kecepatan mobil

|  |  |
| --- | --- |
| **Kecepatan** | **Performa** |
|  |  |
|  |  |

## Hasil dan Analisis

 Quantitative Results: FPS, steering values, offset, curvature radius.

 Qualitative Observations: Turning behavior, overcorrecting issues, and improvements over time.

 Visual Evidence: Descriptions of debug outputs (e.g., warped frames, ROI images).

Mengatur ROI sangat berpengaruh

Kecepatan mempengaruhi perspective game

Ukuran sliding window mempengaruhi kualitas deteksi

Huh, a curve radius of around 100 on a straight road? That’s a head-scratcher—means the algorithm’s seeing *something* curvy even when it shouldn’t.

### Matriks Kualitatif Lane Detection

### Matriks Track-Level

##### Putaran Berhasil Ditempuh

|  |  |  |
| --- | --- | --- |
| Track | Jumlah Belokan | Belokan Berhasil Dilalui |
| 1 | 4 |  |
| 2 | 3 |  |
| 3 | 2 |  |

### Jarak Tempuh Sebelum Menabrak

|  |  |  |
| --- | --- | --- |
| Track |  |  |
| 1 | 4 |  |
| 2 | 3 |  |
| 3 | 2 |  |

### Rasio Keberhasilan

#### MSE

### Analisis Frame Rate

## Tantangan dan Solusi

1. **Tantangan: Deteksi Marka Jalan yang Rusak**

Solusi: Beralih dari deteksi tepi Canny ke pendekatan HSV untuk menargetkan marka jalan hijau, meningkatkan akurasi deteksi.

1. **Tantangan: Frame Rate Rendah**

Solusi: Mengurangi ukuran citra, menerapkan skipping frames, dan mengidentifikasi potensi pipeline standalone yang mencapai 20-30 FPS.

1. **Tantangan: Overcorrecting pada Kemudi**

Solusi: Menambahkan smoothing pada offset dan kemudi, menurunkan steering\_sensitivity, dan menyesuaikan faktor penyesuaian tikungan.

* Key Challenges: Broken detection, low FPS, overcorrecting.
* Innovative Solutions: HSV detection, image downscaling, smoothing techniques.

## Lampiran

# Daftar pustaka

[1] S. G. Klauer, F. Guo, B. G. Simons-Morton, M. C. Ouimet, S. E. Lee, and T. A. Dingus, “Distracted Driving and Risk of Road Crashes among Novice and Experienced Drivers,” *New England Journal of Medicine*, vol. 370, no. 1, pp. 54–59, Jan. 2014, doi: 10.1056/nejmsa1204142.

[2] Y. Pradrityarahman, D. I. Hestiwi, F. Al-Mustaqim, and M. L. Hakim, “Prototype Smart Autonomous Car berbasis Deep Learning dengan Sistem Pencegah Kecelakaan,” Nov. 2021. [Online]. Available: https://journal.uny.ac.id/index.php/jee

[3] S. G. Klauer, F. Guo, B. G. Simons-Morton, M. Claude Ouimet, S. E. Lee, and T. A. Dingus, “The Royal Society for the Prevention of Accidents Road Safety Factsheet,” Jan. 2014. [Online]. Available: www.rospa.com

[4] C. Badue *et al.*, “Self-driving cars: A survey,” Mar. 01, 2021, *Elsevier Ltd*. doi: 10.1016/j.eswa.2020.113816.

[5] X. Pan, Y. You, Z. Wang, and C. Lu, “Virtual to Real Reinforcement Learning for Autonomous Driving,” 2017.

[6] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep Reinforcement Learning framework for Autonomous Driv-ing.”

[7] L. Liu *et al.*, “Computing Systems for Autonomous Driving: State of the Art and Challenges,” *IEEE Internet Things J*, vol. 8, no. 8, pp. 6469–6486, Apr. 2021, doi: 10.1109/JIOT.2020.3043716.

[8] F. N. Nezami, M. A. Wächter, G. Pipa, and P. König, “Project Westdrive: Unity City With Self-Driving Cars and Pedestrians for Virtual Reality Studies,” *Frontiers in ICT*, vol. 7, Jan. 2020, doi: 10.3389/fict.2020.00001.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, “CARLA: An Open Urban Driving Simulator.”

[10] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, “A Survey on Simulators for Testing Self-Driving Cars,” Jan. 2021, [Online]. Available: http://arxiv.org/abs/2101.05337

[11] D. A. Pomerleau, “ALVINN: AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK.”

[12] M. R. Bonyadi, Z. Michalewicz, S. Nallaperuma, and F. Neumann, “IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES 1 Ahura: A heuristic-based racer for the open racing car simulator.” [Online]. Available: http://cig.dei.polimi.it/

[13] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving,” May 2015, [Online]. Available: http://arxiv.org/abs/1505.00256

[14] J.-B. Authier-Carcelen and R. Zadourian, “A Driving Model in the Realistic 3D Game Trackmania Using Deep Reinforcement Learning,” 2024, doi: 10.20944/preprints202409.0778.v1.

[15] C. Erdelyi, “Using Computer Vision Techniques to Play an Existing Video Game,” May 2019. Accessed: Oct. 11, 2024. [Online]. Available: https://scholarworks.calstate.edu/downloads/7w62f8544

[16] A. Punagin, “Analysis of Lane Detection Techniques on Structured Roads using OpenCV,” *Int J Res Appl Sci Eng Technol*, vol. 8, no. 5, pp. 2994–3003, May 2020, doi: 10.22214/ijraset.2020.5502.

[17] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, “Real-Time  Computer  Vision with  openCV,” *Commun ACM*, vol. 55, no. 6, pp. 61–69, Jun. 2012, doi: 10.1145/2184319.2184337.

[18] X. Yang and Z. Ling, “RESEARCH ON LANE DETECTION TECHNOLOGY BASED ON OPENCV,” 2015.

[19] A. S. Rathore, “Lane Detection for Autonomous Vehicles using OpenCV Library,” *International Research Journal of Engineering and Technology*, p. 1326, 2008, [Online]. Available: www.irjet.net

[20] L. Wang, M. Han, X. Li, N. Zhang, and H. Cheng, “Review of Classification Methods on Unbalanced Data Sets,” *IEEE Access*, vol. 9, pp. 64606–64628, 2021, doi: 10.1109/ACCESS.2021.3074243.

[21] T. S. Huang, “Computer Vision: Evolution and Promise.”

[22] M. Couprie, F. Nivando Bezerra, G. Bertrand, G. Bertrand Topological, and G. A. Bertrand Laboratoire, “operators for grayscale image processing,” 2001. [Online]. Available: www.esiee.fr/coupriem/Sdi

[23] R. C. . Gonzalez and R. E. . Woods, *Digital image processing*. Prentice Hall, 2002.

[24] V. S. Bottazzi, P. V. K. Borges, B. Stantic, and J. Jo, “Adaptive regions of interest based on HSV histograms for lane marks detection,” in *Advances in Intelligent Systems and Computing*, Springer Verlag, 2014, pp. 677–687. doi: 10.1007/978-3-319-05582-4\_58.

[25] P. Roy, S. Dutta, N. Dey, G. Dey, S. Chakraborty, and R. Ray, “Adaptive thresholding: A comparative study,” in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014*, Institute of Electrical and Electronics Engineers Inc., Dec. 2014, pp. 1182–1186. doi: 10.1109/ICCICCT.2014.6993140.

[26] H. G. Kaganami and Z. Beiji, “Region-based segmentation versus edge detection,” in *IIH-MSP 2009 - 2009 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2009, pp. 1217–1221. doi: 10.1109/IIH-MSP.2009.13.

[27] D. Kaur and Y. Kaur, “International Journal of Computer Science and Mobile Computing Various Image Segmentation Techniques: A Review,” 2014. [Online]. Available: www.ijcsmc.com

[28] J. Canny, “A VARIATIONAL APPROACH TO EDGE DETECTION,” 1983. [Online]. Available: www.aaai.org

[29] R. Szeliski, “Computer Vision: Algorithms and Applications”, Accessed: Mar. 12, 2025. [Online]. Available: https://docs.opencv.org/3.4/da/d6e/tutorial\_py\_geometric\_transformations.html

[30] J. Illingworth and J. Kittler, “SURVEY A Survey of the Hough Transform,” 1988.

[31] S. S., V. K. S., M. R. H. Z., S. K. K., D. S., and R. P. D., “Advanced Driver Assistant System,” *Indian Journal of Computer Science*, vol. 6, no. 3–4, p. 35, Aug. 2021, doi: 10.17010/ijcs/2021/v6/i3-4/165410.

[32] K. C. Bhupathi and H. Ferdowsi, “An Augmented Sliding Window Technique to Improve Detection of Curved Lanes in Autonomous Vehicles,” in *IEEE International Conference on Electro Information Technology*, IEEE Computer Society, Jul. 2020, pp. 522–527. doi: 10.1109/EIT48999.2020.9208278.

[33] D. Posch and J. Rask, “A model based approach to lane detec-tion and lane positioning using OpenCV,” Gothenburg Sweden, 2017.

[34] A. Sweigart, “PyAutoGUI Documentation,” 2021.