LAPORAN TUGAS

LINE FOLLOWER PEMADAM API
KELOMPOK 2B "KUMBANG JAWA"

Diajukan untuk Memenuhi Tugas Mata Kuliah Robotika

Oleh :

Andre Saputra	**2312601**
Hafizh 'Abid Khalish	**2304423**
Faiz lintang prawira	**2301910**
Mochammad Dimas Saputra	**2311157**
Muzakki Al'Aarif	**2306153**

PROGRAM STUDI MEKATRONIKA DAN KECERDASAN BUATAN
KAMPUS DAERAH PURWAKARTA
UNIVERSITAS PENDIDIKAN INDONESIA

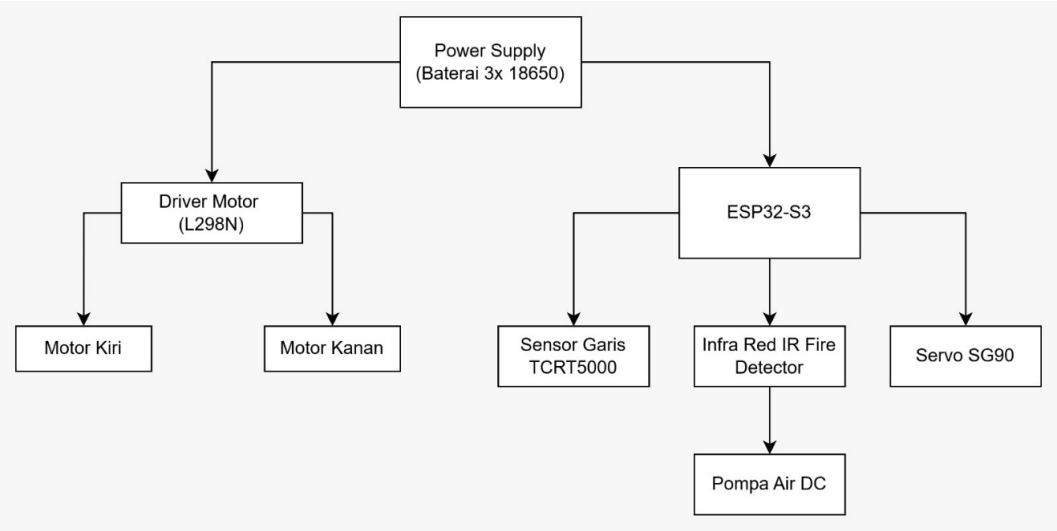
2025

KOMPONEN YANG DIGUNAKAN

No	Nama Barang	Estimasi Spesifikasi	Jumlah	Harga
1	Mikrokontroler	ESP32-S3 WiFi	1	106000
2	Expansion Board	Expansion Board ESP-32 30p Shield Development	1	20000
3	Motor Servo	Micro Servo SG90	4	52000
4	Driver Motor	L298N	1	20000
5	Dinamo	Motor DC Gearbox JGA25-370 12V	2	130000
6	Sensor Garis	TCRT5000 Array (5 Channel)	1	25000
7	Sensor Panas/Api	Infra Red IR Fire Detector	4	28000
8	Kabel	Kabel Jumper (Male-Male, Male-Female)	2 set	25000
9	Power Supply	Baterai Lithium 3.7V 18650	3	21000
10	Holder	Baterai Holder 3 Slot	1	20000
11	Pompa Air	Mini Micro Summersible Water Pump DC	1	15000
12	Roda	Roda PU 1.5 & 2 inch +Bearing Wheel Only - WO Polyurethane	2	16000
13	Selang Silikon	silicone hose tube 2mm x 4mm	1	12000
14	Chassis	3D design + Custom		Free

Untuk link pembelian dapat dilihat di sheet berikut: https://docs.google.com/spreadsheets/d/1De_bu9Nj2lohehVi5v_kCNyxhsBP2wBFyrCxl9-Erc/edit?gid=1536591802#gid=1536591802

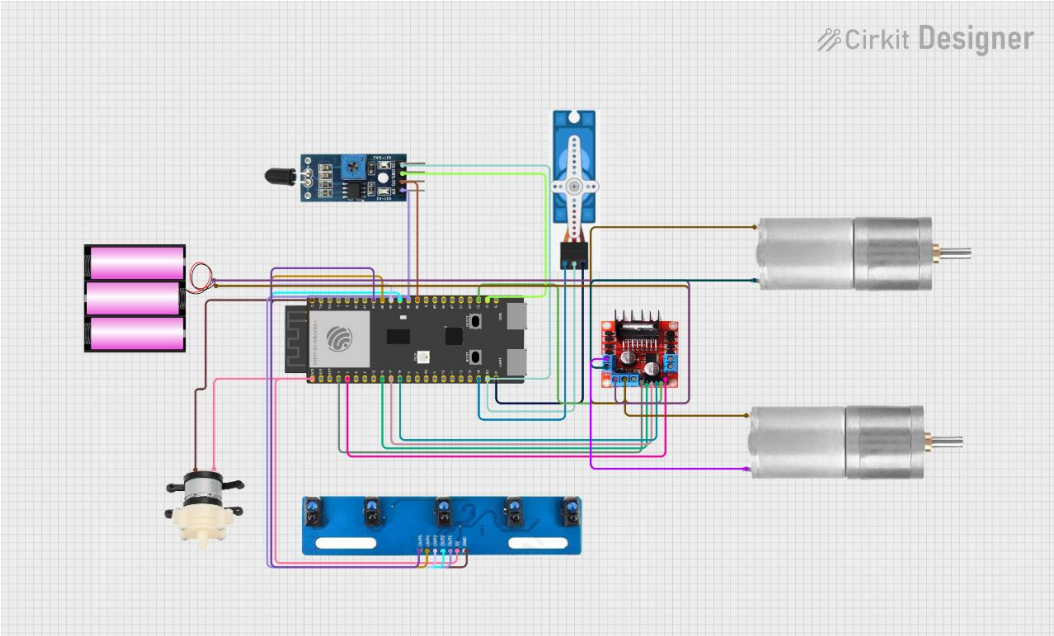
BLOK DIAGRAM INPUT



Robot ini bekerja sebagai sistem terpadu yang dikendalikan pusat oleh ESP32-S3. Sumber daya berasal dari tiga baterai 18650 yang memasok energi ke motor driver L298N, pompa air DC, dan ESP32 itu sendiri. ESP32 bertindak sebagai otak yang menerima semua input sensor dan mengatur seluruh aktuator. Sensor garis TCRT5000 memberi data posisi jalur sehingga ESP32 bisa mempertahankan gerakan robot tetap mengikuti garis menggunakan dua motor DC gearbox yang dikendalikan lewat L298N.

Saat robot mendekati area dengan potensi api, Infra Red Fire Detector mendeteksi radiasi inframerah dari nyala api dan mengirim sinyal ke ESP32. Begitu sinyal diterima, ESP32 menghentikan mode line follower dan mengarahkan robot menuju titik api dengan mengatur kecepatan serta arah kedua motor. Servo SG90 kemudian digunakan untuk menggerakkan arah semprotan air, menyesuaikan posisi nozzle agar tepat menghadap sumber api. Setelah posisi tepat, pompa air DC diaktifkan untuk menyemprotkan air melalui selang silikon, dan proses berlangsung sampai api padam. Setelah tidak ada lagi deteksi api, ESP32 memanggil kembali mode navigasi garis, dan robot melanjutkan perjalanan seperti sebelumnya.

WIRING DIAGRAM

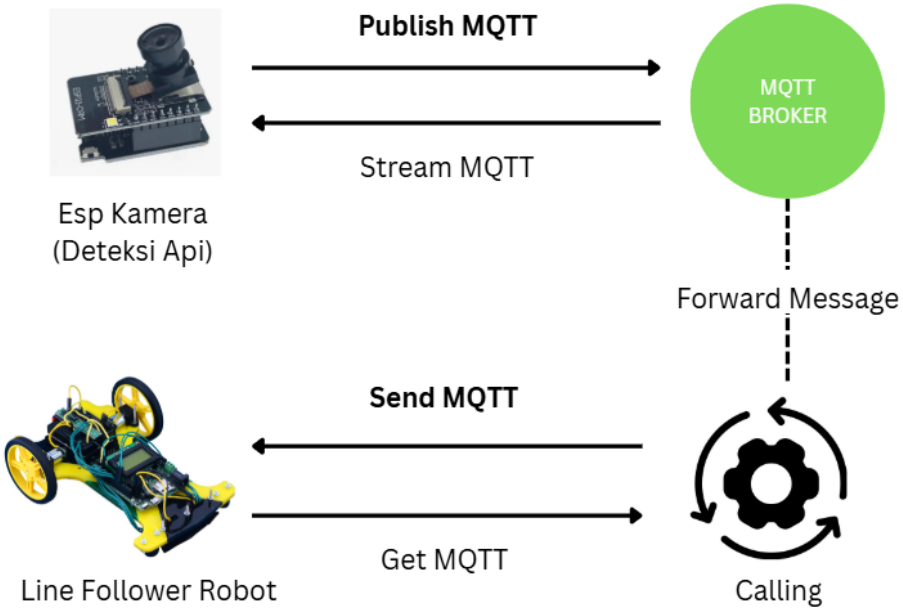


Untuk sistem penggerak utama, kedua motor DC gearbox JGA25-370 dihubungkan langsung ke modul L298N sebagai motor driver. Masing-masing motor memiliki dua kabel keluaran yang disambungkan ke terminal OUT1–OUT2 untuk motor kiri dan OUT3–OUT4 untuk motor kanan. Modul L298N kemudian dikendalikan oleh ESP32-S3 melalui enam pin kontrol: IN1, IN2, dan ENA untuk motor kiri serta IN3, IN4, dan ENB untuk motor kanan. Pin IN1 dan IN2 menentukan arah putaran motor kiri, sedangkan IN3 dan IN4 menentukan arah motor kanan. Kedua pin ENA dan ENB digunakan untuk memberikan sinyal PWM dari ESP32-S3 sehingga kecepatan motor dapat diatur secara halus. Catu daya untuk L298N berasal dari baterai lithium 3S yang memberikan tegangan sekitar 11,1–12,6V, sementara ground baterai disatukan dengan ground ESP32 untuk memastikan referensi sinyal yang stabil di seluruh rangkaian.

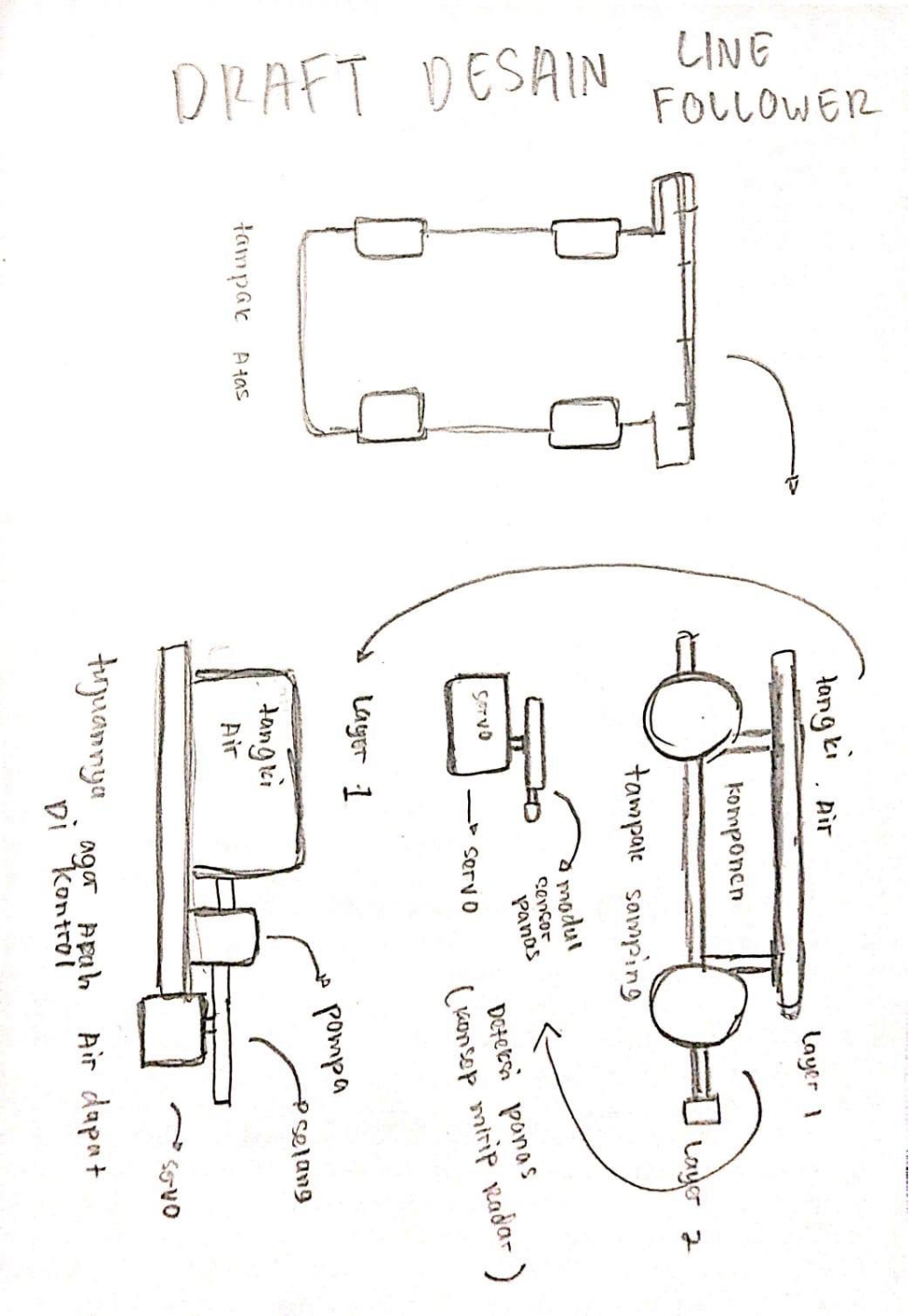
Selain sistem penggerak, robot juga dilengkapi pompa air mini 5V sebagai aktuator pemadam api. Berbeda dengan versi 12V, pompa 5V ini cukup dihubungkan langsung ke regulator 5V dari buck converter tanpa memerlukan MOSFET driver tambahan yang kompleks, karena arus yang digunakan lebih rendah dan kompatibel dengan sistem tegangan sensor lainnya. Pompa dikendalikan melalui satu pin GPIO ESP32-S3 yang memicu relay solid state atau transistor kecil, sehingga pompa dapat diaktifkan dan dimatikan sesuai perintah

logika ketika sensor api mendeteksi nyala. Penggunaan pompa 5V membuat integrasi daya lebih sederhana dan meminimalkan panas pada rangkaian pemicu, namun tetap memberi debit air yang cukup untuk proses pemadaman dalam skala robot line follower.

KOMUNIKASI DAN DATA FLOW



DESAIN MEKANIK



Desain yang akan kami buat dapat dilihat pada gambar diatas,

Pada gambar tersebut line follower yang dilengkapi sistem pemadaman yang akan kami buat terdiri dari dua layer. Layer Pertama akan di isi dengan tangki

air, motor pompa air, selang dan juga motor servo yang akan mengatur arah semprotan air yang akan ditembakkan ke target. Nanti paada layer 1 ini akan kami buat se-aman mungkin untuk mengindari siraman air mengenai komponen yang berada di bawahnya, juga penempatan air berada di atas untuk memudahkan mobilitas dalam isi ulang air dan posisi dalam penyemprotan air.

Layer kedua akan berisi komponen – komponen inti diantaranya ada ESP32, motor untuk menggerakkan roda, motor driver, sensor deteksi panas dan servo. Servo dan sensor deteksi panas akan disatukan, tujuannya adalah untuk menciptakan medan deteksi api selayaknya sebuah radar yang mana juga dapat menghemat pengeluaran dana karena hanya membutuhkan 1 sensor deteksi.

Untuk memudahkan dalam maintenance kami akan membuat agar layer 1 dan layer 2 dapat di bongkar pasang dengan mudah dan tetap kokoh walaupun kedua layer tersebut tidak di pasang secara permanent. Adapun untuk bahan dasar yang akan digunakan sebagai layer masih dalam pertimbangan dengan tim (antara dengan filament 3D Print atau dengan bahan lain seperti kaca mika).

KODE

Kode arduino Uno Line follower:

```
#include <ESP32Servo.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <Preferences.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

/* ================= WiFi & MQTT CONFIG ================= */
const char* ssid = "seipa";
const char* password = "00000001";
const char* mqtt_server = "10.83.74.17";
const int mqtt_port = 1883;

const char* topic_fire_detected = "robot/fire/detected";
const char* topic_status = "robot/status";

WiFiClient espClient;
PubSubClient client(espClient);

/* ================= PIN LINE SENSOR ================= */
#define S1 34
#define S2 35
#define S3 32
#define S4 33
#define S5 25
```

```
/* ================= MOTOR DRIVER ================= */
#define IN1 14
#define IN2 27
#define ENA 13
#define IN3 26
#define IN4 12
#define ENB 23

/* ================= FIRE SYSTEM ================= */
#define FLAME_PIN 39       // LOW = flame detected
#define RELAY_PIN 18

/* ================= SERVO (OPTIONAL) ================= */
#define SERVO_RADAR_PIN 4
#define SERVO_LOCK_PIN 5

Servo servoRadar;
Servo servoLock;

/* ================= PID LINE FOLLOWER ================= */
float Kp = 30;
float Kd = 300;
float lastError = 0;
float filteredError = 0;
float alpha = 0.65;

/* ================= SPEED ================= */
int baseSpeed = 100;
int maxSpeed = 130;

/* ================= STATE MACHINE ================= */
enum RobotState {
  IDLE = 0,
  MOVING_TO_FIRE = 1,
  EXTINGUISHING = 2,
  TURN_BACK_FROM_FIRE = 3,
  RETURNING = 4,
  FINAL_TURN_AT_START = 5,
  COMPLETED = 6
};

RobotState currentState = IDLE;

/* ================= PROTOTYPES ================= */
const char* stateToString(RobotState s);
void setState(RobotState s, const char* publishMsg = nullptr, bool
saveToNVS = true);

/* ================= FLAGS & TIMERS ================= */
bool missionActive = false;
unsigned long extinguishStartTime = 0;
const unsigned long extinguishDuration = 10000;

/*  ================= FLAME VALIDATION (ANTI JENTIKAN API)
================= */
const unsigned long FLAME_CONFIRM_MS = 10;
unsigned long flameLowSince = 0;
```

```
unsigned long flameLockoutUntil = 0;
const unsigned long FLAME_LOCKOUT_MS = 1500;

bool isFlameConfirmed() {
  int v = digitalRead(FLAME_PIN); // LOW = flame
  if (v == LOW) {
    if (flameLowSince == 0) flameLowSince = millis();
    if (millis() - flameLowSince >= FLAME_CONFIRM_MS) return true;
  } else {
    flameLowSince = 0;
  }
  return false;
}

/* ======================================================
   ✅ PERSISTENT STORAGE:
   - RTC memory for frequent updates (brownout-safe)
   - NVS backup only occasionally (power-off-safe)
   ====================================================== */
Preferences prefs;

// ===== RTC snapshot (survive brownout reset) =====
static const uint32_t RTC_MAGIC = 0xA55A1234;

RTC_DATA_ATTR uint32_t rtc_magic = 0;
RTC_DATA_ATTR uint8_t  rtc_state = (uint8_t)IDLE;
RTC_DATA_ATTR uint8_t  rtc_active = 0;
RTC_DATA_ATTR uint32_t rtc_extRemain = 0;
RTC_DATA_ATTR uint32_t rtc_lockRemain = 0;
RTC_DATA_ATTR float    rtc_lastErr = 0.0f;

// ===== NVS backup throttle =====
unsigned long lastNvsBackup = 0;
const unsigned long NVS_BACKUP_MS = 8000; // backup jarang biar gak
freeze

/* ================ DEBUG CONTROL ================ */
unsigned long lastDebugPrint = 0;
const unsigned long DEBUG_PRINT_MS = 300;

int lastLineErrCached = 0;

/* ================ FORWARD DECLARATIONS ================ */
void setupWiFi();
void reconnectMQTT();
void mqttCallback(char* topic, byte* payload, unsigned int length);

void motorStop();
void driveForward(int left, int right);
void rotateInPlace(int speed, int duration);
void executeReturnManeuver();
void lineFollower();
int readLineError();
bool isAtStartPosition();
void printDebugInfo();

/* ================ STATE STRING ================ */
```

```
const char* stateToString(RobotState s) {
  switch (s) {
    case IDLE: return "IDLE";
    case MOVING_TO_FIRE: return "MOVING_TO_FIRE";
    case EXTINGUISHING: return "EXTINGUISHING";
    case TURN_BACK_FROM_FIRE: return "TURN_BACK_FROM_FIRE";
    case RETURNING: return "RETURNING";
    case FINAL_TURN_AT_START: return "FINAL_TURN_AT_START";
    case COMPLETED: return "COMPLETED";
    default: return "UNKNOWN";
  }
}

/*   ================   RTC   SNAPSHOT   UPDATE   (VERY   LIGHT)
================ */
void rtcUpdateSnapshot() {
  // extRemain
  uint32_t extRemain = 0;
  if (currentState == EXTINGUISHING) {
    unsigned long elapsed = millis() - extinguishStartTime;
    extRemain   =   (elapsed   >=   extinguishDuration)   ?   0   :
(uint32_t)(extinguishDuration - elapsed);
  }

  // lockRemain
  uint32_t lockRemain = 0;
  if     (millis()     <     flameLockoutUntil)     lockRemain     =
(uint32_t)(flameLockoutUntil - millis());

  rtc_magic = RTC_MAGIC;
  rtc_state = (uint8_t)currentState;
  rtc_active = missionActive ? 1 : 0;
  rtc_extRemain = extRemain;
  rtc_lockRemain = lockRemain;
  rtc_lastErr = lastError;
}

/* ================ NVS SAVE (HEAVY) - THROTTLED ================
*/
void nvsSaveSnapshot(bool force) {
  if (!force && (millis() - lastNvsBackup < NVS_BACKUP_MS)) return;
  lastNvsBackup = millis();

  // simpan versi RTC yang paling baru (jadi konsisten)
  prefs.putUChar("state", rtc_state);
  prefs.putBool("active", rtc_active == 1);
  prefs.putULong("extRem", rtc_extRemain);
  prefs.putULong("lockRem", rtc_lockRemain);
  prefs.putFloat("lastErr", rtc_lastErr);
  prefs.putULong("savedAt", (uint32_t)millis());
}

/*   ================   RESTORE   (RTC   first,   NVS   fallback)
================ */
void restoreState() {
  // 1) RTC restore (best for brownout)
  if (rtc_magic == RTC_MAGIC) {
    currentState = (RobotState)rtc_state;
```

```
    missionActive = (rtc_active == 1);
    lastError = rtc_lastErr;
    filteredError = 0;

    flameLockoutUntil = millis() + (unsigned long)rtc_lockRemain;

    if (currentState == EXTINGUISHING) {
      if (rtc_extRemain > 0) {
        extinguishStartTime = millis() - (extinguishDuration -
(unsigned long)rtc_extRemain);
        digitalWrite(RELAY_PIN, HIGH);
        missionActive = true;
      } else {
        digitalWrite(RELAY_PIN, LOW);
        currentState = TURN_BACK_FROM_FIRE;
        missionActive = true;
      }
    } else {
      digitalWrite(RELAY_PIN, LOW);
    }

    if (currentState == IDLE || currentState == COMPLETED)
missionActive = false;

    Serial.print("♻️ RTC RESTORE: ");
    Serial.print(stateToString(currentState));
    Serial.print(" | extRemain=");
    Serial.print(rtc_extRemain);
    Serial.print(" | lockRemain=");
    Serial.println(rtc_lockRemain);
    return;
  }

  // 2) NVS fallback (power-off)
  if (!prefs.isKey("state")) {
    Serial.println("💾 No saved state found. Starting fresh
(IDLE).");
    currentState = IDLE;
    missionActive = false;
    digitalWrite(RELAY_PIN, LOW);
    return;
  }

  uint8_t st = prefs.getUChar("state", (uint8_t)IDLE);
  bool active = prefs.getBool("active", false);
  uint32_t extRemain = prefs.getULong("extRem", 0);
  uint32_t lockRemain = prefs.getULong("lockRem", 0);
  float savedLastErr = prefs.getFloat("lastErr", 0.0f);

  if (st > (uint8_t)COMPLETED) st = (uint8_t)IDLE;

  currentState = (RobotState)st;
  missionActive = active;
  lastError = savedLastErr;
  filteredError = 0;

  flameLockoutUntil = millis() + (unsigned long)lockRemain;
```

```
  if (currentState == EXTINGUISHING) {
    if (extRemain > 0) {
      extinguishStartTime = millis() - (extinguishDuration -
(unsigned long)extRemain);
      digitalWrite(RELAY_PIN, HIGH);
      missionActive = true;
    } else {
      digitalWrite(RELAY_PIN, LOW);
      currentState = TURN_BACK_FROM_FIRE;
      missionActive = true;
    }
  } else {
    digitalWrite(RELAY_PIN, LOW);
  }

  if (currentState == IDLE || currentState == COMPLETED)
missionActive = false;

  Serial.print("♻️ NVS RESTORE: ");
  Serial.print(stateToString(currentState));
  Serial.print(" | extRemain=");
  Serial.print(extRemain);
  Serial.print(" | lockRemain=");
  Serial.println(lockRemain);
}

/* ================= SET STATE ================= */
void setState(RobotState s, const char* publishMsg, bool saveToNVS)
{
  if (currentState == s) return;
  currentState = s;

  missionActive = (currentState != IDLE && currentState !=
COMPLETED);

  // update RTC immediately (light)
  rtcUpdateSnapshot();

  // publish status
  if (client.connected()) {
    client.publish(topic_status, publishMsg ? publishMsg :
stateToString(currentState));
  }

  // NVS save only on transition or when requested
  if (saveToNVS) nvsSaveSnapshot(true);

  Serial.print("➡️ STATE = ");
  Serial.println(stateToString(currentState));
}

/* ================================================ */
void setup() {
  Serial.begin(115200);
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  // Sensor Setup
  pinMode(S1, INPUT); pinMode(S2, INPUT); pinMode(S3, INPUT);
```

```
  pinMode(S4, INPUT); pinMode(S5, INPUT);

  // Motor Setup
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);

  // Fire System Setup
  pinMode(FLAME_PIN, INPUT);
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  // Servo Setup
  servoRadar.attach(SERVO_RADAR_PIN);
  servoLock.attach(SERVO_LOCK_PIN);
  servoRadar.write(90);
  servoLock.write(90);

  // NVS init
  prefs.begin("robot", false);

  // restore (RTC first)
  restoreState();

  setupWiFi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(mqttCallback);

  // ✅ OPTIONAL: kalau library PubSubClient kamu support:
  // client.setKeepAlive(60);   // perpanjang keepalive supaya gak
gampang timeout
  // client.setSocketTimeout(5);

  Serial.println("================================");
  Serial.println("🤖 FIRE FIGHTING ROBOT READY");
  Serial.print("🧠 RESUME STATE: ");
  Serial.println(stateToString(currentState));
  Serial.println("================================");
}

/* ================================================ */
void loop() {
  if (!client.connected()) reconnectMQTT();
  client.loop();

  // ===== update snapshot RTC setiap loop (super ringan) =====
  rtcUpdateSnapshot();

  // ===== NVS backup periodik tapi jarang (anti-freeze) =====
  if (missionActive || currentState == EXTINGUISHING) {
    nvsSaveSnapshot(false); // throttled by NVS_BACKUP_MS
  }

  // Debug
  if (millis() - lastDebugPrint >= DEBUG_PRINT_MS) {
    lastDebugPrint = millis();
    printDebugInfo();
```

```
  }

  switch (currentState) {

    case IDLE:
      motorStop();
      break;

    case MOVING_TO_FIRE:
      if (millis() < flameLockoutUntil) {
        lineFollower();
        break;
      }

      if (isFlameConfirmed()) {
        Serial.println("🔥 FLAME CONFIRMED (stable)!");
        motorStop();
        delay(200);

        digitalWrite(RELAY_PIN, HIGH);
        delay(30);

        extinguishStartTime = millis();
        flameLockoutUntil = millis() + FLAME_LOCKOUT_MS;

        setState(EXTINGUISHING, "EXTINGUISHING", true);
        break;
      }

      lineFollower();
      break;

    case EXTINGUISHING:
      motorStop();
      digitalWrite(RELAY_PIN, HIGH);
      missionActive = true;

      if (millis() - extinguishStartTime >= extinguishDuration) {
        digitalWrite(RELAY_PIN, LOW);
        Serial.println("💧 PUMP OFF");
        Serial.println("✅ Fire extinguished! Starting return
maneuver...");

        setState(TURN_BACK_FROM_FIRE,       "TURN_BACK_FROM_FIRE",
true);
      }
      break;

    case TURN_BACK_FROM_FIRE:
      motorStop();
      delay(200);

      executeReturnManeuver();
      setState(RETURNING, "RETURNING", true);
      break;

    case RETURNING:
```

```
      if (isAtStartPosition()) {
        motorStop();
        Serial.println("🏁 ARRIVED AT START POSITION");
        Serial.println("🔄 Final 180° turn at start...");

        setState(FINAL_TURN_AT_START,       "FINAL_TURN_AT_START",
true);
        break;
      }
      lineFollower();
      break;

    case FINAL_TURN_AT_START:
      motorStop();
      delay(200);

      rotateInPlace(100, 800);
      delay(200);

      Serial.println("✅ MISSION COMPLETED");
      setState(COMPLETED, "COMPLETED", true);
      break;

    case COMPLETED:
      motorStop();
      delay(500);

      Serial.println("🔄 Resetting to IDLE state...");
      setState(IDLE, "IDLE", true);
      break;
  }
}

/* ================ WiFi SETUP ================ */
void setupWiFi() {
  delay(10);
  Serial.println();
  Serial.print("📡 Connecting to WiFi: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.println("✅ WiFi connected!");
  Serial.print("📍 IP address: ");
  Serial.println(WiFi.localIP());
}

/* ================ MQTT RECONNECT ================ */
void reconnectMQTT() {
  while (!client.connected()) {
    Serial.print("🔌 Connecting to MQTT...");
```

```
    String clientId = "ESP32FireRobot-";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str())) {
      Serial.println(" connected!");
      client.subscribe(topic_fire_detected);

      client.publish(topic_status, "CONNECTED");
      client.publish(topic_status, stateToString(currentState));

      Serial.println("📩 Subscribed to: " +
String(topic_fire_detected));

      if (currentState == EXTINGUISHING) digitalWrite(RELAY_PIN,
HIGH);

    } else {
      Serial.print(" failed, rc=");
      Serial.print(client.state());
      Serial.println(" - retrying in 0.5 seconds");
      delay(500);
    }
  }
}

/* ================= MQTT CALLBACK ================= */
void mqttCallback(char* topic, byte* payload, unsigned int length)
{
  String message = "";
  for (unsigned int i = 0; i < length; i++) message +=
(char)payload[i];

  Serial.println("📨 MQTT Message received on topic: " +
String(topic));
  Serial.println("📝 Message: " + message);

  if (String(topic) == topic_fire_detected) {
    if (message == "1" || message == "ON" || message ==
"FIRE_DETECTED") {
      if (currentState == IDLE || currentState == COMPLETED) {
        Serial.println("🚨 FIRE ALERT RECEIVED FROM OPENCV!");
        Serial.println("🏃 Starting mission - Moving to fire...");

        setState(MOVING_TO_FIRE, "MOVING_TO_FIRE", true);
      } else {
        Serial.println("⚠️ Mission already in progress - ignoring
signal");
      }
    }
  }
}

/* ================= LINE FOLLOWER ================= */
void lineFollower() {
  int error = readLineError();
  lastLineErrCached = error;
```

```
  filteredError = alpha * filteredError + (1 - alpha) * error;
  float derivative = filteredError - lastError;
  lastError = filteredError;

  float output = Kp * filteredError + Kd * derivative;

  int leftSpeed  = baseSpeed - output * 20;
  int rightSpeed = baseSpeed + output * 20;

  driveForward(leftSpeed, rightSpeed);
}

/* ================= SENSOR ERROR READING ================= */
int readLineError() {
  int s1 = digitalRead(S1);
  int s2 = digitalRead(S2);
  int s3 = digitalRead(S3);
  int s4 = digitalRead(S4);
  int s5 = digitalRead(S5);

  static unsigned long lastPrint = 0;
  if (millis() - lastPrint > 500) {
    Serial.print("Sensors: ");
    Serial.print(s1); Serial.print(" ");
    Serial.print(s2); Serial.print(" ");
    Serial.print(s3); Serial.print(" ");
    Serial.print(s4); Serial.print(" ");
    Serial.println(s5);
    lastPrint = millis();
  }

  if (s3 == LOW) return 0;
  if (s2 == LOW && s3 == HIGH) return -1;
  if (s4 == LOW && s3 == HIGH) return 1;
  if (s1 == LOW) return -2;
  if (s5 == LOW) return 2;

  if (s1 == HIGH && s2 == HIGH && s3 == HIGH && s4 == HIGH && s5 ==
HIGH) {
    return (lastError > 0) ? 2 : -2;
  }

  return 0;
}

/* ================= MOTOR CONTROL ================= */
void motorStop() {
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void driveForward(int left, int right) {
  left = constrain(left, 0, maxSpeed);
```

```
  right = constrain(right, 0, maxSpeed);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

  analogWrite(ENA, left);
  analogWrite(ENB, right);
}

void rotateInPlace(int speed, int duration) {
  speed = constrain(speed, 0, maxSpeed);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);

  analogWrite(ENA, speed);
  analogWrite(ENB, speed);

  delay(duration);
  motorStop();
}

/* ================= RETURN MANEUVER ================= */
void executeReturnManeuver() {
  Serial.println("🔄 Executing 180° turn maneuver...");
  rotateInPlace(100, 800);
  delay(300);
}

/* ================= CHECK START POSITION ================= */
bool isAtStartPosition() {
  int s1 = digitalRead(S1);
  int s2 = digitalRead(S2);
  int s3 = digitalRead(S3);
  int s4 = digitalRead(S4);
  int s5 = digitalRead(S5);

  static int blackCounter = 0;

  if (s1 == LOW && s2 == LOW && s3 == LOW && s4 == LOW && s5 ==
LOW) {
    blackCounter++;
    if (blackCounter > 8) {
      blackCounter = 0;
      return true;
    }
  } else {
    blackCounter = 0;
  }
  return false;
}

/* ================= DEBUG INFO ================= */
void printDebugInfo() {
```

```
  Serial.print("State: ");
  Serial.print(stateToString(currentState));

  Serial.print(" | Mission: ");
  Serial.print(missionActive ? "ON" : "OFF");

  Serial.print(" | Flame: ");
  Serial.print(digitalRead(FLAME_PIN)  ==  LOW  ?  "DETECTED"  :
"NONE");

  Serial.print(" | Relay: ");
  Serial.print(digitalRead(RELAY_PIN) == HIGH ? "ON" : "OFF");

  Serial.print(" | LineErr: ");
  Serial.print(lastLineErrCached);

  Serial.print(" | LockRem(ms): ");
  if   (millis()   <   flameLockoutUntil)   Serial.print((unsigned
long)(flameLockoutUntil - millis()));
  else Serial.print(0);

  Serial.print(" | ExtRem(ms): ");
  if (currentState == EXTINGUISHING) {
    unsigned long elapsed = millis() - extinguishStartTime;
    Serial.print((elapsed >= extinguishDuration) ? 0 : (unsigned
long)(extinguishDuration - elapsed));
  } else {
    Serial.print("-");
  }

  Serial.println();
}
```

Kode python deteksi api:

```
import cv2
import numpy as np
import sys
import time
import paho.mqtt.client as mqtt

# ================== CONFIG ==================
USE_MQTT = True
BROKER = "10.83.74.17"
PORT = 1883

TOPIC_FIRE = "robot/fire/detected"
TOPIC_STATUS = "robot/status"

PAYLOAD_ON  = "ON"
PAYLOAD_OFF = "OFF"

lower_fire = np.array([0, 50, 180], dtype="uint8")
upper_fire = np.array([35, 255, 255], dtype="uint8")

FIRE_AREA_THRESHOLD = 100
DEBOUNCE_ON_HITS = 2
DEBOUNCE_OFF_HITS = 3
```

```
# ================== MQTT STATE ==================
client = None
mqtt_ready = False

last_robot_status = "-"       # status terakhir
prev_robot_status = "-"          # status sebelumnya (buat logic
CONNECTED)
last_status_ts = 0.0          # timestamp status terakhir
mission_latched = False       # True = mission sedang jalan, jangan
retrigger

# READY states asli
ROBOT_READY_STATES = {"IDLE", "COMPLETED"}
# BUSY states
ROBOT_BUSY_STATES = {
    "MOVING_TO_FIRE",
    "EXTINGUISHING",
    "TURN_BACK_FROM_FIRE",
    "RETURNING",
    "FINAL_TURN_AT_START"
}

STATUS_STALE_SEC  =  6.0    #  kalau  status  lama  banget,  anggap
unknown/busy

def now_s():
    return time.time()

def safe_decode(b: bytes) -> str:
    try:
        return b.decode(errors="ignore").strip()
    except Exception:
        return ""

# ================== MQTT CALLBACKS ==================
def on_connect(c, userdata, flags, rc):
    global mqtt_ready
    mqtt_ready = (rc == 0)
    if mqtt_ready:
        print(f"\n✅ MQTT: Connected to {BROKER}:{PORT}")
        c.subscribe(TOPIC_STATUS)
    else:
        print(f"\n❌ MQTT: Connect failed rc={rc}")

def on_disconnect(c, userdata, rc):
    global mqtt_ready
    mqtt_ready = False
    print(f"\n⚠️ MQTT: Disconnected rc={rc}")

def on_message(c, userdata, msg):
    global  last_robot_status,  prev_robot_status,  mission_latched,
last_status_ts

    if msg.topic != TOPIC_STATUS:
        return
```

```
    status = safe_decode(msg.payload)
    if not status:
        return

    prev_robot_status = last_robot_status
    last_robot_status = status
    last_status_ts = now_s()

    # latch logic
    if status in ROBOT_BUSY_STATES:
        mission_latched = True

    if status in ROBOT_READY_STATES:
        mission_latched = False

    # NOTE: status CONNECTED tidak langsung reset latch
    # karena bisa terjadi reconnect saat mission resume

# ================== MQTT HELPERS ==================
def mqtt_init():
    global client, mqtt_ready, USE_MQTT

    if not USE_MQTT:
        return

    try:
        client    =    mqtt.Client(client_id="OpenCV-FireDetector",
clean_session=True)
        client.on_connect = on_connect
        client.on_disconnect = on_disconnect
        client.on_message = on_message

        client.will_set(TOPIC_FIRE,   payload=PAYLOAD_OFF,   qos=0,
retain=False)

        client.connect(BROKER, PORT, 60)
        client.loop_start()

        t0 = now_s()
        while not mqtt_ready and (now_s() - t0) < 2.0:
            time.sleep(0.05)

        if not mqtt_ready:
            raise RuntimeError("MQTT connect timeout")

    except Exception as e:
        print(f"⚠️ MQTT tidak tersedia: {e}")
        USE_MQTT = False
        mqtt_ready = False
        client = None

def mqtt_publish(payload: str) -> bool:
    global USE_MQTT, mqtt_ready, client
    if not USE_MQTT:
        return False

    if (not mqtt_ready) or (client is None):
        mqtt_init()
```

```
        if not mqtt_ready:
            return False

    try:
        info = client.publish(TOPIC_FIRE, payload, qos=0,
retain=False)
        info.wait_for_publish(timeout=1.0)
        return True
    except Exception as e:
        print(f"\n⚠️ MQTT publish gagal: {e}")
        mqtt_ready = False
        return False

def robot_status_stale() -> bool:
    if last_status_ts == 0:
        return True
    return (now_s() - last_status_ts) > STATUS_STALE_SEC

def connected_can_be_ready() -> bool:
    """
    ✅ CONNECTED dianggap READY hanya jika aman:
    - mission tidak sedang latched
    - DAN (sebelumnya memang READY) ATAU (belum pernah punya status
valid)
    Dengan begitu reconnect di tengah mission tidak bikin retrigger.
    """
    if mission_latched:
        return False

    if prev_robot_status in ROBOT_READY_STATES:
        return True

    if prev_robot_status == "-" and last_robot_status ==
"CONNECTED":
        # baru mulai hidup, status belum kebaca selain CONNECTED
        return True

    return False

def robot_is_ready() -> bool:
    if robot_status_stale():
        return False

    if last_robot_status in ROBOT_READY_STATES and (not
mission_latched):
        return True

    # ✅ bridge: CONNECTED
    if last_robot_status == "CONNECTED" and
connected_can_be_ready():
        return True

    return False

mqtt_init()

# ================== CAMERA ==================
```

```
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("❌ Kamera tidak terdeteksi!")
    sys.exit()

print("SISTEM PENDETEKSI API + MQTT (ESP32)")
print("Tekan 'd' untuk toggle mask debug, 'q' untuk keluar.")
print("-----------------------------------")

show_mask = False
fire_sent = "OFF"

on_hits = 0
off_hits = 0
fire_state = False

MIN_ON_INTERVAL = 1.0
last_on_publish_t = 0.0

try:
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.resize(frame, (640, 480))
        blur = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

        mask = cv2.inRange(hsv, lower_fire, upper_fire)
        mask = cv2.erode(mask, None, iterations=1)
        mask = cv2.dilate(mask, None, iterations=2)

        _, _, v = cv2.split(hsv)
        max_brightness = int(np.max(v))
        fire_area = int(cv2.countNonZero(mask))

        raw_fire = fire_area > FIRE_AREA_THRESHOLD

        # debounce
        if raw_fire:
            on_hits += 1
            off_hits = 0
        else:
            off_hits += 1
            on_hits = 0

        if on_hits >= DEBOUNCE_ON_HITS:
            fire_state = True
        elif off_hits >= DEBOUNCE_OFF_HITS:
            fire_state = False

        desired = "ON" if fire_state else "OFF"

        # publish ON hanya jika robot siap
        if desired == "ON":
            can_send   =   robot_is_ready()   and   ((now_s()   -
last_on_publish_t) >= MIN_ON_INTERVAL)
```

```
            if can_send and fire_sent != "ON":
                ok = mqtt_publish(PAYLOAD_ON)
                fire_sent = "ON"
                last_on_publish_t = now_s()
                print(f"\n📡 PUBLISH {TOPIC_FIRE} = {PAYLOAD_ON}
({'OK' if ok else 'FAIL'})")
        else:
            fire_sent = "OFF"

        # bounding box
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours:
            if cv2.contourArea(contour) > FIRE_AREA_THRESHOLD:
                x, y, w, h = cv2.boundingRect(contour)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0,
255), 2)
                cv2.putText(frame, "API!", (x, y - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0,
255), 2)

        status_txt = "⚠️ KEBAKARAN! ⚠️" if fire_state else "AMAN"
        if robot_status_stale():
            gate_txt = "STALE"
        else:
            gate_txt = "READY" if robot_is_ready() else "BUSY"

        sys.stdout.write(
            f"\r[MONITOR]       Bright:{max_brightness:<3}       |
Area:{fire_area:<6} | CV:{status_txt:<14} "
            f"|   MQTT:{'ON'   if   mqtt_ready   else   'OFF'}   |
ROBOT:{last_robot_status:<18} | GATE:{gate_txt:<5}   "
        )
        sys.stdout.flush()

        cv2.imshow("Monitor Kamera", frame)
        if show_mask:
            cv2.imshow("Mask (Debug)", mask)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            print("\nProgram selesai.")
            break
        elif key == ord('d'):
            show_mask = not show_mask
            if not show_mask:
                cv2.destroyWindow("Mask (Debug)")

finally:
    mqtt_publish(PAYLOAD_OFF)

    cap.release()
    cv2.destroyAllWindows()

    if client:
        try:
            client.loop_stop()
        except:
```

```
            pass
        try:
            client.disconnect()
        except:
            pass

print("🧯 Selesai.")
```

HASIL DAN KESIMPULAN

Progress

Pengerjaan Line Follower dengan system calling api dengan MQTT mosquito menggunakan webcam Laptop dikerjakan selama 4 minggu dengan runtutan kegiatan yang kami lakukan Sebagai berikut:

1. Pembuatan rancangan line follower
2. Pembuatan Diagram input
3. Pembuatan skematic diagram
4. Pembuatan Desain Line Follower
5. Pemilihan dan pembelian bahan-bahan
6. Perancangan (soldering, wiring, QC)
7. Mengembangkan dan mengupload program kedalam esp
8. Trail dan Error line follower agar dapat mengikuti garis dan dapat memadamkan api.
9. Test

Kendala dan Solusi

Selama pengerjaan tugas line follower ini kami mengalami banyak masalah dan juga selama itu juga kami mendapatkan Solusi dari permasalahan yang kami alami. Diantaranya:

1. **Penggunaan 4 motor**

Pada awal konsep dan rancangan yang sudah dibuat kami hanya menggunakan 2 motor. Namun, terjadi kendala pada bagian 2 ban yang diletakkan motor diatasnya, Dimana karena dinamisnya kondisi tanah/tarck yang akan digunakan dan desain body yang cukup besar dan tidak stabil membuat Line follower

kesulitan Ketika masuk ke area tikungan. Sehingga kami menggunakan 4 motor untuk mrngatasi ketidak stabilan tersebut.

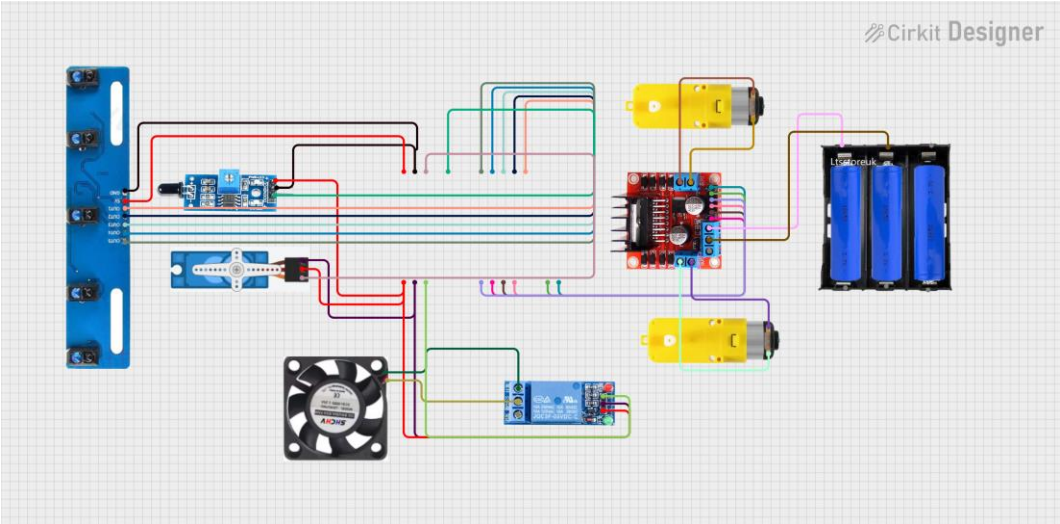
2. **Menghilangkan Servo**

   Servo dalam konsep yang kami buat memang berperan penting dalam mengetahui keberadaan akurat posisi api. Namun, Ketika kami sudah melakukan banyakanya trail dan error kami mendapati bahwasannya dengan banyaknya beban yang di tamping oleh esp 32 sehingga servo terkadang salah kondisi yang menyebabkan pergerakkan line follower terdelay dan juga kondisi jalan line follower yang masih belum sempurna dalam mendeteksi garis. Setelah servo dihilangkan, kami melakukan trail dan error Kembali dan mendapati hasil yang bisa dibilang cukup dengan menghilangkan keberadaan servo.

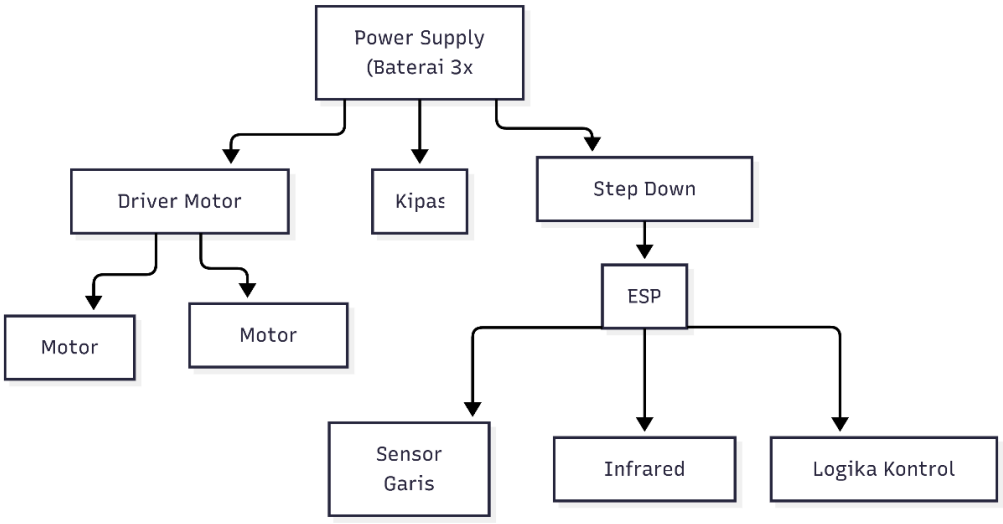
3. **Mengubah pompa air menjadi kipas**

   Ketika mendekati hari pelaksaan test kami menyadari bahwa beban yang diberikan pompa ke esp terlalu besar, Dimana Ketika pompa berhasil menyala setelah mendeteksi api, riak yang diberikan setelahnya memberikan feedback yang kuat sehingga memutus MQTT dan melakukan disconnecting secara paksa, yang mana hal ini sangat menggangu proses untuk Kembali ke home base karena harus dilakukan koneksi MQTT Kembali agar terhubung dengan Line Folower. Dan setelah diganti dengan kipas memang benar, karena kipas tidak memberikan beban dan riak yang sama besar dengan pompa air sehingga MQTT tidak terputus dan line follower dapat berjalan dengan baik.

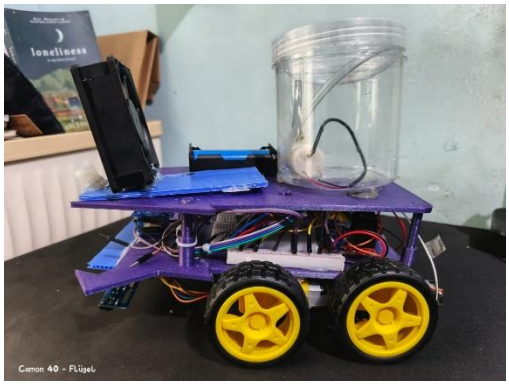
Perubahan wiring diagram (Final)

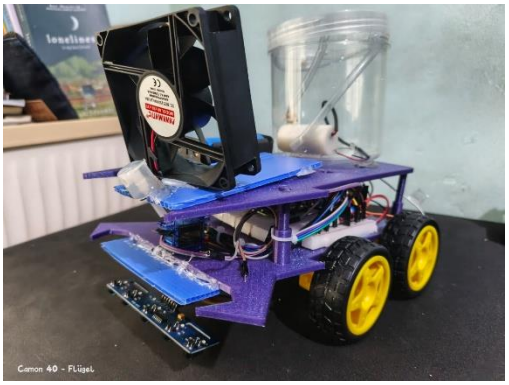


Perubahan Blok diagram input (Final)



Desain hasil





Vidio Dokumentasi

Untuk melihat hasil dari projek ini dapat di lihat di link drive berikut:

https://drive.google.com/drive/folders/1Zl2bPBADgyV54n89UK4QG29WAw35GJoh

Kesimpulan

Dari hasil yang kami dapati pada hari Sabtu 20 Desember 2025 melalui Test, kami mendapati hasil yang cukup baik, meskipun banyak dari sisi konsep line follower yang sudah banyak diganti baik itu spare part maupun kodingan. Juga melihat dari performa Ketika berjalan diatas line yang masih sangat perlu untuk dilakukan perbaikan dan pengembangan kedepannya. uhuy