TUGAS AKHIR SEMESTER
Traffic Control System Based On Fuzzy Logic

Mata Kuliah: Sistem Kendali Lanjut
Dosen Pengampu: Ni Putu Devira Ayu Martini, S.Tr.T., M.Tr.T

Anggota Kelompok :

1.Marselinus Allen Nugraha (2310314001)
2.Muhammad Rafif Firjatullah (2310314009)
3.Mauli Andika Rahman (2310314027)

PROGRAM STUDI TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS PEMBANGUNAN NASIONAL VETERAN JAKARTA

2025

DAFTAR ISI

DAFTAR ISI 1
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 1
1.3 Tujuan 2
BAB II KAJIAN TEORI DAN IMPLEMENTASI SISTEM 3
2.1 Object Detection 3
2.1.1 Model : YOLO11 3
2.1.3 Proses Training 4
2.2 Wifi Communication 4
2.3 Sensor Ultrasonic 5
2.4 Fuzzy Logic 5
2.4.1 Fuzzy Set 5
2.4.2 Fuzzy Rule 7
2.5 Hardware & Software 8
BAB III PENGOLAHAN DATA 10
3.1. Pengukuran Dan Statistik 10
3.2 Analisis 11
BAB IV KESIMPULAN DAN SARAN 12
4.1 Kesimpulan 12
4.1 Saran 12
DAFTAR PUSTAKA 13
LAMPIRAN 14

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pertumbuhan jumlah kendaraan bermotor yang pesat di berbagai kota besar menyebabkan permasalahan lalu lintas menjadi semakin kompleks. Salah satu titik krusial yang sering menjadi sumber kemacetan adalah pada persimpangan jalan, di mana sistem pengaturan lalu lintas seperti lampu merah memainkan peran penting dalam menjaga kelancaran arus kendaraan. Namun, sistem lampu lalu lintas konvensional yang bekerja secara statis berdasarkan waktu yang telah ditentukan sering kali kurang responsif terhadap kondisi riil di lapangan, seperti panjang antrean kendaraan atau kepadatan lalu lintas.

Untuk menjawab tantangan tersebut, diperlukan sistem pengaturan lalu lintas yang cerdas dan adaptif. Dalam proyek ini, dikembangkan sebuah sistem lampu merah otomatis berbasis kamera dan sensor ultrasonik yang memanfaatkan logika fuzzy sebagai sistem pengambilan keputusan. Kamera digunakan untuk mendeteksi dan menghitung jumlah kendaraan yang mengantre di persimpangan, sementara sensor ultrasonik digunakan untuk mengukur panjang antrean berdasarkan jarak dari titik tertentu. Informasi yang diperoleh dari kedua sensor ini akan dianalisis menggunakan logika fuzzy untuk menentukan durasi lampu merah yang optimal dan menyesuaikan waktu nyala lampu secara real-time.

Dengan pendekatan ini, diharapkan sistem dapat memberikan solusi yang lebih efisien dan dinamis dalam mengatur lalu lintas, serta mengurangi waktu tunggu yang tidak perlu bagi pengendara, khususnya saat kondisi jalan tidak seimbang antara satu arah dan arah lainnya.

1.2 Rumusan Masalah

1. Bagaimana merancang sistem lampu merah yang dapat mendeteksi jumlah antrean kendaraan secara otomatis menggunakan kamera?

2. Bagaimana memanfaatkan sensor ultrasonik untuk mengukur panjang antrean kendaraan secara akurat?

3. Bagaimana merancang sistem logika fuzzy untuk menentukan lama nyala lampu merah berdasarkan jumlah dan panjang antrean kendaraan?

1.3 Tujuan

1. Merancang dan mengimplementasikan sistem lampu merah otomatis berbasis kamera dan sensor ultrasonik.

2. Mengembangkan metode penghitungan jumlah kendaraan menggunakan kamera dan pengukuran panjang antrean dengan sensor ultrasonik.

3. Mengaplikasikan logika fuzzy untuk mengatur waktu nyala lampu merah berdasarkan kondisi antrean kendaraan secara real-time.

BAB II
KAJIAN TEORI DAN IMPLEMENTASI SISTEM

2.1 Object Detection

Object detection merupakan suatu teknik pengolahan gambar yang digunakan untuk mendeteksi dan mengklasifikasi objek. Teknologi merupakan bagian dari AI (Artificial Intellegence) yang berupa suatu model berisi weight dan bias. Untuk menjalankan infrerence dari object detection ini kami menggunakan framework yang mampu untuk menangkap dan memproses citra yakni Open CV

2.1.1 Model : YOLO11

Ultralytics YOLO11 merupakan lanjutan terkini dari seri YOLO yang dirancang untuk performa real‑time dan kemudahan integrasi. Model ini menggunakan convolutional neural network (CNN) untuk memproses gambar utuh dan menghasilkan prediksi posisi serta probabilitas kelas dalam satu langkah inferensi. Selain deteksi objek, YOLO11 mendukung instance segmentation, pose estimation, tracking, dan klasifikasi gambar, sehingga menjadi solusi all‑in‑one bagi proyek computer vision.

Arsitektur YOLO11 mengadopsi desain modular yang meliputi backbone untuk ekstraksi fitur, neck untuk agregasi multi‑skala (misalnya FPN/PAN), dan head untuk prediksi bounding box dan skor kepercayaan. Optimalisasi dilakukan melalui teknik seperti meta‑anchors adaptif, augmentasi mosaic, serta pelatihan terdistribusi, sehingga mencapai trade‑off optimal antara kecepatan inferensi dan ketepatan deteksi.

2.1.2 OpenCV

OpenCV (Open Source Computer Vision Library) adalah pustaka open-source terpopuler untuk aplikasi computer vision dan machine learning. Beberapa fitur utamanya:

- **Pemrosesan Gambar Dasar**: Baca/tulis gambar, transformasi geometris, filtering, thresholding, serta deteksi tepi (Canny, Sobel).
- **Video I/O**: Membaca dan menulis video, serta menangani stream kamera secara real-time.
- **Fitur Deteksi dan Tracking**: Implementasi detektor wajah (Haar Cascade), deteksi sudut (Shi–Tomasi), serta tracker multi-object (KCF, CSRT, MIL).
- **Integrasi DNN**: Mendukung inference model deep learning (Caffe, TensorFlow, PyTorch, ONNX)—termasuk YOLO—secara langsung melalui modul

- **Optimasi Performa**: Dukungan multithreading, OpenCL, dan T-API (Transparent API) untuk akselerasi di berbagai hardware.

Pada implementasi sistem ini, OpenCV berfungsi sebagai “wrapping” untuk preprocessing input, menggambar hasil deteksi (bounding box + label), serta menyajikan video output ke antarmuka pengguna.

2.1.3 Proses Training

Untuk mengembangkan model kami menggunakan platform roboflow dengan dataset custom yang merupakan campuran dari data yang diambil dari internet dan data yang diambil secara mandiri oleh penulis. dalam langkah training dibagi menjadi 2 tahap, diantaranya:

1. Persiapan Dataset di Roboflow: Dataset dari internet yang di Fork lalu ditambahkan dengan data yang diambil secara mandiri sesuai dengan kebutuhan. Setelah mengupload gambar , label diberikan pada masing masing gambar melalui bantuan dari fitur auto-label yang dimiliki roboflow lalu diperiksa kembali. Jika terdapat kesalahan dari proses auto label maka bisa diperbaiki secara manual oleh pengguna. Setelah semua dataset berlabel maka langkah selanjutnya augmentasi untuk menambah jumlah data dan pemisahan data yang masing masing untuk train/test/split.

2. Pelatihan Model menggunakan Google Collab: Setelah dataset siap maka langkah selanjutnya adalah melatih model menggunakan Google Collab. Langkah pertama adalah menginstal dependensi yang diperlukan seperti ultralytics dan roboflow dengan perintah “pip install”. Setelah semua depedensi terinstall maka kita bisa mengimpor dataset dengan API yang didapat saat kita akan mendownload dataset dari platform roboflow. Selanjutnya proses training bisa langsung dilakukan. Setelah proses training selesai maka anda memiliki model yang anda inginkan

2.2 Wifi Communication

Pada sistem ini, koneksi WiFi dimanfaatkan sebagai media komunikasi antara perangkat komputer (client) dan mikrokontroler ESP32 (server) untuk mengirimkan data hasil deteksi objek secara real-time. Setelah model YOLO mendeteksi objek pada frame video, data jumlah objek yang terdeteksi (dalam hal ini mobil) akan dihitung dan dihaluskan menggunakan metode buffer (moving average). Hasil perhitungan ini kemudian dikirim ke ESP32 melalui permintaan HTTP GET dengan membentuk URL yang berisi alamat IP ESP32 dan parameter jumlah mobil,Sebagai contoh : http://192.168.0.194/count?cars=2. Proses pengiriman dilakukan secara asinkron menggunakan thread terpisah agar tidak menghambat proses deteksi video utama. Library requests digunakan untuk melakukan pengiriman HTTP tersebut, dan ESP32 harus berada dalam satu jaringan WiFi yang sama dengan perangkat komputer agar komunikasi dapat berjalan lancar. Dengan pendekatan ini, sistem dapat mengirim data hasil deteksi secara efisien dan real-time tanpa mengganggu performa utama dari proses pengolahan citra.

2.3 Sensor Ultrasonic

Pada alat ini digunakan sensor ultrasonic HCSR04 sebagai bagian dari sistem yang dirancang untuk mengukur panjang antrian kendaraan di persimpangan lalu lintas. Sensor ini bekerja dengan mengirimkan gelombang suara ultrasonik ke arah objek (dalam hal ini kendaraan) dan menerima pantulan gelombang tersebut untuk menghitung jarak berdasarkan waktu yang dibutuhkan oleh gelombang suara untuk kembali. Informasi jarak yang diperoleh digunakan bersama data dari kamera untuk menentukan panjang antrian kendaraan. Sensor ini mendukung sistem logika fuzzy yang mengatur durasi lampu lalu lintas berdasarkan kepadatan lalu lintas secara real-time.

2.4 Fuzzy Logic

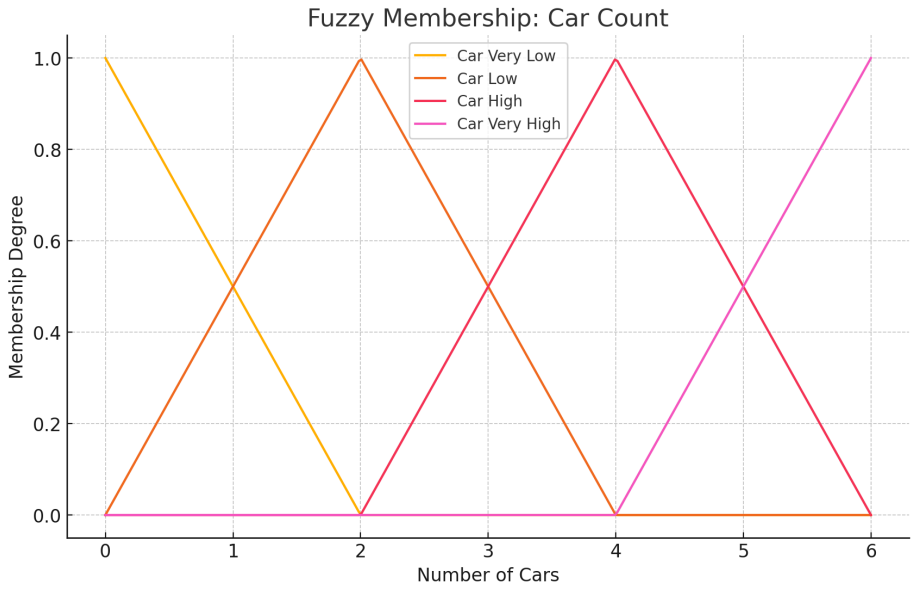
Untuk menjalankan alat, kami menggunakan logika fuzzy yang memiliki input variabel “banyak mobil” dan “jarak antrian” yang diinterpretasikan dengan derajat keanggotaan yang merupakan rentang nilai antara 0 dan 1. Dalam sistem traffic light ini logika fuzzy digunakan untuk menyesuaikan durasi lampu hijau secara bervariatif sesuai dengan kebutuhan yang didasarkan pada jumlah mobil dan antrean kendaraan.

2.4.1 Fuzzy Set

Pada tahap ini, setiap variabel input dan output didefinisikan dalam beberapa himpunan fuzzy (fuzzy sets) dengan fungsi keanggotaan (membership functions) berbentuk segitiga (triangular).

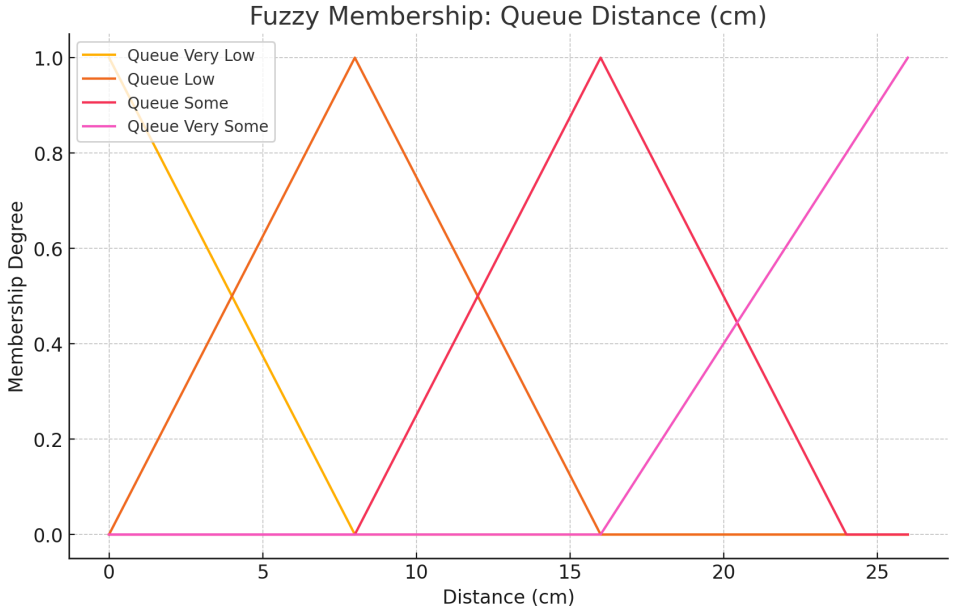
Input 1: Jumlah Mobil (“cars”)

- *Very Low (VL)*: puncak di 0 mobil, turun hingga nol di 2 mobil
- *Low (L)*: puncak di 2 mobil, turun hingga nol di 0 dan 4 mobil
- *High (H)*: puncak di 4 mobil, turun hingga nol di 2 dan 6 mobil
- *Very High (VH)*: puncak di 6 mobil, naik dari nol mulai 4 mobil
  -



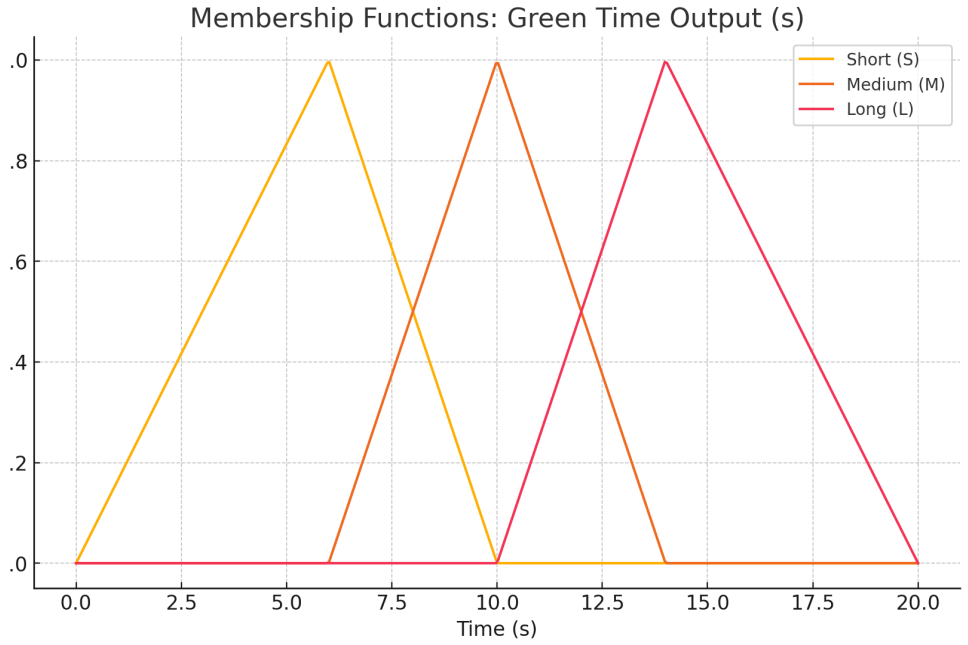
Input 2: Panjang Antrean (“distance”)

- *Very Low (QVL)*: puncak di 0 cm, turun hingga nol di 8 cm
- *Low (QL)*: puncak di 8 cm, turun hingga nol di 0 dan 16 cm
- *Some (QS)*: puncak di 16 cm, turun hingga nol di 8 dan 24 cm
- *Very Some (QVS)*: puncak di 26 cm, naik dari nol mulai 16 cm



Output: Durasi Lampu Hijau (“green time”)

- *Short (S)*: puncak di 5 s, turun hingga nol di 0 s dan 10 s
- *Medium (M)*: puncak di 10 s, turun hingga nol di 5 s dan 15 s
- *Long (L)*: puncak di 15 s, turun hingga nol di 10 s dan 20 s



2.4.2 Fuzzy Rule

Berdasarkan kombinasi derajat keanggotaan kedua input (cars dan distance), kita Dapat menggunakan rule tabel (rule matrix) 4×4:

Cars ↓ \ Distance →	QVL	QL	QS	QVS
Car VL	S	S	S	S
Car L	S	M	M	M
Car H	S	M	L	L
Car VH	S	M	L	L

Di dalam Rule, setiap nilai huruf (S,M,L) mengacu pada satu kategori durasi lampu hijau sebagai berikut:

- “Short” (S) : Menggunakan fungsi keanggotaan untuk durasi pendek (puncak di 5 s).

- Medium” (M): Menggunakan fungsi keanggotaan untuk durasi sedang (puncak di 10 s).

- Long” (L): Menggunakan fungsi keanggotaan untuk durasi panjang (puncak di 15 s).

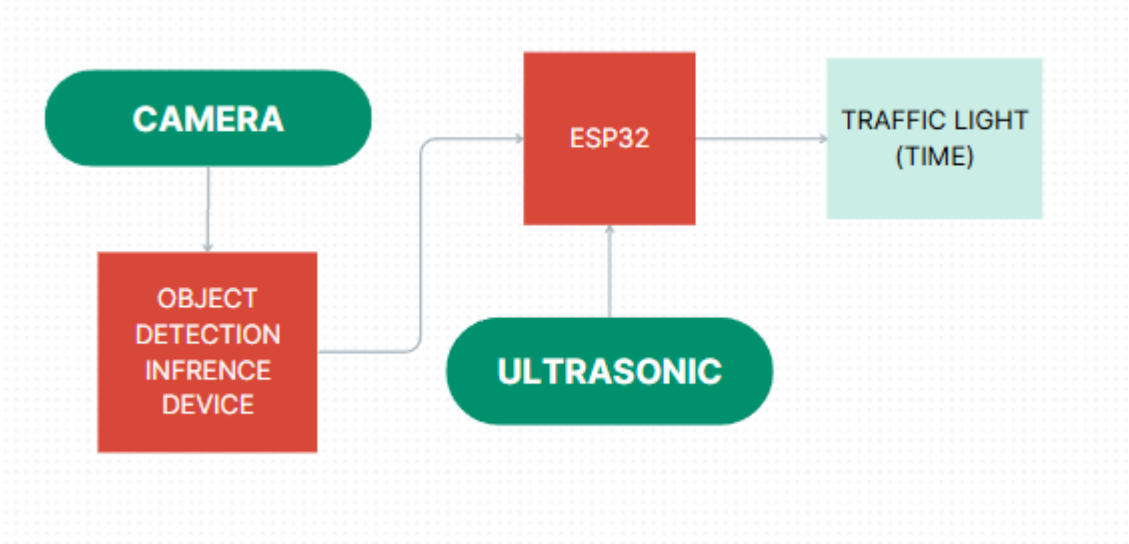
2.4.3 Defuzzification

Setelah agregasi, diperoleh fungsi keanggotaan gabung pada domain durasi $t \in [0,20]$ s. Metode defuzzifikasi yang dipakai adalah **centroid**:

$$t^* = \frac{\int_0^{20} t\ \mu_{\text{agg}}(t)\,dt}{\int_0^{20} \mu_{\text{agg}}(t)\,dt}$$

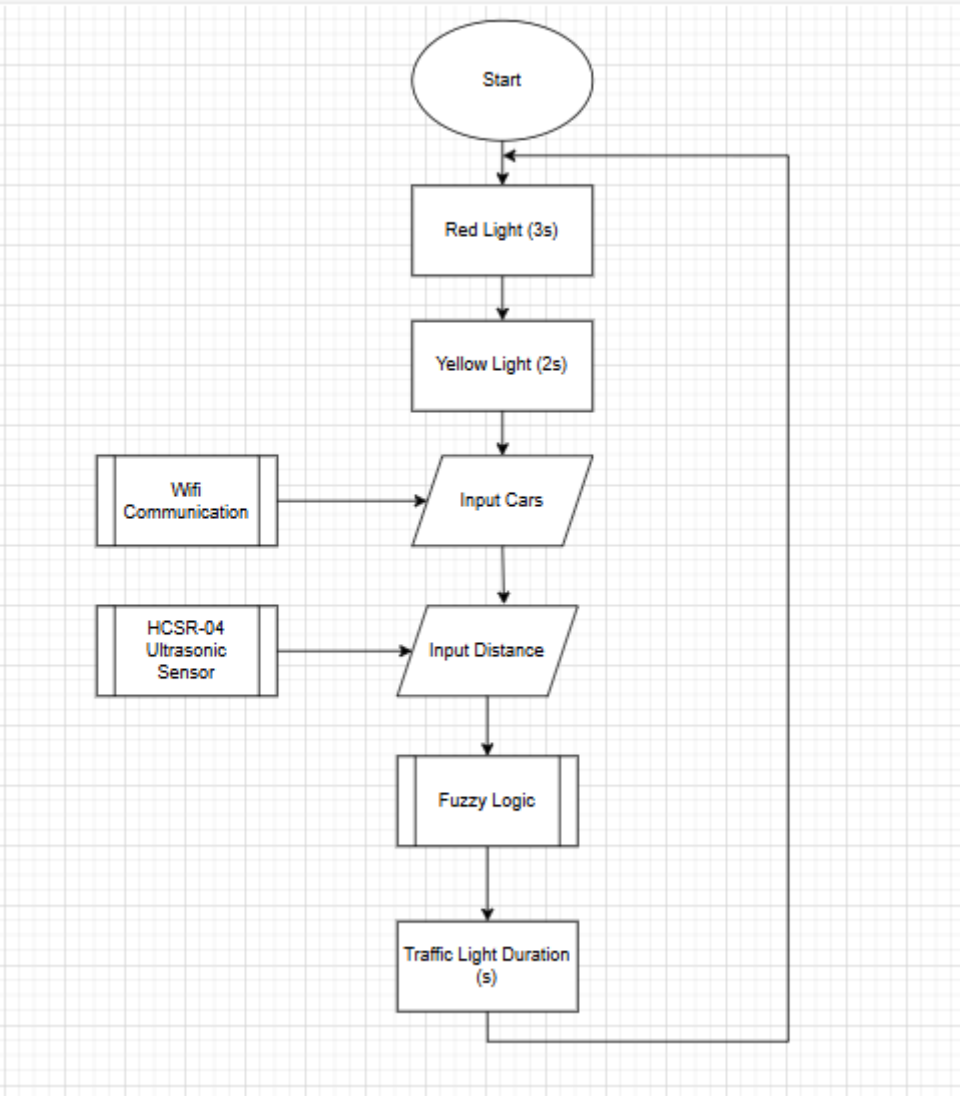
Dengan pendekatan ini, traffic light dapat menyesuaikan durasi hijau secara halus sesuai tingkat kepadatan kendaraan, mengurangi waktu tunggu saat antrian panjang dan menghindari lampu hijau terlalu lama saat lalu lintas sepi.

2.5 Hardware & Software



Dari sisi hardware sistem ini menggunakan camera dan sensor ultrasonik sebagai input dan lampu lalu lintas sebagai outpunya. Camera menangkap gambar yang diproses melalui sebuah model deteksi objek akan menginput jumlah mobil yang terdeteksi oleh

kamera melalui protokol komunikasi Wifi melalui bahasa pemograman python (Lihat Lampiran 1).



Di sisi lain sensor ultrasonik akan mengukur jarak antara posisinya dan posisi mobil (dalam cm) yang dikombinasikan dengan input jumlah mobil untuk masuk ke kontroller berbasis logika fuzzy yang akan menghasilkan lamanya lampu lalu lintas akan menyala (Lihat Lampiran 2).

BAB III
PENGOLAHAN DATA

3.1. Pengukuran Dan Statistik

No.	Jumlah Mobil (Cars)	Jarak Rata-rata (cm)	Durasi Lampu Hijau (s)
1	0	26.0	5.33
2	1	24.2	5.22
3	2	26.0	5.33
4	1	16.7	5.22
5	3	17.0	7.23
6	4	14.4	10.00
7	5	9.6	12.77
8	6	5.6	14.71

Jumlah Sampel: 8 pengamatan.

- **Rentang Cars: 0–6 unit.**
- **Rentang Jarak: 5.6–26.0 cm.**
- **Durasi Lampu Hijau: 5.22–14.71 s.**

3.2 Analisis

1. **Jumlah Mobil vs. Durasi Hijau**

    - **Jumlah Mobil dan Durasi lampu berkorelasi positif : semakin banyak mobil, semakin lama durasi lampu hijau.**

2. **Jarak vs. Durasi Hijau**

    - **Jarak dan durasi lampu berkorelasi negativ: semakin dekat jarak (kerumunan lebih rapat), semakin lama lampu hijau.**

Logika Fuzzy berhasil menangkap dua faktor utama—kepadatan kendaraan (Cars) dan kerapatan pergerakan (Dist) yang memungkinkan output dari durasi lampu hijau yang meningkat secara signifikan saat kondisi lalu lintas padat namun durasi cepat saat kondisi lalu lintas lancar.

BAB IV
KESIMPULAN DAN SARAN

4.1 Kesimpulan

Model fuzzy yang dikembangkan menunjukkan respons adaptif terhadap variabel kepadatan (Cars) dan jarak antar-kendaraan (Dist), menghasilkan output durasi lampu hijau yang dapat berubah ubah sesuai kebutuhan. Korelasi tinggi menegaskan efektivitas rule base.

4.1 Saran

- namun uji lebih lanjut dengan data beragam dan visualisasi mendetail akan memperkuat validitas model.

DAFTAR PUSTAKA

Abdul-Manan, A. F., Ahmad, F. B. H., & Majid, A. H. A. (2011). *Fuzzy logic based traffic signal control system*. In 2011 IEEE Symposium on Industrial Electronics & Applications (pp. 410–415). IEEE.

Khiang, Kok & Khalid, Marzuki & Yusof, Rubiyah. (1997). Intelligent Traffic Lights Control By Fuzzy Logic. Malaysian Journal of Computer Science. 9. 29-35.

Ultralytics. (2025, February 26). *YOLO11 NEW*. https://docs.ultralytics.com/models/yolo11/

Espressif Systems. (2021). *ESP-IDF Programming Guide: Wi-Fi API*. Espressif Systems. Dokumentasi resmi API Wi-Fi untuk ESP32, mencakup konfigurasi STA/AP, event handler, dan pengelolaan koneksi.

OpenCV: OpenCV modules. (n.d.). https://docs.opencv.org/4.x/

Handson Technology. (–). *HC-SR04 Ultrasonic Sensor Module User Datasheet*. Handson Technology.

N. I. Abdulkhaleq, I. J. Hasan, & N. A. J. Salih, “Investigating the resolution ability of the HC-SR04 ultrasonic sensor,” *ResearchGate*, 2019.

LAMPIRAN

Python Code

```
import cv2
import threading
import queue
import time
from collections import Counter, deque

import requests
from ultralytics import YOLO
from ensemble_boxes import weighted_boxes_fusion

def async_request(url, timeout=0.5, retries=2, backoff=0.2):
    """Send an HTTP GET asynchronously, with simple retry/backoff."""
    try:
        for attempt in range(retries):
            print(f"[HTTP→ESP32] GET {url}")
            resp = requests.get(url, timeout=timeout)
            if resp.status_code == 200:
                print("✓ Sent successfully")
                return
            else:
                print(f"✗ HTTP {resp.status_code}, attempt {attempt+1}")
                time.sleep(backoff * (attempt + 1))
        print("✗ All retries failed")
    except Exception as e:
        print(f"✗ Request error: {e}")

class VideoCaptureAsync:
    """Threaded video capture with non-blocking read()."""
    def __init__(self, src=0, width=640, height=480):
        self.cap = cv2.VideoCapture(src)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
        self.q = queue.Queue(maxsize=1)
        self.stopped = False
        threading.Thread(target=self._reader, daemon=True).start()

    def _reader(self):
        while not self.stopped:
            ret, frame = self.cap.read()
            if not ret:
                continue
            if not self.q.empty():
                try:
```

```
                self.q.get_nowait()
            except queue.Empty:
                pass
        self.q.put(frame)

    def read(self, timeout=1.0):
        try:
            return self.q.get(timeout=timeout)
        except queue.Empty:
            return None

    def release(self):
        self.stopped = True
        self.cap.release()

def fuse_boxes(boxes, scores, labels, frame_w, frame_h,
               iou_thr=0.5, skip_box_thr=0.0):
    """Apply Weighted Box Fusion."""
    boxes_norm = [[x1/frame_w, y1/frame_h, x2/frame_w, y2/frame_h]
                  for x1, y1, x2, y2 in boxes]
    boxes_fused, scores_fused, labels_fused = weighted_boxes_fusion(
        [boxes_norm], [scores], [labels],
        iou_thr=iou_thr,
        skip_box_thr=skip_box_thr
    )
    boxes_out = [
        [int(x1*frame_w), int(y1*frame_h), int(x2*frame_w), int(y2*frame_h)]
        for x1, y1, x2, y2 in boxes_fused
    ]
    return boxes_out, scores_fused, labels_fused

class ObjectDetection:
    def __init__(
        self,
        model_path='best.pt',
        esp32_ip='192.168.0.195',
        confidence_thresh=0.65,
        buffer_size=5,
        nms_iou=0.50,
        max_det=50
    ):
        self.model = YOLO(model_path)
        print("Model classes:", self.model.names)
        self.esp32_ip = esp32_ip
        self.conf_thresh = confidence_thresh
        self.window = deque(maxlen=buffer_size)
        self.nms_iou = nms_iou
        self.max_det = max_det
```

```
def draw_boxes(self, frame, boxes, labels, scores):
    for (x1, y1, x2, y2), cls_id, conf in zip(boxes, labels, scores):
        name = self.model.names[int(cls_id)]
        label = f"{name} {conf:.2f}"
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(frame, label, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    return frame

def process_frame(self, frame):
    h, w = frame.shape[:2]
    results = self.model(
        frame,
        conf=self.conf_thresh,
        iou=self.nms_iou,
        max_det=self.max_det,
        verbose=False
    )[0]

    raw_boxes, raw_scores, raw_labels = [], [], []
    if results.boxes is not None and results.boxes.cls.numel() > 0:
        for xyxy, cid, conf in zip(
            results.boxes.xyxy.cpu().numpy(),
            results.boxes.cls.cpu().numpy().astype(int),
            results.boxes.conf.cpu().numpy()
        ):
            raw_boxes.append(xyxy.tolist())
            raw_scores.append(float(conf))
            raw_labels.append(int(cid))

    boxes, scores, labels = fuse_boxes(
        raw_boxes, raw_scores, raw_labels,
        frame_w=w, frame_h=h,
        iou_thr=self.nms_iou,
        skip_box_thr=self.conf_thresh
    )

    # Count any class containing "car"
    counts = Counter()
    for cid, conf in zip(labels, scores):
        name = self.model.names[int(cid)]
        if 'car' in name.lower() and conf >= self.conf_thresh:
            counts['car'] += 1
    count = counts['car']

    # Smooth the count
    self.window.append(count)
```

```
        smooth_count = round(sum(self.window) / len(self.window))

        # Fire off HTTP
        url = f"http://{self.esp32_ip}/count?cars={smooth_count}"
        threading.Thread(target=async_request, args=(url,), daemon=True).start()

        vis = self.draw_boxes(frame.copy(), boxes, labels, scores)
        return vis, smooth_count

    def run(self, src=0, save_output=False, output_path='output.avi'):
        cap = VideoCaptureAsync(src)
        cv2.namedWindow('Detection', cv2.WINDOW_NORMAL)

        if save_output:
            fourcc = cv2.VideoWriter_fourcc(*'XVID')
            writer = cv2.VideoWriter(output_path, fourcc, 20,
                                     (int(cap.cap.get(3)), int(cap.cap.get(4))))

        prev_time = time.time()
        try:
            while True:
                frame = cap.read()
                if frame is None:
                    continue

                out_frame, smooth_count = self.process_frame(frame)

                now = time.time()
                fps = 1.0 / max(now - prev_time, 1e-6)
                prev_time = now
                cv2.putText(out_frame, f"FPS: {fps:.2f}", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

                cv2.imshow('Detection', out_frame)
                if save_output:
                    writer.write(out_frame)

                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

        finally:
            cap.release()
            cv2.destroyAllWindows()
            if save_output:
                writer.release()

if __name__ == '__main__':
    detector = ObjectDetection(
```

```
    model_path='CDM200.pt',
    esp32_ip='192.168.0.195',
    confidence_thresh=0.55,
    buffer_size=5,
    nms_iou=0.5,
    max_det=50
)
detector.run(src=1, save_output=False, output_path='cars_output.avi')
```

Arduino Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <WebServer.h>

// ———————— Wi-Fi credentials
————————————————————————————————————
const char* SSID = "Siska W";
const char* PASS = "md162322";

// ———————— Ultrasonic sensor pins
————————————————————————————
const uint8_t TRIG_PIN = 23;
const uint8_t ECHO_PIN = 22;

// ———————— Traffic light LED pins ————————————————————————————————————
const uint8_t RED_PIN    = 17;
const uint8_t YELLOW_PIN = 5;
const uint8_t GREEN_PIN  = 18;

// ———————— Fixed phase durations (seconds) ——————————————————————
const float YELLOW_DURATION = 3.0f;
const float RED_DURATION    = 3.0f;

// ———————— Fuzzy logic parameters ————————————————————————————————
template<typename T>
float trimf(T x, T a, T b, T c) {
  if (x == b) return 1.0f;
  if (x <= a || x >= c) return 0.0f;
  if (x < b)  return (float)(x - a) / (float)(b - a);
  return (float)(c - x) / (float)(c - b);
}

float mfCarVL(float x) { return trimf(x, 0.0f, 0.0f, 2.0f); }
float mfCarL (float x) { return trimf(x, 0.0f, 2.0f, 4.0f); }
float mfCarH (float x) { return trimf(x, 2.0f, 4.0f, 6.0f); }
float mfCarVH(float x){ return trimf(x, 4.0f, 6.0f, 6.0f); }

float mfQVL(float x){ return trimf(x, 0.0f,   0.0f,  8.0f); }
float mfQL (float x) { return trimf(x, 0.0f,   8.0f, 16.0f); }
float mfQS (float x) { return trimf(x, 8.0f,  16.0f, 24.0f); }
float mfQVS(float x){ return trimf(x, 16.0f, 26.0f, 26.0f); }

// ———————— Defuzz output centroids ——————————————————————————————
const float T_MIN =  0.0f;
const float T_MAX = 20.0f;
```

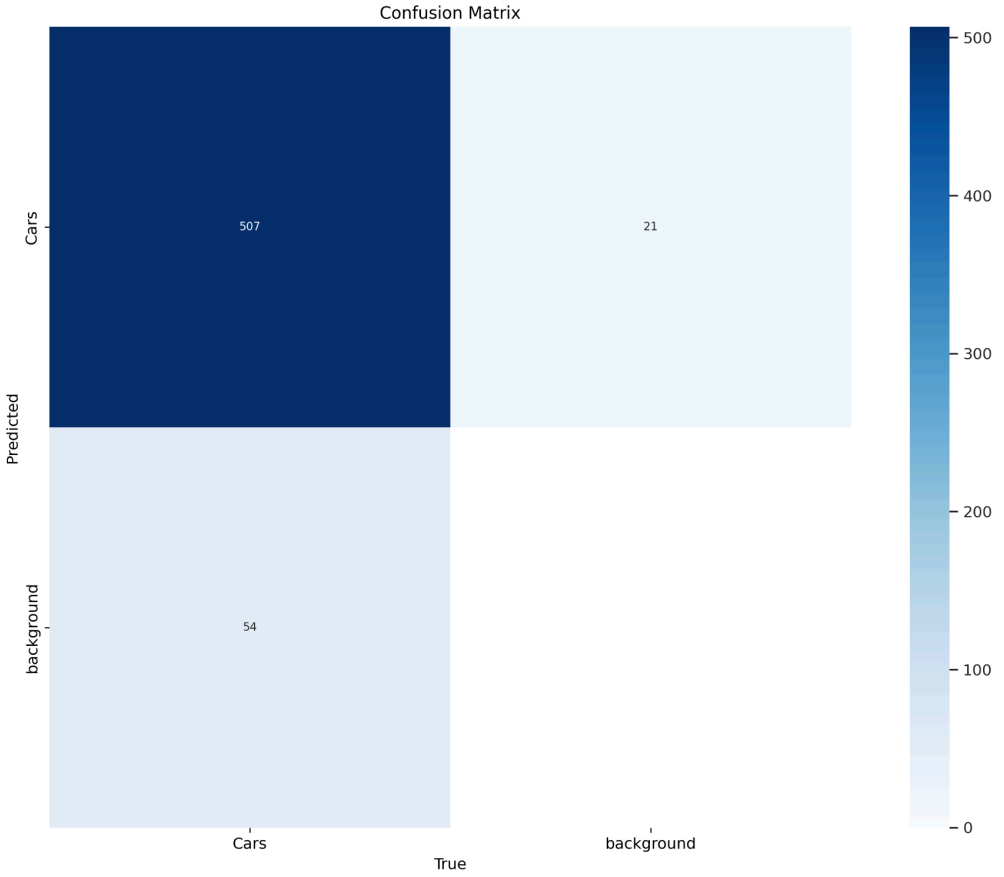
```
// shifted peaks for 5–7s, 6–8s, …, 13–15s bands
const float OUT_S =  6.0f;
const float OUT_M = 10.0f;
const float OUT_L = 14.0f;

float mfOutS(float t) { return trimf(t, T_MIN, OUT_S, OUT_M); }
float mfOutM(float t) { return trimf(t, OUT_S, OUT_M, OUT_L); }
float mfOutL(float t) { return trimf(t, OUT_M, OUT_L, T_MAX); }

// ———————— Updated rule matrix
————————————————————————————————————
// rows = {VL, L, H, VH} traffic levels
// cols = {QVS, QS, QL, QVL}   queue levels (we index [i][3-j] in code)
const int ruleMatrix[4][4] = {
  /*        QVS  QS   QL   QVL */
  /* VL */ { 0,   0,   0,    1 },
  /*  L */ { 0,   0,   1,    1 },
  /*  H */ { 0,   1,   1,    2 },
  /* VH */ { 1,   1,   2,    2 }
};

// ———————— HTTP server & car count storage
————————————————————————————
WebServer server(80);
volatile float latestCarCount = 0.0f;

void handleCount() {
  if (!server.hasArg("cars")) {
    server.send(400, "text/plain", "Missing ?cars=");
    return;
  }
  latestCarCount = constrain(server.arg("cars").toFloat(), 0.0f, 6.0f);
  server.send(200, "text/plain", "OK");
}

// ———————— Read HC-SR04 distance in cm
————————————————————————————
float readUltrasonicCm() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH, 30000L);
  if (duration == 0) return 100.0f;
  return duration / 29.1f / 2.0f;
}
```

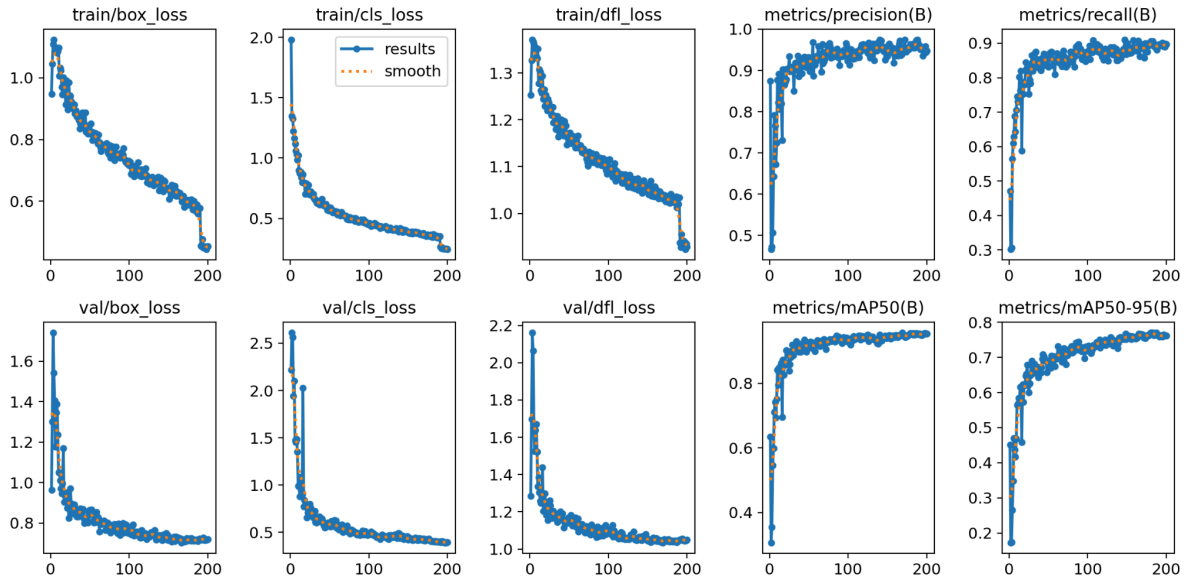
```
// ———————— Fuzzy centroid defuzzification ————————————————————
float computeGreenTimeCentroid(float cars, float dist) {
  float uCar[4]   = { mfCarVL(cars), mfCarL(cars), mfCarH(cars), mfCarVH(cars) };
  float uQueue[4] = { mfQVL(dist),   mfQL(dist),   mfQS(dist),   mfQVS(dist) };

  const int N = 200;
  float sumNum = 0.0f, sumDen = 0.0f;
  for (int k = 0; k <= N; k++) {
    float t = T_MIN + (T_MAX - T_MIN) * k / (float)N;
    float muAgg = 0.0f;
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        float f = min(uCar[i], uQueue[j]);
        if (f <= 0.0f) continue;
        int code = ruleMatrix[i][3 - j];
        float muOut = (code == 0 ? mfOutS(t)
                    : (code == 1 ? mfOutM(t)
                                 : mfOutL(t)));
        muAgg = max(muAgg, min(f, muOut));
      }
    }
    sumNum += t * muAgg;
    sumDen += muAgg;
  }
  return (sumDen > 0.0f) ? (sumNum / sumDen) : OUT_S;
}

// ———————— Phase State Machine
————————————————————————————————————————

enum Phase { RED, YELLOW, GREEN };
Phase phase = RED;
unsigned long phaseStart = 0;
unsigned long greenDurationMs = 0;

void setup() {
  Serial.begin(115200);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(RED_PIN,    OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN,  OUTPUT);

  WiFi.begin(SSID, PASS);
  Serial.print("Connecting Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print('.');
```

```
  }
  Serial.printf("\nConnected! IP=%s\n", WiFi.localIP().toString().c_str());

  server.on("/count", HTTP_GET, handleCount);
  server.begin();
  Serial.println("HTTP server running; GET /count?cars=<n>");

  phaseStart = millis();
}

void loop() {
  server.handleClient();

  unsigned long now = millis();
  unsigned long elapsed = now - phaseStart;

  switch (phase) {
    case RED:
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(GREEN_PIN, LOW);
      if (elapsed >= RED_DURATION * 1000UL) {
        phase = YELLOW;
        phaseStart = now;
      }
      break;

    case YELLOW:
      digitalWrite(RED_PIN, LOW);
      digitalWrite(YELLOW_PIN, HIGH);
      digitalWrite(GREEN_PIN, LOW);
      if (elapsed >= YELLOW_DURATION * 1000UL) {
        float dist = readUltrasonicCm();
        dist = constrain(dist, 0.0f, 26.0f);
        float gt = computeGreenTimeCentroid(latestCarCount, dist);
        greenDurationMs = (unsigned long)(gt * 1000.0f);

        phase = GREEN;
        phaseStart = now;
        digitalWrite(YELLOW_PIN, LOW);
        digitalWrite(GREEN_PIN, HIGH);

        Serial.printf("Cars=%.1f | Dist=%.1fcm → Green=%.2fs\n",
                      latestCarCount, dist, gt);
      }
      break;

    case GREEN:
```

```
      digitalWrite(RED_PIN, LOW);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(GREEN_PIN, HIGH);
      if (elapsed >= greenDurationMs) {
        phase = RED;
        phaseStart = now;
      }
      break;
  }
}
```

Training Statistics





Model Predictions

Software Outlook