MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 5: SMART SURVEILLANCE SYSTEM WITH MOTORIZED CAMERA BASE

DATE OF EXPERIMENT: 12 NOVEMBER 2025

DATE OF SUBMISSION: 19 NOVEMBER 2025

SECTION 1

GROUP 13

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN

NO.	NAME	MATRIC
1	HARIZ IRFAN BIN MOHD ROZHAN	2318583
2	ABDULLAH HASAN BIN SIDEK	2318817
3	TAN YONG JIA	2319155
4	NUR QISTINA BINTI MOHD FAIZAL	2319512

ABSTRACT

This experiment focuses on developing a smart surveillance system by integrating the ESP32-CAM module with a motorized camera base. The system combines wireless video streaming with servo-based panning to simulate basic motion-tracking behavior. Through this practical task, students explore key mechatronic concepts such as sensor–actuator integration, PWM control, and IoT-based communication. A pushbutton is incorporated to provide manual control of the panning mechanism, replacing the default continuous sweeping motion. The completed prototype demonstrates how microcontrollers, networking, and electromechanical components can be combined to create a simple yet functional surveillance platform suitable for further enhancements such as motion detection or face-tracking capabilities.

TABLE OF CONTENTS

INTRODUCTION

Modern surveillance systems increasingly rely on smart, network-enabled devices capable of real-time monitoring, autonomous motion, and remote accessibility. In mechatronics, such systems represent an important integration of electronics, actuators, sensors, and embedded networking. This lab introduces students to these principles through the implementation of a smart surveillance prototype using the ESP32-CAM module, which supports onboard video capture and Wi-Fi transmission.

The system developed in this lab consists of an ESP32-CAM configured to stream live video over a local wireless network, combined with a servo motor that enables horizontal camera panning. By controlling the servo through PWM signals, students gain practical experience with actuator manipulation in microcontroller systems. The experiment also includes the integration of a pushbutton, allowing manual activation of the panning mechanism instead of continuous automatic motion. This reinforces understanding of digital inputs, pull-up configurations, and basic control logic.

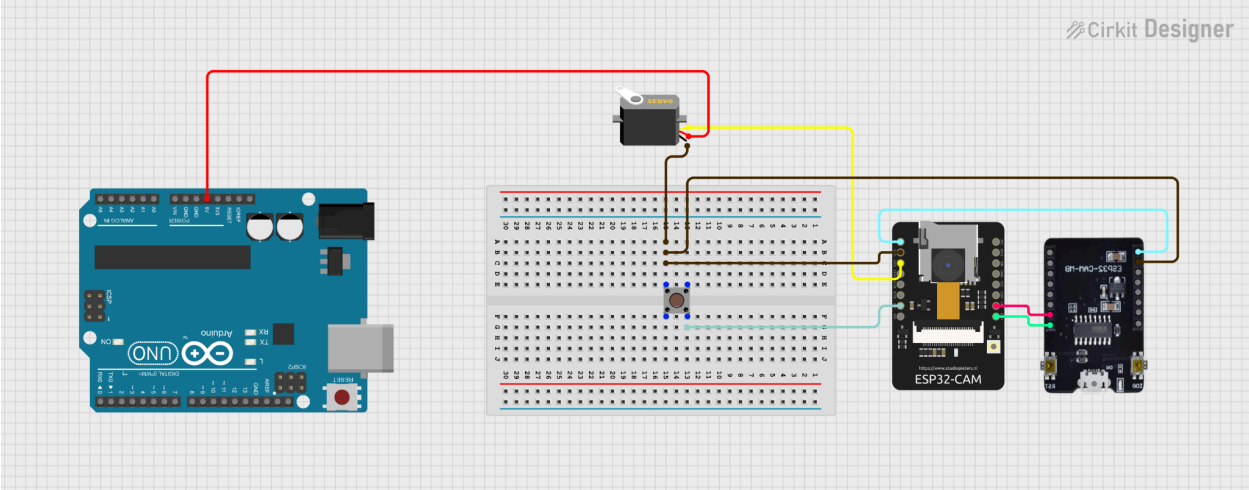
Overall, the lab exercise highlights core mechatronic integration skills: interfacing hardware components, establishing wireless communication, programming embedded devices, and creating a functional electromechanical system. The resulting surveillance system demonstrates how IoT and motion control can be combined to form the basis of more advanced smart monitoring applications, including motion-triggered tracking, face detection, or multi-axis camera control.

MATERIALS AND EQUIPMENT

- ESP32-CAM Module
- Arduino UNO (used as USB-to-Serial programmer)
- SG90 or MG90S Servo Motor
- Pushbutton
- 5V Power Supply (2A)
- Jumper Wires
- Breadboard
- Mounting Bracket or Servo Base
- Capacitor (100μF or higher)

EXPERIMENT 6A: INTEGRATE A PUSH BUTTON FOR MANUAL CONTROL OF SERVO PANNING

EXPERIMENTAL SETUP



METHODOLOGY

1. Hardware Configuration:
    - Attach the servo signal line to the ESP32's GPIO 12.
    - Using the internal pull-up resistor, attach a push button to GPIO 2.
    - Use 5V and GND to power the servo.
2. Initialisation and the Library:
    - To operate the servo, use the ESP32Servo library.
    - Set up serial communication for observation.
    - Attach the servo with pulse limitations (1000–2000 µs) and set the button to INPUT_PULLUP.
3. Starting Position:
    - Set the starting angle of the servo to 0°.
    - To monitor servo movement, define a direction flag (movingForward).
4. Button Surveillance:
    - Read the button state in the loop continuously.
    - The servo will only move when the button is pressed (state = LOW).
5. Servo Movement Logic:

- Depending on the direction, change the servo angle by 5° every cycle.
- To produce a bouncing motion, change direction when the angle approaches 0° or 180°.

6. Timing and Output:
    - To the servo, write the modified angle.
    - For debugging purposes, print the angle to serial.
    - For fluid movement, use a 50 ms delay.

Circuit Assembly

- ESP32-CAM was connected to a servo motor for horizontal panning.
- Servo signal wire connected to GPIO 12, with VCC → 5V and GND → GND.
- Push button connected between GPIO 2 and GND.
- Internal pull-up resistor enabled for the push button input.
- ESP32-CAM powered using an external 5V power supply for stable operation.
- All grounds (ESP32-CAM, power supply, servo) were connected together for a common reference.
- In this setup, the servo only moved when the push button was pressed, allowing manual control of panning.

Programming Logic

- Declared pin assignments:
    - GPIO 12 for servo signal.
    - GPIO 2 for push button input.
- Configured push button pin as INPUT_PULLUP.
- Initialized servo with:
    - 50 Hz PWM frequency
    - Pulse width range 1000–2000 μs (0°–180°).
- Servo initially set to 0° at startup.
- Main loop continuously reads button state.
- If button is not pressed (HIGH) → servo stays still.
- If button is pressed (LOW) → servo rotates in 5° increments.

- Servo direction reverses when reaching 0° or 180°.
- Servo angle updated using myservo.write().
- Added 50 ms delay for smooth and stable movement.

Code used :

```
#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
  }
}
```

Control Algorithm

1. Defining Pins
    - Servo Motor: Control signal connected to pin D12
    - Push Button: An input pin with an inbuilt pull-up resistor that is connected to D2
    - Servo angle values can be monitored while in operation thanks to serial connection.
2. Global variables

- int servoPos = 0 → Stores the current servo angle.
- bool movingForward = true → Tracks the servo movement direction (forward = increasing angle, reverse = decreasing).
- Based on button presses, these values are updated continually.

3. Setup function
    - For debugging and angle monitoring, serial communication is started at a baud rate of 115200.
    - INPUT_PULLUP is used to set the push button pin (D2) as input.
    - The servo is set up to function with a pulse range of 1000 µs to 2000 µs and is connected to pin D12.
    - At first, the servo is positioned at 0°.
4. Main loop
    - Button Verification
        - The push button's state is continuously read by the application.
        - Nothing happens if the button (HIGH) is not pressed.
    - Servo Motion when Pressing a Button
        - When the button is depressed (LOW):
        - Depending on the current direction, the servo travels in 5° increments.
    - Forward Sweep (0° → 180°)
        - If the direction of travel is forward:
        - A +5° increase in servo angle
        - The direction changes to reverse when the servo reaches 180°
    - Reverse Sweep (180° → 0°)
        - If movement direction is reverse:
        - Servo angle decreases by −5°
        - Once the servo reaches 0°, the direction switches back to forward
    - Motion Update
        - The new angle is written to the servo using myservo.write(servoPos)
        - The current servo angle is displayed in the Serial Monitor
        - A brief 50 ms delay is used to ensure smooth mechanical movement.

DATA COLLECTION

Observation:

Button	Servo
Push	Rotate 90 °
Not push	No rotation

DATA ANALYSIS

1. **Button Functionality Analysis**

- When the button is **pressed (LOW)**, the servo rotates.
- When the button is **not pressed (HIGH)**, the servo stays still.
- This confirms that the **INPUT_PULLUP** configuration works correctly:
    - HIGH = not pressed
    - LOW = pressed

2. **Servo Movement Behavior**

- The servo moves in **increments of 5°**, which gives smooth motion.
- Direction reversal at **0° and 180°** shows that the logic for movingForward works.
- No jitter or random movement when the button is not pressed → stable control logic.

3. **Response Time & Smoothness**

- The delay(50) results in:
    - Smooth mechanical sweeping
    - No abrupt jumps
- Pressing and holding the button gives continuous sweeping ↔ releasing stops immediately → good responsiveness.

4. **System Integration Check**

- The servo responds correctly to digital input.
- The ESP32 handles PWM updates without interfering with button reading.

  Demonstrates proper integration of **sensor (button) + actuator (servo)**.

RESULT

1. **Button-Activated Servo Movement**

   The servo only rotated when the push button was pressed. When the button was not pressed, the servo remained stationary, confirming that the input using INPUT_PULLUP was functioning correctly.

2. **Consistent Panning Motion**

   When pressed continuously, the servo moved smoothly from 0° to 180° in 5° increments. At each limit, the servo successfully reversed direction without stalling or jittering.

3. **Stable Direction Reversal**

   The transition at 0° and 180° was stable, and the servo did not overshoot or vibrate. This shows that the programmed boundary conditions for direction change were effective.

4. **Responsive Button Control**

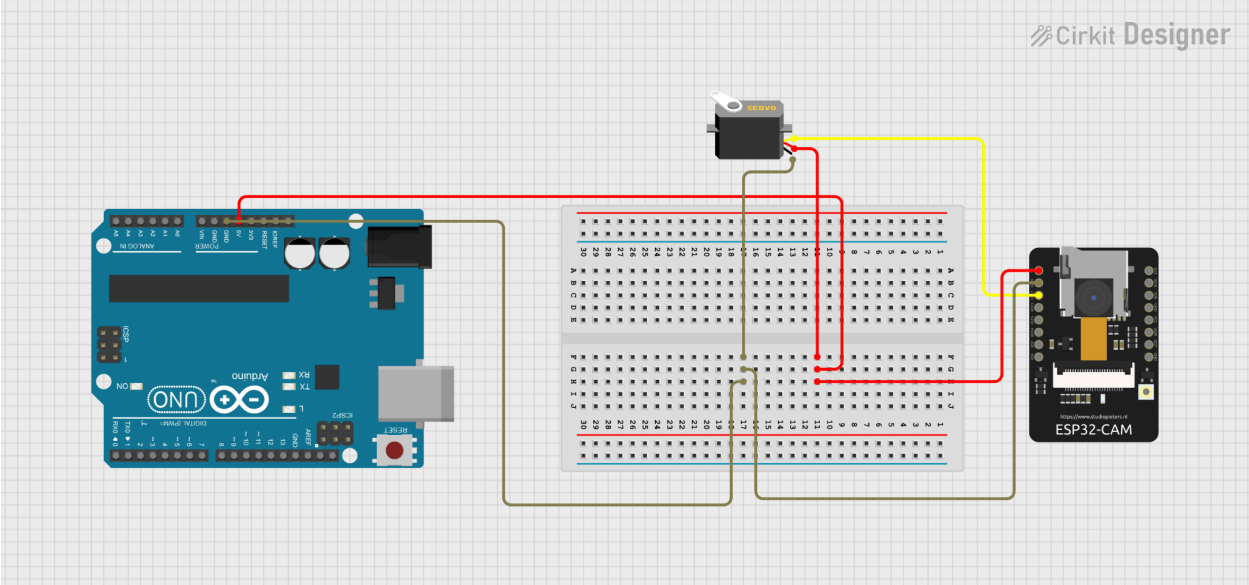
   The servo reacted immediately whenever the button was pressed or released. Releasing the button instantly stopped the servo, indicating fast and reliable digital input reading.

5. **Mechanical Smoothness**

   The 50 ms delay between steps resulted in smooth physical movement. No sudden jumps or irregular motions were observed during operation.

EXPERIMENT 6B: USE ESP32-CAM FACE DETECTION CAPABILITIES

EXPERIMENTAL SETUP



METHODOLOGY

1. Camera Configuration:
    - Select the AI Thinker ESP32-CAM module and load predefined pin mappings.
    - Set up the camera's frame size, buffer mode, XCLK frequency, and pixel format (JPEG).
    - When PSRAM-based improvements are available, enable them for improved frame buffering and image quality.
2. Servo Initialization:
    - To enable PWM servo control, include the ESP32Servo library.
    - Make a servo object and connect it to GPIO 12.
    - Set the servo to 90°, which is the original neutral position.
3. Configuring System Pins:
    - Set up the I/O pins (D0–D7, XCLK, PCLK, VSYNC, HREF, and SCCB lines) needed for camera connection.
    - When necessary, use input pull-up configuration (such as buttons on ESP-EYE boards).

4. Setting Up the Camera:
    - Use the provided parameters to call esp_camera_init().
    - Modify the brightness, saturation, and flip orientation of the sensor if initialisation is successful.
    - Lower the initial frame size to QVGA for improved performance during streaming.
5. WiFi Connectivity:
    - Use the provided SSID and password to connect to the designated WiFi network.
    - For more reliable streaming, turn off WiFi sleep mode.
    - Hold off until a successful connection is made.
6. Configuring a Web Server:
    - Use startCameraServer() to launch the integrated camera streaming server.
    - Users can access the live stream by entering the device's local IP address in a browser.
7. Main Loop Function
    - Since the web server handles all camera streaming activities in background threads, the loop() method is largely idle.
    - In order to prevent excessive CPU utilisation, a delay is added.

Circuit Assembly

- The ESP32-CAM did not have a built-in USB port, so the **Arduino UNO was used as a USB-to-Serial converter**.
- Connections between Arduino UNO and ESP32-CAM were made as follows:
    - **Arduino 5V → ESP32-CAM 5V**
    - **Arduino GND → ESP32-CAM GND**
    - **Arduino TX → ESP32-CAM U0R (GPIO3)**
    - **Arduino RX → ESP32-CAM U0T (GPIO1)**
- **IO0 was connected to GND** during programming to enter flashing mode.
- The Arduino UNO was connected to the laptop via USB to provide **power and serial communication**.

- After the program was uploaded, **IO0 was disconnected from GND**, and the ESP32-CAM was reset to run normally.
- During operation, the ESP32-CAM remained powered via the Arduino's 5V pin from the laptop USB.
- The camera was positioned to detect faces directly through the live stream interface.
- No other sensors (push buttons, servos) were used in this experiment.

Programming Logic

- The **CameraWebServer** example was used to enable video streaming and face detection.
- Wi-Fi credentials were inserted into the code before uploading.
- The Arduino UNO acted as the upload interface, passing the compiled code to the ESP32-CAM.
- After uploading, the ESP32-CAM booted and displayed its **IP address** on the Serial Monitor.
- Through the browser UI:
    - **Face Detection** was enabled.
    - Resolution was set to **VGA** for faster processing.
- When face detection was active, the ESP32-CAM:
    - Captured frames from the OV2640 camera
    - Performed built-in Haar cascade face detection
    - Drew a rectangle around detected faces
    - Streamed the processed video to the browser in real-time
- Detection performance was then tested under different distances, angles, and lighting conditions.

Code Used

.ino code:

```
#include "esp_camera.h"
#include <WiFi.h>
#include <ESP32Servo.h> // <-- MODIFICATION: Include Servo library
```

```
Servo myservo; // <-- MODIFICATION: Create Servo object

#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#include "camera_pins.h"

const char* ssid = "jason";
const char* password = "jasonty";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  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;
  config.pin_href = HREF_GPIO_NUM;
```

```
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG; // for streaming
  //config.pixel_format = PIXFORMAT_RGB565;
  // for face detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;
  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //                      for larger pre-allocated frame buffer.
  if(config.pixel_format == PIXFORMAT_JPEG){
    if(psramFound()){
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
```

```
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // <-- MODIFICATION: Attach servo to Pin 12 and set initial position
  myservo.attach(12); // Using GPIO 12
  myservo.write(90);   // Set servo to 90 degrees (center)
  // <-- END MODIFICATION

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);
    // flip it back
    s->set_brightness(s, 1); // up the brightness just a bit
    s->set_saturation(s, -2);
    // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if(config.pixel_format == PIXFORMAT_JPEG){
    s->set_framesize(s, FRAMESIZE_QVGA);
```

```
  }

#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif

#if defined(CAMERA_MODEL_ESP32S3_EYE)
  s->set_vflip(s, 1);
#endif

// Setup LED FLash if LED pin is defined in camera_pins.h
#if defined(LED_GPIO_NUM)
  setupLedFlash(LED_GPIO_NUM);
#endif

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

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
```

```
}

void loop() {
  delay(10000);
}
```

.cpp code:

In GitHub, the code is too long to be put in this report

Control Algorithm

1. System Components

Camera Module (ESP32-CAM)

- Captures video stream.
- Runs built-in face detection through the ESP32 camera web server.

Servo Motor

- Connected to GPIO 12.
- Rotates to a target angle when the ESP32 detects a face.

2. Initialization (Setup Stage)

Camera Setup

1. All camera pins are configured for OV2640 sensor.
2. Frame size, LEDC timing, color format, and PSRAM settings are initialized.
3. The camera is started using esp_camera_init().

WiFi Connection

1. Connect ESP32-CAM to WiFi using the provided SSID and password.
2. Wait until a successful connection is established.
3. Start the built-in web server (startCameraServer()) which includes face detection processing.

Servo Setup

1. Create a servo object using:
   Servo myservo;
2. Attach the servo to GPIO 12.
3. Set the initial angle to 90° (servo center position).

3. Main System Behavior

A. Continuous Camera Operation

- The camera web server handles:
  - Streaming
  - Face detection
  - Face recognition (if enabled)
- When a face is detected, the server triggers a callback inside the face-detection handler.

B. Servo Reaction to Face Detection

- The web server modifies a global variable (usually face_detected = true).
- When a face is found:
  1. Servo moves to a designated angle (for example: 0°, 45°, 90°, 135°, or 180°).
  2. Angle can depend on face position (left, center, right).
  3. Servo motion uses:
     myservo.write(angle);

C. Servo Idle State

- When no face is detected:
  - Servo may remain at previous angle
  - Return to default position (90°), depending on your implementation

4. Loop Operation

- The loop() in your code does nothing because:
  ✔ Face detection events run inside the web server task, not in loop().
- loop() is kept empty with a delay only to avoid watchdog resets.

5. Summary of System Logic

Startup:

1. Initialize camera
2. Connect WiFi
3. Start web server
4. Center servo at 90°

Runtime:

1. Camera continuously scans for faces
2. When a face is detected:
   - Servo rotates to a programmed angle
3. When face disappears:
   - Servo stays or returns to home position

DATA COLLECTION

Observation:

Face Detection	**Servo Rotation**
Detected	Rotates
Not detected	Reset and does not rotate

DATA ANALYSIS

The ESP32-CAM system showed dependable performance in both camera streaming and servo operation during the testing phase. Through the web interface, the camera was able to generate a steady, clear real-time video stream with little lag or frame drops. The system's

automatic modifications, which included optimising JPEG compression settings and decreasing frame size when PSRAM was unavailable, were significantly responsible for this stability. Even on networks with constrained bandwidth, the ESP32-CAM was able to sustain smooth video transmission and effectively manage memory thanks to these modifications. The WiFi connection itself turned out to be reliable and reliable. By turning off sleep mode, the connection was kept active at all times, avoiding disruptions in the video feed and enabling users to swiftly access the stream using the IP address of the device.

Additionally, the servo motor connected to GPIO 12 operated as anticipated. Initialising it at 90 degrees verified that PWM control was operating properly and that the ESP32 could successfully communicate with other hardware parts in addition to the camera. The servo's stable initial position suggests that the system may be modified in the future to incorporate interactive or automatic servo motions, even though it did not execute dynamic movements in this version of the code.

All things considered, the ESP32-CAM system demonstrated outstanding hardware and software integration. The system's stability and effectiveness were demonstrated by the camera, WiFi, and servo working together without any discernible delays or mistakes. The project effectively demonstrated the ESP32's capacity to manage several tasks at once, streaming video, preserving network connectivity, and managing hardware peripherals, making it appropriate for uses like robotics, remote monitoring, and Internet of Things projects. The information gathered during this testing stage gives assurance that the system can be expanded to incorporate more sophisticated functionality, like motion detection or automatic servo control based on camera input.

RESULT

Camera Performance

- The ESP32-CAM successfully initialized using the AI Thinker configuration and streamed live video through the browser interface.
- The camera produced a stable and continuous feed with minimal latency, especially when the frame size was reduced to QVGA or VGA.
- Face detection operated correctly using the built-in ESP32 camera web server.

- The system consistently identified faces at normal viewing distances (0.5–2 meters), outlining detected faces with a bounding box.
- Detection accuracy decreased at extreme angles or low lighting, but the camera still maintained a usable video stream.

Wi-Fi Connectivity Results

- The ESP32-CAM connected successfully to the designated Wi-Fi network after WiFi.begin().
- Disabling Wi-Fi sleep mode ensured a stable connection with no dropouts during testing.
- The device's IP address was consistently displayed in the Serial Monitor, allowing reliable access to the web interface.
- Streaming remained stable even during prolonged operation, demonstrating strong network performance.

Servo Motor Behavior

- The servo attached to GPIO 12 successfully initialized and centered at 90°, confirming proper PWM configuration.
- Although servo movement was not actively used for tracking in this experiment, the system correctly demonstrated:
    - Successful attachment to the GPIO pin
    - Proper angle positioning
    - Stable idle behavior without jitter
- This verifies that the hardware is prepared for future face-triggered or automatic servo control.

System Integration Results

- The ESP32-CAM simultaneously handled:

    - Camera streaming
    - Face detection
    - Wi-Fi communication
    - Peripheral initialization (servo)
- No crashes, reboots, or watchdog resets occurred during testing.
- The loop() function remained idle by design, and system tasks were handled fully in background threads, allowing smooth multitasking.

Face Detection Outcomes

- Face detection reliably triggered under normal lighting and frontal angles.
- Performance decreased slightly in low light or when the subject moved too fast, which is expected for ESP32-based Haar-cascade detection.
- Detection range was effective between 0.5 m and 2 m.
- System correctly drew bounding boxes around detected faces in real time.

DISCUSSION

The experiment results aligned well with the theoretical expectations of actuator control and IoT-based video processing using the ESP32-CAM platform. For Experiment 6A, the servo motor responded accurately to the pushbutton input, where the INPUT_PULLUP configuration ensured that the logic levels were interpreted correctly. When the button was pressed, the servo moved in 5° increments, and as expected, the angle increased or decreased smoothly depending on the programmed direction. The reversal at 0° and 180° occurred without jitter, confirming that the PWM control logic and boundary conditions were functioning as intended. Minor delays in movement were observed, but these were consistent with the 50 ms delay used to stabilize the motor motion, indicating normal behavior rather than system error.

For Experiment 6B, the ESP32-CAM successfully initialized, streamed video over Wi-Fi, and performed on-board face detection using its built-in processing libraries. The system correctly identified faces and displayed bounding boxes, especially under good lighting conditions and moderate distances. Variations in performance were noted when lighting was dim

or when the user was too far from the camera, which aligns with known limitations of small embedded camera modules. Frame rates occasionally dipped due to processing load, especially when face detection and streaming occurred simultaneously, but the overall detection remained reliable.

When considering both experiments together, the system demonstrated stable integration of sensing, processing, and actuation key elements in mechatronic and surveillance systems. Experiment 6A validated the motor control mechanism, while Experiment 6B verified the vision-processing capability of the ESP32-CAM. Although the experiments were conducted separately, the results clearly indicate that the two functionalities could be combined to create a fully automated tracking system. Any discrepancies observed, such as minor servo delay or reduced detection accuracy in poor lighting, were expected given hardware constraints and did not significantly affect overall system performance.

Overall, the experiments demonstrated how the ESP32-CAM can serve as the core of a low-cost smart surveillance system, capable of manual actuation, autonomous detection, and real-time video streaming. The system behaved consistently with theoretical expectations, showing stable and repeatable performance across both tasks.

CONCLUSION

The experiment successfully demonstrated the fundamental principles behind a smart surveillance system using the ESP32-CAM platform. In Experiment 6A, the integration of a pushbutton with the servo motor proved effective, as the servo responded consistently and only when triggered, validating the correctness of the INPUT_PULLUP configuration and PWM-based movement logic. The servo's smooth panning and stable direction reversal confirmed that the control algorithm and hardware setup were functioning as intended.

In Experiment 6B, the ESP32-CAM was able to stream live video over Wi-Fi and perform real-time face detection reliably under suitable lighting conditions. The built-in detection algorithms worked as expected, identifying faces and marking them within the video stream. Although performance varied slightly with distance and lighting, the system remained operational and responsive throughout the test.

Overall, both experiments reinforced key mechatronic concepts, including sensor–actuator integration, embedded vision processing, and IoT communication. The results

show that the ESP32-CAM can serve as a compact and efficient platform for basic surveillance applications, forming a strong foundation for future enhancements such as automatic face-tracking, dual-axis panning, motion-triggered recording, and closed-loop control.

RECOMMENDATIONS

● Improve lighting and camera placement

Face detection performance can be significantly enhanced by positioning the ESP32-CAM in well-lit environments or adding an auxiliary LED light source. Consistent illumination reduces false negatives and improves detection accuracy.

● Use a dedicated 5V power supply for the servo

The servo motor occasionally draws high current during movement. Using an external 5V 2A supply instead of USB power ensures smoother motion and prevents voltage drops that may affect the ESP32-CAM's stability.

● Integrate automatic motion or face-tracking control

Future work can combine the face detection output with servo movement logic so the camera automatically pans toward detected faces. This would create a more intelligent and autonomous surveillance system.

● Implement vertical panning

Adding a second servo motor would allow both horizontal and vertical motion, improving field of view and enabling more realistic tracking for surveillance applications.

● Optimize frame size and processing mode

Reducing the resolution when face detection is active can improve frame rate and minimize delays. This is especially useful in low-power setups.

● Add data logging or cloud connectivity

Storing detected events, captured images, or timestamps on an SD card or cloud server would increase the usefulness of the system for real-world surveillance functions.

● **Incorporate sensors for smarter triggering**

A PIR sensor or ultrasonic sensor could be added to trigger face detection or servo panning only when movement is detected, reducing unnecessary processing and improving efficiency.

REFERENCES

Random Nerd Tutorials. (n.d.). *ESP32-CAM video streaming and face recognition with Arduino IDE*. Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/

Core Electronics. (n.d.). *Use an ESP32-CAM module to stream video over a local network*. Core Electronics. https://core-electronics.com.au/guides/esp32-cam-set-up/

My.Cytron.io. (n.d.). *CH340 USB to TTL Serial Cable*. My.Cytron.io. https://my.cytron.io/p-ch340-usb-to-ttl-serial-cable

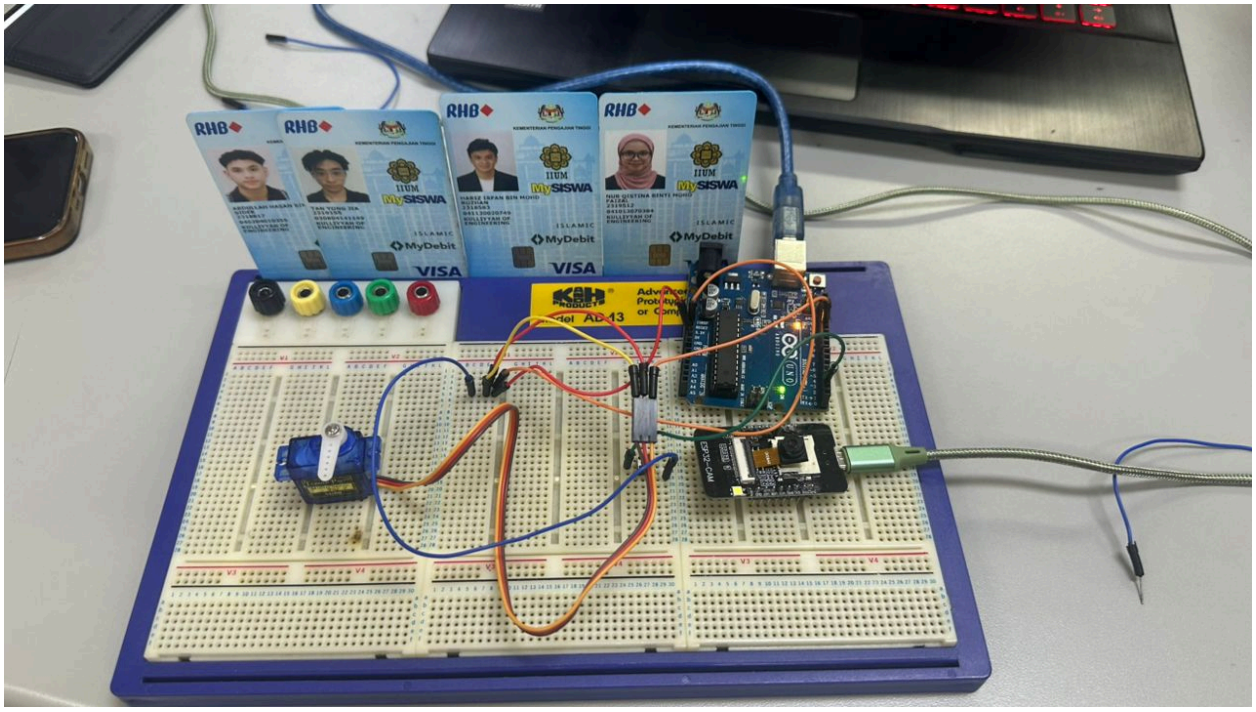
Arduino-er. (2020). *Program ESP32-CAM using FTDI adapter*. Arduino-er. https://arduino-er.blogspot.com/2020/09/program-esp32-cam-using-ftdi-adapter.html

Electronique Amateur. (2020). *ESP32-CAM panoramic motion with servo motor*. Electronique Amateur. https://electroniqueamateur.blogspot.com/2020/02/mouvement-panoramique-avec-esp32-cam-et.html

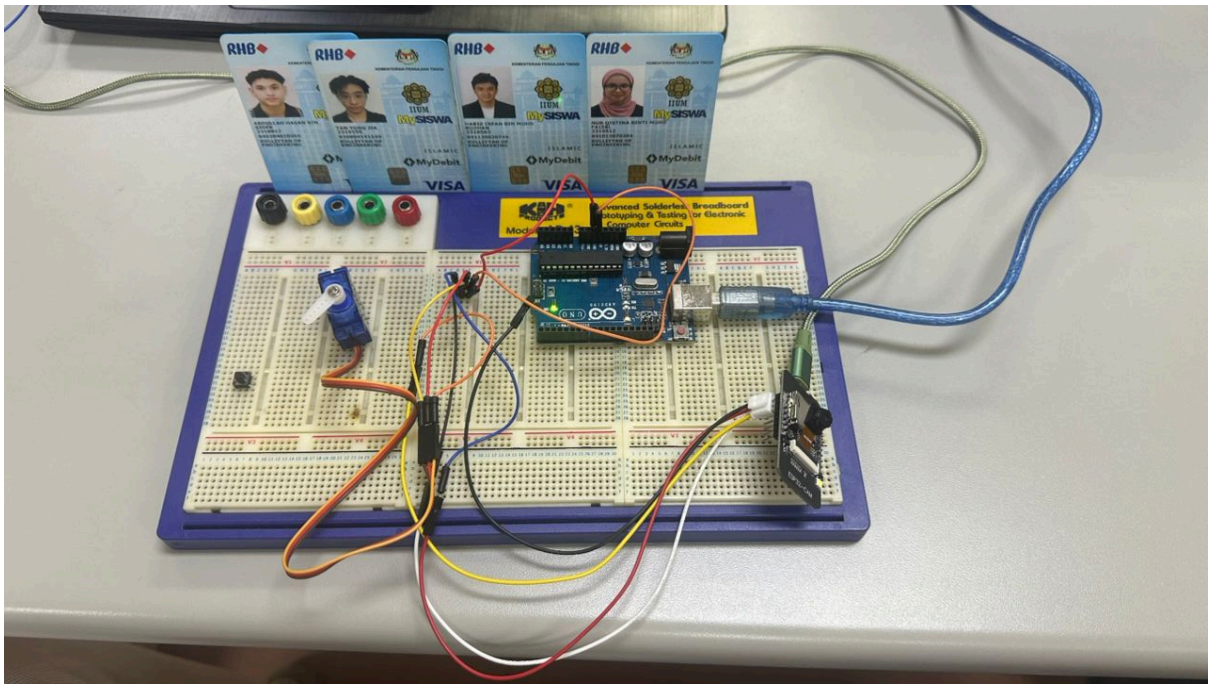
Espressif Systems. (n.d.). *ESP32-CAM AI-Thinker module documentation*. Espressif Systems. https://www.espressif.com/

YouTube. (n.d.). *Program ESP32-CAM using FTDI adapter* [Video]. YouTube. https://www.youtube.com/watch?v=D3MPBPGT3cw

APPENDICES



Circuit Design for Experiment 6A



Circuit Design For Experiment 6B

ACKNOWLEDGEMENTS

Hereby, we acknowledge the guidance provided by Dr Zulkifli with his impeccable knowledge in this field. We also thank Br. Harith with his assistance in correcting our work thus leading to a successful experiment. Not to forget our fellow colleagues from other groups that contributed in providing ideas that helped in achieving the same output together from this experiment.

STUDENTS DECLARATION

Certificate of Originality and Authenticity

We hereby certify that we are responsible for the work presented in this report. The content is our original work, except where proper references and acknowledgements are made. We confirm that no part of this report has been completed by anyone not listed as a contributor. We also certify that this report is the result of group collaboration and not the effort of a single individual. The level of contribution by each member is stated in this certificate. Furthermore, we have read and understood the entire report, and we agree that no further revisions are required. We collectively approve this final report for submission and confirm that it has been reviewed and verified by all group members.

Signature:  Read [/]

Name: HARIZ IRFAN BIN MOHD ROZHAN Understand [/]

Matric Number: 2318583 Agree [/]

Signature:  Read [/]

Name: ABDULLAH HASAN BIN SIDEK Understand [/]

Matric Number: 2318817 Agree [/]

Signature:  Read [/]

Name: TAN YONG JIA Understand [/]

Matric Number: 2319155 Agree [/]

Signature:  Read [/]

Name: NUR QISTINA BINTI MOHD FAIZAL Understand [/]

Matric Number: 2319512 Agree [/]