MECHATRONICS SYSTEM INTEGRATION MCTA3203

Smart Surveillance System with Motorized Camera Base (ver 1.0)

EXPERIMENT 6

DATE: 10TH November 2025
SECTION: 2
GROUP: 18
SEMESTER 1, 2025/2026

NO	NAME	MATRIC NO
1.	SHAMSUL HAKIMEE BIN SHAMSUL HAIREE	2315027
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	NUREL HUMAIRAH BINTI MUHAMMAD FIRDAUS	2315680

DATE OF SUBMISSION :
Monday, 17TH November 2025

Abstract

This experiment focuses on developing a smart surveillance system using the ESP32-CAM module by completing three core tasks: integrating a pushbutton for manual panning control, enabling built-in face detection features, and adding vertical camera movement using a second servo motor. The implementation combined wireless video streaming, sensor processing, and PWM-based actuator control to create an IoT-enabled camera platform capable of horizontal and vertical motion. Live video was streamed over Wi-Fi, while additional control logic allowed the system to respond to user input or visual triggers. Through these tasks, the experiment demonstrated how microcontrollers, servo actuators, and basic vision-processing capabilities can be integrated into a functional surveillance prototype. The results highlight the potential of ESP32-CAM systems for low-cost monitoring applications and show key mechatronic principles such as real-time processing, hardware and software interfacing, and embedded system integration.

Table of Contents

1.0 Introduction

Smart surveillance systems represent a growing area within modern mechatronics, as they merge embedded computing, wireless communication, and automated actuation into a single functional platform. This experiment introduced these concepts through the ESP32-CAM, a compact microcontroller equipped with an OV2640 camera sensor and Wi-Fi capability. The module allows real-time video streaming, remote monitoring, and onboard processing. Making it an accessible tool for understanding the fundamentals of IoT-based vision systems.

The practical work in this lab centered on extending the ESP32-CAM beyond basic streaming by integrating actuation and simple user-interaction features. Servo motors were used to provide mechanical panning, enabling the camera to sweep horizontally or vertically through PWM control. A pushbutton was added to allow manual initiation of motion, demonstrating digital input handling and interrupt-based or logic-based control. In addition, the system's built-in face detection algorithm was explored, illustrating how embedded processors can perform lightweight image-processing tasks. By completing Tasks 1, 2, and 3, the experiment provided a hands-on opportunity to apply mechatronic principles in a realistic surveillance scenario, reinforcing the importance of system integration and responsive design in modern embedded applications.

2.0 Materials and Equipments

Materials and components that were used in the experiment :

2.1 Electronic Components

- ESP32-CAM module (with OV2640 camera)
- USB-to-Serial programmer
- Servo motors (x2)
- Pushbutton switch
- 5V external power supply
- Jumper wires
- Breadboard

2.2 Equipment

- Arduino IDE Software
- USB Cable
- Wi-Fi network for ESP32-CAM connection

3.0 Experimental Setup

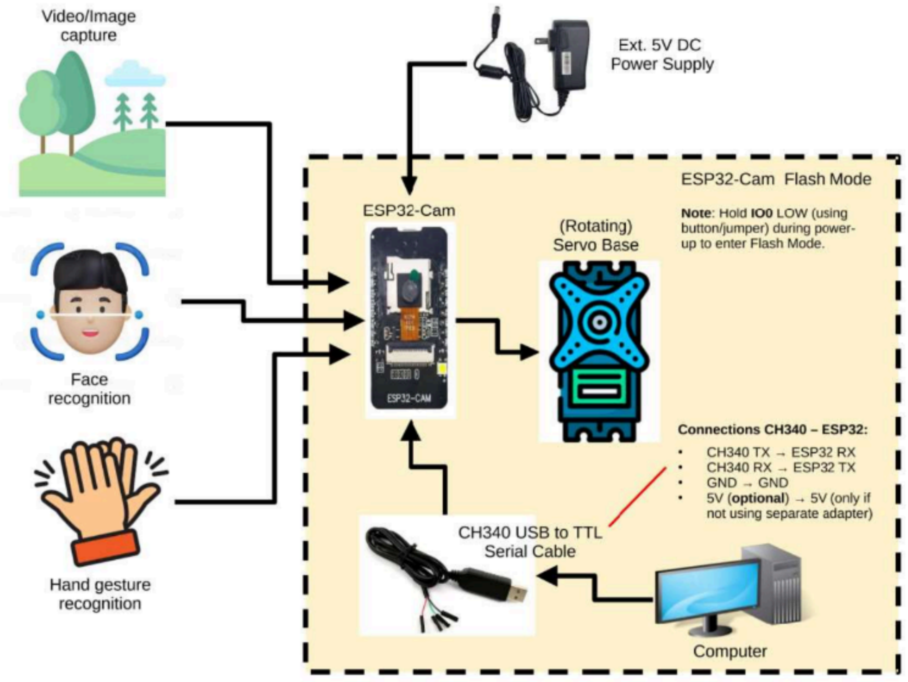

Video/Image capture
Ext. 5V DC Power Supply
ESP32-Cam Flash Mode
Note: Hold IO0 LOW (using button/jumper) during power-up to enter Flash Mode.
ESP32-Cam
(Rotating) Servo Base
Face recognition
Connections CH340 – ESP32:
CH340 TX → ESP32 RX
CH340 RX → ESP32 TX
GND → GND
5V (optional) → 5V (only if not using separate adapter)
Hand gesture recognition
CH340 USB to TTL Serial Cable
Computer

Figure 3.1 System Overview

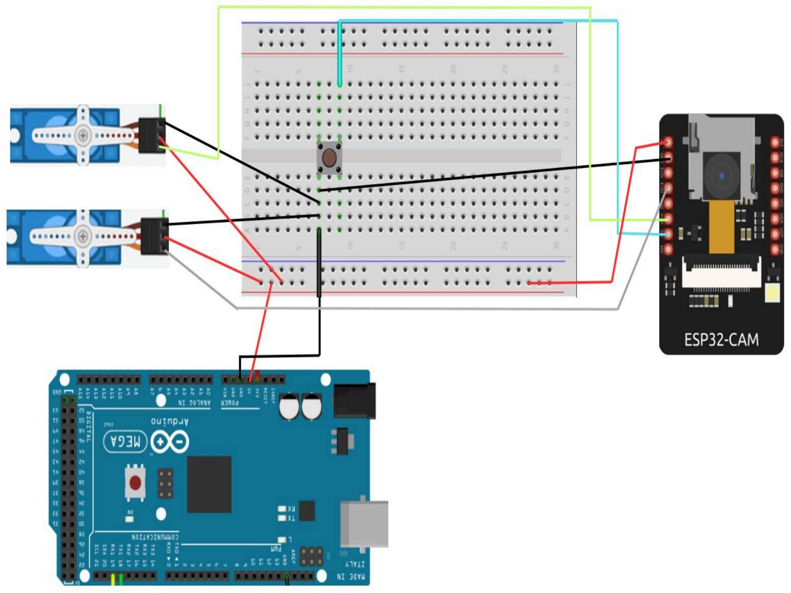


Figure 3.1.1 Schematic Diagram Smart Surveillance System

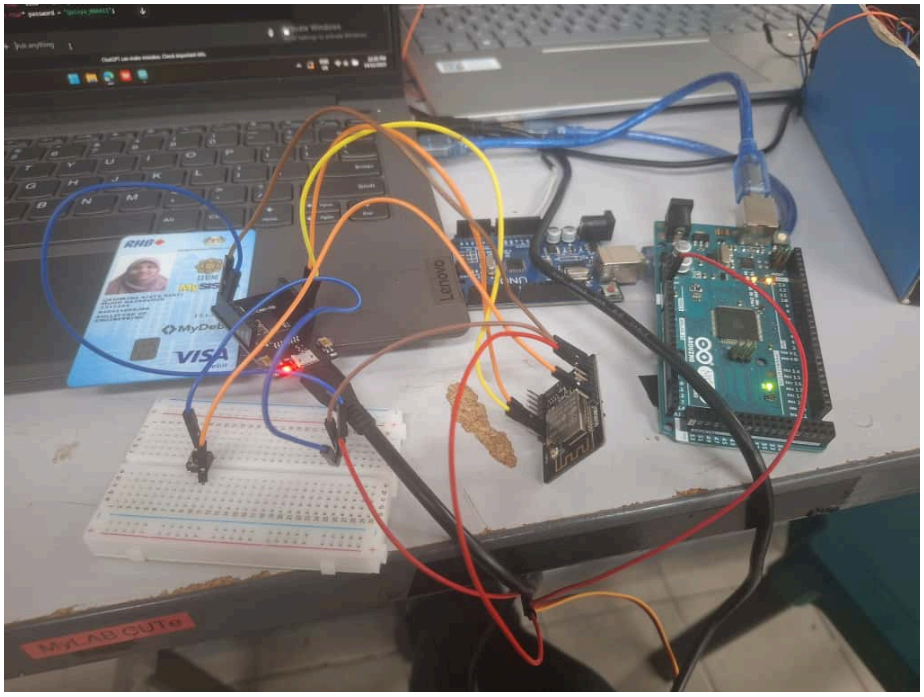


Figure 3.1.2 Schematic Diagram Smart Surveillance System

3.1 Circuit Setup

The circuit was assembled to integrate the ESP32-CAM module with two servo motors and a pushbutton, enabling manual panning and vertical movement. The setup also included an ESB-to-Serial programmer for uploading firmware and establishing initial communication with the microcontroller.

The circuit connection for Task 1 was made as follows:

- Servo signal wire was connected to GPIO 12 on the ESP32-CAM for PWM control.
- Servo 5V was connected to a 5V external power supply to ensure stable connection.
- Serco GND was connected to the common GND shared with the ESP32 board.
- Pushbutton one leg was connected to GPIO 2, configured as an input with an internal pull-up resistor.
- Pushbutton other leg was connected to GND, allowing GPIO 2 to read LOW when pressed.
- Common ground was shared among the ESP32 board, servo motor, and external 5V supply to maintain a consistent reference.

The circuit connection for Tasks 2 and 3 was made as follows:

- ESP32-CAM 5V pin was connected to a stable 5V external supply.
- ESP32-CAM GND pin was connected to the common ground shared by all components.
- Pan servo signal wire was connected to GPIO 14 for horizontal face-tracking movement.
- Tilt servo signal wire was connected to GPIO 15 for vertical face-tracking movement.
- Both servo 5V wires were connected to the external 5V supply.
- Both servo GND wires were connected to the common GND.
- Dummy Servo 1 signal wire was connected to GPIO 12, required by the ESP32Servo library to avoid timer conflicts with the camera.
- Dummy Servo 2 signal wire was connected to GPIO 13, also used to reserve PWM channels for proper servo operation.
- (Dummy servos can be real servos or left unconnected; only the signal pins matter.)
- FTDI/USB-to-Serial RX was connected to U0T (GPIO 1) on the ESP32-CAM during programming.
- FTDI/USB-to-Serial TX was connected to U0R (GPIO 3).
- FTDI 5V was connected to ESP32-CAM 5V only during code upload.
- FTDI GND was connected to ESP32-CAM GND.
- IO0 was connected to GND during flashing and disconnected afterwards.
- A shared ground was maintained between the ESP3

4.0 Methodology

4.1 Implementation and Testing

TASK 1:

This task involved implementing a simple panning mechanism controlled by a pushbutton. The servo motor was connected to GPIO 12, while the pushbutton input was wired to GPIO 2 with an internal pull-up resistor. The ESP32Servo library was used to generate stable signals for the servo.

The implementation began by configuring the button as input and initializing the servo at a neutral position of 0°. The system monitored the button state continuously; when the button was pressed, the servo increased or decreased its angle in increments of 5°. Boundary conditions (0° and 180°) were applied to prevent mechanical over-rotation.

Testing was carried out by repeatedly pressing the pushbutton to verify the servo responded correctly at each button press, confirming that manual panning control was functional.

TASK 2&3:

Tasks 2 and 3 required integrating face detection, dual-axis servo motion, and video streaming. The ESP32-CAM was programmed to capture frames using the OV2640 sensor, convert them to RGB format, and run the MTMN face detection model. Two servo motors were attached to GPIO 14 (pan) and GPIO 15 (tilt), while two dummy servos were linked to GPIO 12 and 13 to reserve LEDC timers for proper PWM operation.

The system hosted a local web server on port 80 for the control interface and port 81 for MJPEG streaming. When a frame was captured, it was passed through `fmt2rgb888()` to generate data suitable for detection. If a face was identified, the bounding box coordinates were processed to determine the horizontal and vertical centers of the face. This data was sent to the control algorithm to adjust servo angles accordingly.

Testing involved powering the ESP32-CAM with a stable 5V supply and accessing the video stream through a browser. The face tracking feature was verified by moving in front of the camera and observing whether the pan and tilt servos followed the subject smoothly. The system was also tested for stabilization behavior when no face was detected; the servos performed periodic scanning until a face reappeared.

4.2 Control Algorithm

TASK 1:

The control algorithm for this task followed a simple event-based structure:

1. Read the pushbutton state (LOW=pressed).
2. If pressed, adjust the servo angle by ±5° depending on the direction flag.
3. If the angle reaches 180°, switch direction to reverse.
4. If the angle reached 0, switch direction to forward.
5. Write the updated angle to the servo output.

This algorithm ensured predictable oscillating motion initiated only when the user interacted with the system.

TASK 2&3:

The algorithm for the face detection system used the following steps:

1. Capture a frame from the camera sensor.
2. Convert the captured frame to RGB888 format for the detection engine.
3. Run the MTMN detector to identify face bounding boxes.
4. If a face is detected,
    - Calculate the center of the face (x,y).
    - Compare it with the ideal frame center (160 for horizontal, 120 for vertical).
    - Adjust posH and posV proportionally using an error-based correction.
    - Apply angle limits to avoid mechanical overrotation.
    - Update both servos with panServo.write(posH) and tiltServo.write(posV).
5. If no face is detected several times:
    - Increment counter
    - Perform slow scanning
    - Reset tilt to default values
    - Restart face tracking when a face reappears.

4.3 Code Used

TASK 1:

```
#include <ESP32Servo.h>

#define BUTTON_PIN 2
#define SERVO_PIN 12

Servo myservo;

int servoPos = 0;           // Current servo angle
bool movingForward = true;  // Direction of movement

void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.println("Fast Bounce Servo Test Start");

  pinMode(BUTTON_PIN, INPUT_PULLUP);

  myservo.setPeriodHertz(50);
  myservo.attach(SERVO_PIN, 1000, 2000);

  myservo.write(servoPos);
}

void loop() {
  bool state = digitalRead(BUTTON_PIN);

  if (state == LOW) {           // Button pressed
    // Move servo faster (e.g., 5° per loop)
    if (movingForward) {
      servoPos += 5;
      if (servoPos >= 180) {
        servoPos = 180;
        movingForward = false;  // Reverse direction at max
      }
    } else {
      servoPos -= 5;
      if (servoPos <= 0) {
        servoPos = 0;
        movingForward = true;   // Reverse direction at min
      }
    }

    myservo.write(servoPos);
    Serial.print("Servo angle: ");
    Serial.println(servoPos);
    delay(50); // Small delay for smooth movement
  }
}
```

TASK 2&3:

```
        //ESP32 camera: face detection
        //NOW WITH LIVE STREAMING WEB SERVER
        //It can detect a human face with the model MTMN, move servos, and stream
the video.

        //ESP32 Camera Driver, Camera OV2640/1600 x 1200/Len Size:1/4"

        // --- LIBRARIES ---
        #include "esp_camera.h"
        #include <WiFi.h>            // --- NEW: Added for web server
        #include "esp_http_server.h"  // --- NEW: Added for web server
        #include "fd_forward.h"
        #include "img_converters.h"   // --- NEW: Added for image conversion
        #include "soc/soc.h"          // disable brownout problems
        #include "soc/rtc_cntl_reg.h"  // disable brownout problems
        #include <ESP32Servo.h>

        //***** Servo def
        #define DUMMY_SERVO1_PIN 12     //We need to create 2 dummy servos.
        #define DUMMY_SERVO2_PIN 13     //So that ESP32Servo library does not
interfere with pwm channel and timer used by esp32 camera.

        #define PAN_PIN 14
        #define TILT_PIN 15
```

```
Servo dummyServo1;
Servo dummyServo2;
Servo panServo;
Servo tiltServo;
// Published values for SG90 servos; adjust if needed
int minUs = 500;
int maxUs = 2400;
//**** Servo the end

int  posH;  //Position Panoram
int  posV;  //Position Tilt
int altPosH;
bool toNul; //servo moves to Null

// --- 3 THINGS TO CHANGE ---

// 1. & 2. Your Wi-Fi Credentials
const char* ssid = "Archetype Kismees";
const char* password = "raisin2004";

// 3. Your Camera Model
#define CAMERA_MODEL_AI_THINKER

// --- END OF CHANGES ---

#include "camera_pins.h"

// --- NEW: Web Server Handles ---
httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

// --- Recognition Variables ---
static mtmn_config_t mtmn_config = {0};   // Detection config

int noDetection = 0;
  //define a variable that will count how many times we have not detected
faces.
  //We will initialize it with the value zero and increment it every time faces
are not detected in a frame.
```

```
        // --- FIX: Add Function Prototype for initCamera ---
        bool initCamera();

        //++++++++++++++++++++++++++++++++++++++++++++++          function
called draw_face_boxes that is used to display a box around a detected face
        // --- MODIFIED: This function now just moves the servos ---
        static  void  draw_face_boxes(dl_matrix3du_t  *image_matrix,  box_array_t
*boxes)
        {
         int x, y, w, h, i, half_width, half_height;

         for (i = 0; i < boxes->len; i++) {

          // --- FIX: Calculate x,y,w,h first ---
          x = (int)boxes->box[i].box_p[0];
          y = (int)boxes->box[i].box_p[1];
          w = (int)boxes->box[i].box_p[2] - x;
          h = (int)boxes->box[i].box_p[3] - y;

          // finding face centre...
          half_width = w / 2;

          // --- FIX: Define face_center_pan. ---
            int  face_center_pan  =  x  +  half_width;  //  image  frame  face  centre  x
co-ordinate

          altPosH=posH;
          half_height = h / 2;

          // --- SPEED CONTROL ---
          // To make it faster, change the /1 to /2 or /1
          // To make it slower, change it to /3 or /4
          posH =posH + (160 - face_center_pan)/1; //was 4,

          if ((altPosH-posH)>0)
          {
           //+++++++++
           toNul=false;  //// servo move to 0 grad
          }
          else
          {
            toNul=true; // servo move to plus 180
          }

          altPosH=posH;

          if (posH>170)
          {
           //+++++++++
           posH=170;
          }
          if (posH<20)
           {
           //+++++++++
           posH=20;
          }

          Serial.printf("Center detected at %d dots\n", face_center_pan);
          panServo.write(posH);

         //
         Serial.printf("H%d \n", posH);

                   //+++++++++++++++++++++++++    Center    TILT    is    120
+++++++++++++++++++++++++++++++++++++++++++++++++++++
            int  face_center_tilt  =  y  +  half_height;   //  image  frame  face  centre  y
co-ordinate

          // --- SPEED CONTROL ---
          posV =posV - (120 - face_center_tilt)/1;  //

          if (posV>130)
          {
           posV=130;   //LIMIT UP;
            }
          if (posV<50)
           {
           posV=50;    //LIMIT DOWN
           }
```

```
     tiltServo.write(posV);
    Serial.printf("V%d \n", posV);
  }
}


//
==============================================================
==========
// === NEW: WEB SERVER CODE ===
//
==============================================================
==========

// --- NEW: Simple HTML Page ---
const char* INDEX_HTML = R"EOF(
<!DOCTYPE html>
<html>
<head>
  <title>ESP32-CAM Face Tracker</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin: 0; padding:
0; background-color: #f4f4f4; }
    h1 { background-color: #007bff; color: white; padding: 20px; margin: 0; }
    #stream-container {
      margin: 20px auto;
      border: 5px solid black;
      width: 800px;
      max-width: 90%;
    }
    img { width: 100%; height: auto; display: block; }
  </style>
</head>
<body>
  <h1>ESP32-CAM Face Tracker</h1>
  <div id="stream-container">
    <!-- The src is now blank. JavaScript will fill it in. -->
    <img src="" id="stream">
  </div>

   <!-- --- FIX: This script tells the browser to look for the stream on port 81
--- -->
  <script>
    document.getElementById("stream").src = window.location.protocol + "//"
+ window.location.hostname + ":81/stream";
  </script>
</body>
</html>
)EOF";

// --- NEW: Handler for the live stream ---
static esp_err_t stream_handler(httpd_req_t *req){
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];

  // --- MJPEG Stream Header ---
                  res           =          httpd_resp_set_type(req,
"multipart/x-mixed-replace;boundary=--boundarydonotcross");
  if(res != ESP_OK){
    return res;
  }
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

  // --- CORE LOGIC LOOP ---
  // This loop runs continuously, sending frames to the browser
  while(true){

    // --- 1. Get a frame from the camera ---
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Frame buffer not available");
      res = ESP_FAIL;
      break;
    }
```

```
        // --- 2. Allocate matrix and convert image for detection ---
        // (This is your code from the old loop())
            dl_matrix3du_t  *image_matrix  =  dl_matrix3du_alloc(1,  fb->width,
fb->height, 3);
        if (!image_matrix) {
          Serial.println("dl_matrix3du_alloc failed");
          esp_camera_fb_return(fb);
          continue; // Skip this frame
        }

        if(!fmt2rgb888(fb->buf, fb->len, fb->format, image_matrix->item)){
            Serial.println("fmt2rgb888 failed");
            dl_matrix3du_free(image_matrix);
            esp_camera_fb_return(fb);
            continue; // Skip this frame
        }

        // --- 3. Run Face Detection & Servo Logic ---
        // (This is your code from the old loop())
        box_array_t *boxes = face_detect(image_matrix, &mtmn_config);

        if (boxes != NULL) {
          noDetection = 0;
          Serial.printf("Faces detected at %d \n", millis());
          draw_face_boxes(image_matrix, boxes); // This moves the servos
          dl_lib_free(boxes->score);
          dl_lib_free(boxes->box);
          dl_lib_free(boxes->landmark);
          dl_lib_free(boxes);
        }
        else
        {
          noDetection = noDetection+1;
          Serial.printf("Faces not not detected %d times \n", noDetection);
          switch(noDetection){
            case 10:  Serial.printf("Case10 Nodetected at %d \n", millis()); break;
            case 40: {
                   Serial.printf("Case40 Nodetected at %d \n", millis());
                   noDetection = 0; // stop
                 }
```

```
                break;
           default:  break;
         }
         if (toNul)
         {
            posH += 2; // --- SPEED INCREASED ---
            if (posH>170) {
              posH=170;
              posV=50;
              tiltServo.write(posV);
              toNul=false;
            }
            panServo.write(posH);
          }
          else
          {
            posH -= 2; // --- SPEED INCREASED ---
            if (posH<10) {
              posH=10;
              posV=90;
              tiltServo.write(posV);
              toNul=true;
            }
            panServo.write(posH);
          }
        }

        // --- 4. Convert the final frame (with box) to JPEG ---
        // We convert the RGB matrix *back* to a JPEG to send to the browser
             if(!fmt2jpg(image_matrix->item,   fb->width*fb->height*3,   fb->width,
fb->height, PIXFORMAT_RGB888, 90, &_jpg_buf, &_jpg_buf_len)){
           Serial.println("fmt2jpg failed");
           dl_matrix3du_free(image_matrix);
           esp_camera_fb_return(fb);
           continue;
        }

        // --- 5. Free resources from detection ---
        dl_matrix3du_free(image_matrix);
```

```
          esp_camera_fb_return(fb); // IMPORTANT: return the original frame
buffer

        // --- 6. Send the JPEG frame to the browser ---
        if(res == ESP_OK){
          res = httpd_resp_send_chunk(req, "--boundarydonotcross\r\n", 22);
        }
        if(res == ESP_OK){
            sprintf((char *)part_buf, "Content-Type: image/jpeg\r\nContent-Length:
%u\r\n\r\n", _jpg_buf_len);
            res = httpd_resp_send_chunk(req, (const char *)part_buf, strlen((char
*)part_buf));
        }
        if(res == ESP_OK){
          res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
        }
        if(res == ESP_OK){
          res = httpd_resp_send_chunk(req, "\r\n", 2);
        }

        // --- 7. Free the JPEG buffer ---
        free(_jpg_buf);
        _jpg_buf = NULL;

        if(res != ESP_OK){
          break; // Stop streaming if an error occurs
        }

        delay(10); // Small delay to not overwhelm the system
      } // --- End of while(true) loop ---

      return res;
    }

    // --- NEW: Handler for the HTML Page (v1.0.6 compatible) ---
    static int index_handler(httpd_req_t *req){
        httpd_resp_set_type(req, "text/html");
        httpd_resp_set_hdr(req, "Content-Encoding", "identity");
        return httpd_resp_send(req, INDEX_HTML, strlen(INDEX_HTML));
    }
```

```
// --- NEW: Function to start the web servers ---
void startCameraServer(){

  // --- FIX: Use two separate config structs for safety ---
  httpd_config_t config_main = HTTPD_DEFAULT_CONFIG();

  httpd_uri_t index_uri = {
    .uri      = "/",
    .method   = HTTP_GET,
    .handler  = index_handler,
    .user_ctx  = NULL
  };

  httpd_uri_t stream_uri = {
    .uri      = "/stream",
    .method   = HTTP_GET,
    .handler  = stream_handler,
    .user_ctx  = NULL
  };

  // Start the server for port 80 (HTML page)
  if (httpd_start(&camera_httpd, &config_main) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
  }

  // Start the server for port 81 (Stream)
   httpd_config_t config_stream = HTTPD_DEFAULT_CONFIG(); // Create a
NEW config
  config_stream.server_port = 81;
  config_stream.ctrl_port = 32769; // Use a different control port

  if (httpd_start(&stream_httpd, &config_stream) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }
}

//
=================================================================
==========
// === SETUP FUNCTION ===
//
===============================================================
==========
void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG,   0);   //   disable
brownout detector

  Serial.begin(115200);

  // --- FIX: Attach dummy servos *before* camera init ---
  dummyServo1.attach(DUMMY_SERVO1_PIN);
  dummyServo2.attach(DUMMY_SERVO2_PIN);

  if (!initCamera()) {
    Serial.printf("Failed to initialize camera...");
    return;
  }

  posH=90;
  posV=90;
  toNul = true; // Start by searching

  Serial.println("Face_FollowMe24-1");

  mtmn_config = mtmn_init_config();
  //****** Servo
  panServo.setPeriodHertz(50);      // Standard 50hz servo
  tiltServo.setPeriodHertz(50);      // Standard 50hz servo

  panServo.attach(PAN_PIN, minUs, maxUs);
  tiltServo.attach(TILT_PIN, minUs, maxUs);
  panServo.write(posH); //pan angle 90 start
  tiltServo.write(posV);  //tilt angle 90 start

  // --- NEW: Connect to WiFi ---
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

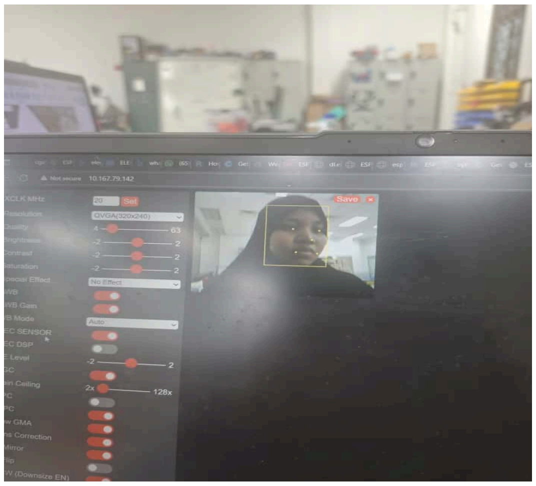
```
  }
  Serial.println("");
  Serial.println("WiFi connected!");

  // --- NEW: Start Web Server ---
  startCameraServer();

  Serial.print("Go to this IP in your browser: http://");
  Serial.println(WiFi.localIP());
}


//
======================================================================
==========
// === LOOP FUNCTION ===
//
======================================================================
==========
void loop() {
  // Everything is handled by the web server (stream_handler)
  // The loop can be empty.
  delay(10000);
}

//
======================================================================
==========
// === CAMERA INIT FUNCTION ===
//
======================================================================
==========
bool initCamera() {
  camera_config_t config;

  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;

  // --- FIX: Remove the bad line ---
  config.pin_href = HREF_GPIO_NUM; // Corrected

  // --- FINAL FIX (v1.0.6): Use the correct 1.0.6 board version pin names ---
  config.pin_sscb_sda = SIOD_GPIO_NUM; // Was config.pin_sccb_sda
  config.pin_sscb_scl = SIOC_GPIO_NUM; // Was config.pin_sccb_scl

  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;

  // --- MODIFIED: Use a raw format for face detection ---
  config.pixel_format = PIXFORMAT_YUV422; // Was JPEG
  config.frame_size = FRAMESIZE_QVGA;     // 320x240
  config.jpeg_quality = 10;
  config.fb_count = 1;

  // --- FINAL TYPO FIX ---
  esp_err_t result = esp_camera_init(&config); // Was esp_camera_.init

  if (result != ESP_OK) {
    return false;
  }

  return true;
}


//new tab
//
// This file contains the pin definitions for the AI-THINKER ESP32-CAM
// as defined in the main sketch (esp32_pan_tilt_tracker.ino)
//

// Ensure this file is only included once
#ifndef _CAMERA_PINS_H_
#define _CAMERA_PINS_H_

#if defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1 // -1 = Not used
  #define XCLK_GPIO_NUM      0

  #define SIOD_GPIO_NUM     26 // SCCB SDA
  #define SIOC_GPIO_NUM     27 // SCCB SCL

  #define Y9_GPIO_NUM       35 // D7
  #define Y8_GPIO_NUM       34 // D6
  #define Y7_GPIO_NUM       39 // D5
  #define Y6_GPIO_NUM       36 // D4
  #define Y5_GPIO_NUM       21 // D3
  #define Y4_GPIO_NUM       19 // D2
  #define Y3_GPIO_NUM       18 // D1
  #define Y2_GPIO_NUM        5 // D0

  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22

#else
    #error "Unknown camera model selected. Please check your #define
CAMERA_MODEL_AI_THINKER in the main .ino file."
#endif

#endif // _CAMERA_PINS_H_
```

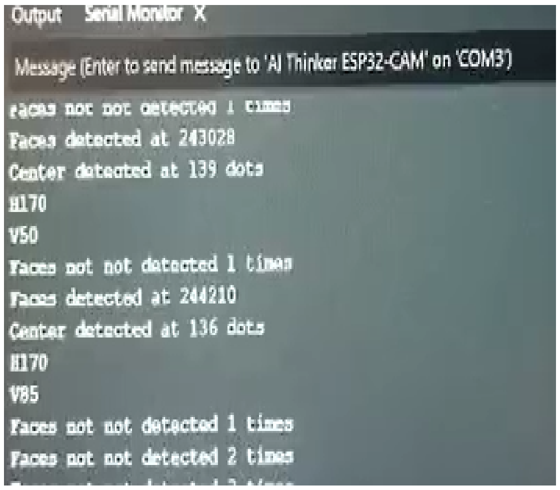
5.0 Result

The implemented system successfully performed manual control, face detection, and pan-tilt tracking using the ESP32-CAM. The pushbutton input reliably triggered servo movement, with the motor transitioning across its range and correctly reversing direction at both limits. Input readings from GPIO 2 remained stable due to the internal pull-up configuration, and the servo on GPIO 12 responded to each press.

For the face tracking portion, the ESP32-CAM detected human faces accurately within its field of view using the MTMN model. Once a face was identified, the system calculated its senate coordinates and adjusted both servos accordingly. When no face was detected, the system entered a scanning routine, allowing the camera to sweep the area until a subject reappeared.

The live video streaming was accessible through a web browser, and the ESP32-CAM maintained a stable connection while simultaneously performing detection and servo motion. Although minor frame rate drops occurred during rapid movement, the stream remained useable for real-time tracing. Overall, the integrated setup demonstrated effective operation across all tasks, confirming that the ESP32-CAM, servos, and control algorithms functioned well as a unified surveillance system.




Output
Serial Monitor X
Message (Enter to send message to 'AI Thinker ESP32-CAM' on 'COM3')

```
Faces not not detected 1 times
Faces detected at 243028
Center detected at 139 dots
H170
V50
Faces not not detected 1 times
Faces detected at 244210
Center detected at 136 dots
H170
V85
Faces not not detected 1 times
Faces not not detected 2 times
```

6.0 Discussion

For Task 1, the pushbutton control successfully highlighted the basic interaction between user input and actuator output. This served as a foundation for understanding how servo movement responds to logic-based conditions, including angle boundaries and direction changes. The responsiveness of the servo indicated that the internal pull-up configuration and PWM generation through the ESP32Servo library were implemented well.

For Tasks 2 and 3, integrating face detection with pan-tilt control introduced a greater system. The MTMN face detection model performed reliably at QVGA resolution, providing consistent bounding box outputs for the control algorithm. The proportional error-correction approach allowed the camera to track faces smoothly. The inclusion of dummy servos ensured that PWM timer conflicts were avoided, which was important to prevent jitter.

7.0 Conclusion

This experiment achieved its objectives by implementing a functional smart surveillance prototype that combined manual user control, face detection capability, and dual-axis servo movement. The pushbutton interface allowed straightforward manual actuation, while the face detection algorithm automated the camera's pan-tilt response based on detected facial positions.

The system performed reliably across all tasks, demonstrating stable servo actuation, accurate detection, and effective environmental scanning when no face was present. These outcomes highlight the potential of the ESP32-CAM as a cost-effective platform for basic surveillance and object tracking applications, as well as its suitability for educational mechatronics projects requiring sensor-actuator integration.

8.0 Recommendation

Several improvements can be suggested. It is recommended to use a dedicated 5V power supply capable of delivering at least 2A. Servos are sensitive to voltage fluctuations, and insufficient current can cause sudden drops that may affect both servo movement and camera stability. By providing a stable power source, the system can maintain consistent operation, preventing interruptions during tracking and improving overall performance.

Improving the smoothness of the tracking responses is also advised. Sudden jumps in servo movement were observed when the detected face moved quickly or when detection signals fluctuated. Incorporating a PID controller or implementing a smoothing filter can reduce these abrupt movements, resulting in more precise tracking. Additionally, while the use of QVGA resolution allows faster processing, increasing the camera resolution to VGA or higher could enhance detection accuracy, particularly in situations where finer details are necessary.

Finally, system robustness can be further enhanced by adopting advanced tracking methods and improving mechanical stability. Implementing object reidentification or color tracking algorithms could help the system maintain focus when multiple faces are present or during rapid movements. Moreover, enclosing the setup in a rigid mount would reduce vibrations that may interfere with servo precision and camera detection. These modifications would contribute to a more reliable and accurate face tracking system.

9.0 References

1. Mechatronics System Integration Lab Module - Google Drive https://drive.google.com/drive/folders/1QecGGN96D76UL2gwBjevFwiEXv2UEniO

2. Random Nerd Tutorials. (n.d.). *How to program / upload code to ESP32-CAM AI-Thinker (Arduino IDE)*. https://randomnerdtutorials.com/program-upload-code-esp32-cam/
3. Cytron Technologies. (n.d.). *CH340 USB to TTL serial cable*. https://my.cytron.io/p-ch340-usb-to-ttl-serial-cable
4. Arduino-er. (2020, September). *Program ESP32-CAM using FTDI adapter*. https://arduino-er.blogspot.com/2020/09/program-esp32-cam-using-ftdi-adapter.html
5. TechTime. (2020, October 6). *Program ESP32-CAM using FTDI adapter* [Video]. YouTube. https://www.youtube.com/watch?v=D3MPBPGT3cw
6. Core Electronics. (n.d.). *Use an ESP32-CAM module to stream HD video over local network*. https://core-electronics.com.au/guides/esp32-cam-set-up/

10.0 Acknowledgement

I would like to express my sincere gratitude to Dr. Wahju Sediono, Dr. Zulkifli Bin Zainal Abidin, my teaching assistant, and my peers for their invaluable guidance and support throughout the completion of this report. Their insights, feedback, and expertise greatly contributed to the depth and quality of this work. I truly appreciate their time, patience, and commitment to my academic growth.

11.0 Student's Declaration

Certificate of Originality and Authenticity

This is to certify that we are responsible for the work submitted in this report, that the original work is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.

We hereby certify that this report has not been done by only one individual and all of us have contributed to the report. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have read and understand the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We therefore, agreed unanimously that this report shall be submitted for marking and this final printed report has been verified by us.

Name: Shamsul Hakimee Bin Shamsul Hairee	Read	[/]
Matric Number: 2315027	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Qashrina Aisya binti Mohd Nazarudin	Read	[/]
Matric Number: 2315184	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Nurel Humairah Binti Muhammad Firdaus	Read	[/]
Matric Number: 2315680	Understand	[/]
Signature: [illegible]	Agree	[/]