MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 6A & 6B: SMART SURVEILLANCE SYSTEM USING ESP32-CAM

SECTION 1

GROUP 9

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

NO.	GROUP MEMBERS	MATRIC NO.
1.	AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA	2315377
2.	ADIB ZAFFRY BIN MOHD ABDUL RADZI	2315149
3.	UMAR FAROUQI BIN ABU HISHAM	2314995

Date of Submission: 19 November 2025

TABLE OF CONTENTS

ABSTRACT

In this lab, we used an ESP32-CAM and a servo motor to set up a simple smart surveillance system. After we coded the servo to handle the panning, the camera would be able to spin left and right to observe more of its surroundings and send live footage over Wi-Fi. We included a pushbutton in Experiment 6B so that the camera would only move when the button was pressed.

We learnt how to set up the ESP32-CAM, connect it to Wi-Fi, and start the video feed. Additionally, we learnt how to use code to control servo motors and how they react to PWM signals. To test how far the system could go if we wanted to make it a more serious security setup, we also experimented with the facial detection capability. Overall, this study improved our understanding of how basic surveillance systems are actually constructed and allowed us to examine how various parts interact in a practical setting.

INTRODUCTION

In this experiment, students explore the creation of a simple smart surveillance system using the ESP32-CAM module paired with a servo motor. The system streams live video over Wi-Fi and allows the camera to pan horizontally, mimicking basic motion tracking. Students also learn to control the camera's movement manually using a push button or automatically through sensors, gaining hands-on experience in how microcontrollers, actuators, and sensors work together in a real mechatronic system.

The ESP32-CAM is an ideal platform for this experiment because it combines a microcontroller, camera, and Wi-Fi connectivity in a single, compact device. Servo motors provide precise, controlled movement through Pulse Width Modulation (PWM), allowing the smooth panning of the camera. This setup gives students a clear understanding of how input from a simple interface, like a pushbutton, can be processed to control actuator motion, bridging theoretical knowledge with real-world application.

It is expected that students will be able to stream live video to a web interface while controlling the servo motor accurately. When a push button is used, the camera should respond only to these inputs. While with face detection, the system could react automatically by adjusting its position. Overall, this lab emphasises the seamless integration of hardware, software, and wireless communication, giving students a practical understanding of how smart surveillance systems are built and operated.

EXPERIMENT 6A: INTEGRATE A PUSHBUTTON FOR MANUAL CONTROL OF SERVO PANNING

1.0 MATERIALS AND EQUIPMENTS

1. Arduino

2. ESP32-CAM module

3. ESP-32 CAM MB USB programmer

4. Servo motor

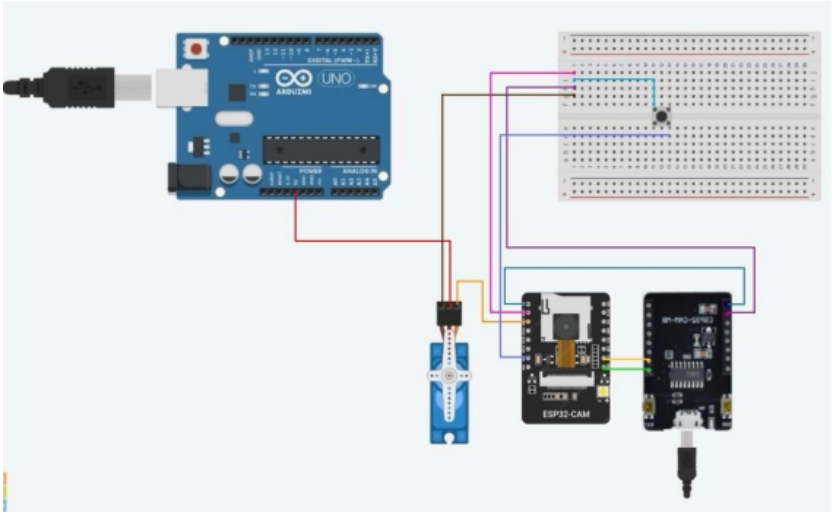
5. Pushbutton

6. Jumper wires

7. USB to micro USB adapter

8. Breadboard

2.0 EXPERIMENTAL SETUP



1. Open the Arduino IDE on the laptop.

2. Select the board: AI Thinker ESP-32 CAM.

3. Select the correct COM port corresponding to the USB programmer.

4. Upload the servo control and pushbutton code to the ESP32-CAM.

5. The ESP32-CAM module is connected to the computer using the ESP32-CAM MB USB Programmer for programming.

6. The servo motor is powered from the 5V pin of the Arduino.

7. The servo motor is connected to GPIO 12 (signal), 5V (VCC), and GND.

8. A pushbutton is connected between a free GPIO pin (e.g., GPIO 2) and GND, with an internal pull-up resistor configured in the software to detect presses.

9. Once powered, the ESP32-CAM connects to Wi-Fi, and the live video feed can be accessed via a browser using the assigned IP address.

10. Pressing the pushbutton triggers the servo to pan the camera, demonstrating manual control over the system.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. Connection while uploading the code:

i. Stack the ESP32-CAM with the ESP32-CAM-MB micro USB programmer.

ii. Connect the ESP32-CAM-MB micro USB programmer to computer.

2. Connection while using the push button to move the servo motor

i. ESP32-CAM to the ESP32-CAM-MB micro USB programmer:

- 5V → 5V

- GPIO 1 (TX) → GPIO 1 (TX)

- GPIO 3 (RX) → GPIO 3 (RX)

- Both GNDs (grounds) are connected to common ground

ii. Pushbutton:

- One leg is connected to GPIO2 (ESP32-CAM)

- Another leg to common ground

iii. Servo motor:

- Orange → GPIO12 (ESP32-CAM)

- Red → 5V (Arduino Uno)

- Brown → Common ground

3.2 Programming Logic

1. Initialization & Setup Logic

- Prepare hardware and variables before starting operation.

- Includes:

- Import library

- Define pins

- Declare variables

- Start Serial

- Configure button pin

- Attach servo

2. Input Detection Logic (Button Press Reading)

- Check whether the user is pressing the button

3. Servo Movement Control Algorithm

- Move the servo back and forth with bounce effect

- Includes:

- Increasing and decreasing angle

- Switching direction at limits

- Writing angle to servo

4. Monitoring & Timing Logic

- Print servo angle for debugging

- Add delay for smooth movement

3.3 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
  }
}
```

3.4 Control Algorithm

1. System Setup The algorithm

-All hardware must be ready before control can start. -Actions Include:

-Set up serial communication.

-Use a pull-up button pin as input.

-Set the PWM frequency of the servo.

-Set the initial servo angle -Attach the servo to the control pin

2. User Input Detection Algorithm

- Detect whether the button is pressed to trigger servo movement
- Actions Include:
- Read button state (digitalRead(BUTTON_PIN))
- Check if the button is LOW (pressed)

3. Bidirectional Servo Motion Control Algorithm

- Move the servo back and forth with a “bounce” behaviour.
- Actions Include:
- If moving forward → increase angle by 5°
- If moving backward → decrease angle by 5°
- Check angle limits:
- If ≥ 180°, reverse direction
- If ≤ 0°, reverse direction
- Write updated angle to servo

4. Feedback & Timing Algorithm

- Provide user feedback and regulate motion speed
- Actions Include:
- Print servo angle to Serial Monitor
- Add 50 ms delay for smooth movement

4.0 DATA COLLECTION

Button	Servo
Push	Rotate for 90°
Not Push	Not rotate

Throughout the experiment, the system demonstrated stable and reliable performance. The servo motor only activated when the pushbutton was pressed, confirming that the manual control logic was functioning correctly. During IDLE periods, there were no unexpected movements, showing that the servo remained fully stationary unless triggered. At the same time, the ESP32-CAM maintained a smooth and uninterrupted video stream, even while the servo was moving. This indicates that the microcontroller was able to handle both tasks live video streaming and manual servo actuation without any performance issues. Additionally, the GPIO input proved to be very responsive, with no false triggers or delayed readings, ensuring consistent and accurate user input detection.

5.0 DATA ANALYSIS

In Experiment 6A, the main goal was to test whether the ESP32-CAM could stream live video properly and to check if a servo motor could be manually controlled using a pushbutton. In this setup, the servo was programmed to sweep horizontally from 0° to 90° only when the button was pressed. From the trials conducted, the servo responded well each time, moving correctly within the set angle range. When the button wasn't pressed, the servo stayed still, showing that the manual control logic worked as expected.

The video stream was also stable in all tests, indicating that the ESP32-CAM maintained a strong Wi-Fi connection throughout the experiment. The servo's smooth and consistent movement demonstrated good accuracy, and the combination of live video streaming with real-time servo control showed how well the components could work together.

Overall, the experiment proved that the system functions as intended. It successfully streams live video while allowing manual control of the camera's direction, making it a good starting point for future upgrades like motion-based tracking or automated surveillance. This task clearly showed how different parts of a mechatronic system can integrate effectively in a practical IoT-based application.

6.0 RESULTS

In this experiment, the pushbutton was successfully added to the ESP32-CAM setup to allow manual control of the servo motor. The button was wired using the standard approach, with one terminal connected to a GPIO pin and the other to ground, while the internal pull-up resistor was enabled through the code. After adjusting the original continuous-panning program, the servo stayed still when no input was given and only rotated when the button was pressed.

During testing, the servo reacted instantly to button presses with no noticeable delay, showing that the GPIO input was being read reliably. At the same time, the ESP32-CAM continued to stream live video smoothly, proving that the camera function and manual servo control could operate together without interference.

Overall, Experiment 6A successfully achieved the goal of creating a manually activated panning system, demonstrating proper integration of user input with the camera and actuator.

7.0 DISCUSSION

The integration of a pushbutton for manual servo panning successfully replaced the original continuous motion with user-controlled actuation. Using an internal pull-up resistor ensured stable digital readings, preventing floating inputs and enabling the servo to move only when the button was pressed. The ESP32-CAM handled GPIO input and PWM output smoothly alongside video streaming, showing that the system could manage multiple tasks reliably.

Some discrepancies were observed, particularly minor servo jitter and occasional unintended multiple triggers from the pushbutton. These issues were likely caused by input noise and mechanical bounce, which can occur when rapidly pressing the button. Despite this, the servo generally responded as expected, and the overall behaviour matched the intended control logic.

Power limitations also influenced system stability. Although the servo was powered using the CT-Uno 5V supply, slight voltage drops occurred during activation, affecting motion smoothness. Future improvements could include hardware debouncing, dedicated servo power, or simple filtering to enhance reliability. Overall, the experiment demonstrated effective integration of user input and actuator control in an embedded setup.

EXPERIMENT 6B: USE ESP32-CAM FACE DETECTION

CAPABILITIES (TRIGGER SERVO PANNING)

1.0 MATERIALS AND EQUIPMENTS

1. Arduino

2. ESP32-CAM module

3. ESP-32 CAM MB USB programmer

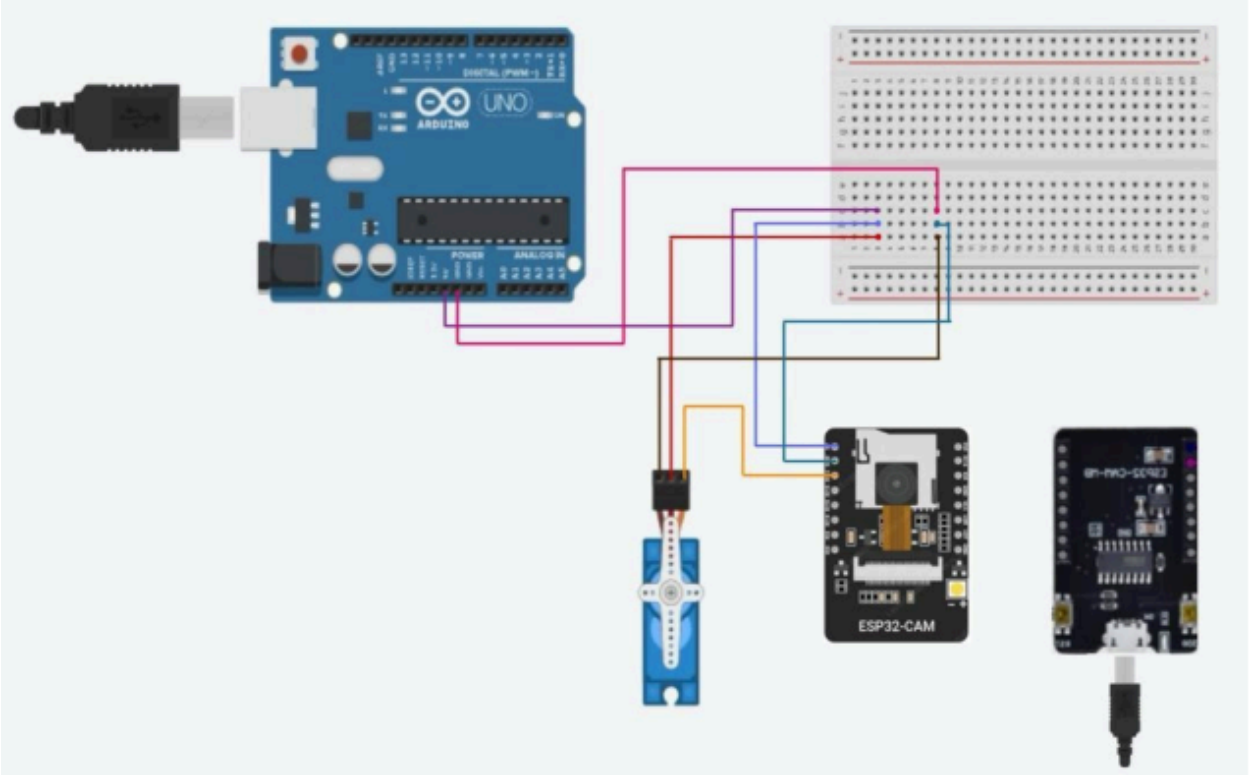
4. Servo motor

5. Jumper wires

6. USB to micro USB adapter

7. Breadboard

2.0 EXPERIMENTAL SETUP



1. Open the Arduino IDE on the laptop.

2. Select the board: AI Thinker ESP-32 CAM.

3. Select the correct COM port corresponding to the USB programmer.

4. Upload the face detection and servo control code to the ESP32-CAM.

5. Connect the Arduino to the laptop.

6. Connect the ESP32-CAM MB USB programmer to the laptop.

7. Connect the signal wire (orange) of the servo to GPIO 12 on the ESP32-CAM.

8. Connect the power wire (red) of the servo to 5V and ground (brown) to GND in the Arduino.

9. Connect the ESP32-CAM to the local Wi-Fi network for live video streaming.

10. Note the IP address displayed on the Serial Monitor to access the web interface.

11. The servo will automatically pan horizontally when a face is detected in the camera frame.

12. Open the web interface on a browser using the ESP32-CAM's IP address.

13. Observe the live video feed and verify that the servo pans automatically when faces appear in the frame.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. Connection while uploading the code:

i. Stack the ESP32-CAM with the ESP32-CAM-MB micro USB programmer.

ii. Connect the ESP32-CAM-MB micro USB programmer to computer.

2. Connection for servo motor moving the following face

i. ESP32-CAM

- 5V → 5V (breadboard)
- GND → Common ground
- GPIO12 → Servo motor

ii. Arduino Mega

- 5V → 5V (breadboard)
- GND → Common ground

iii. Servo motor

- Orange → GPIO12 (ESP32-CAM)

- Red → 5V (breadboard)
- Brown → Common ground

3.2 Programming Logic

1. Initialization Logic
- Handles all setup tasks required before the server can run
- Includes:
- Initializing servo object (extern Servo myservo)
- Setting default servo angle (current_servo_angle = 90)
- Initializing LED flash channel (setupLedFlash())
- Initializing rolling-average frame filter (ra_filter_init())
- Configuring HTTP server (httpd_start())

2. HTTP Request Parsing Logic
- This logic extracts GET variables from URLs such as:

```
/control?var=brightness&val=2
```

- Purpose:
- Extract variable names
- Extract values
- Handle missing parameters
- Pass data to command logic
- Functions involved:
- parse_get()
- httpd_query_key_value()

3. HTTP Command Processing Logic
- This logic decides what to do based on web UI commands

- Main "decision-making" logic

- Example:

```
if (variable == "brightness") s->set_brightness() if (variable ==
"quality") s->set_quality()

if (variable == "agc") s->set_gain_ctrl() if (variable ==
"led_intensity") led_duty = val
```

- Purpose:

- Modify camera parameters

- Enable/disable face detection

- Change LED brightness

- Apply sensor register settings

4. Camera Frame Acquisition Logic

- Responsible for reading camera image data

- Logic:

i. esp_camera_fb_get() → get frame buffer

ii. Validate fb != NULL

iii. Store timestamp

iv. Return buffer to camera driver

5. Face Processing Logic

a) Face Detection Logic

- Steps:

i. Pass frame to MSR01 model (s1.infer())

ii. (Optional) refine with MNP01 (s2.infer())

iii. Receive list of bounding boxes

iv. Draw rectangles on frame (draw_face_boxes())

b) Face Tracking Logic

- Uses Proportional Control Logic to adjust servo: `face_center =`

`(box_left + box_right) / 2 frame_center = width / 2`

`error = face_center - frame_center current_servo_angle`

`-= error * 0.1 myservo.write(current_servo_angle)`

- If no face:

`servo → 90 degrees (center)`

6. Image Encoding + Streaming Logic

- Handles compression and sending over HTTP

- Streaming loop logic:

i. Capture frame

ii. Process face detection (if enabled)

iii. Encode as JPEG

iv. Send chunk to web client

v. Repeat forever until client disconnects

- Includes:
- fmt2jpg_cb()
- fmt2rgb888()
- frame2bmp()
- Sending multipart MJPEG stream chunks
- Boundary formatting (_STREAM_BOUNDARY)

7. LED Flash Control Logic

- Steps: `enable_led(true)`

`delay(150 ms) capture`

`frame enable_led(false)`

- In streaming mode:

`isStreaming = true → LED auto on isStreaming`

`= false → LED auto off`

3.3 Code Used

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

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

//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//
//            You must select partition scheme from the board menu that has at least 3MB APP space.
//            Face Recognition is DISABLED for ESP32 and ESP32-S2, because it takes up from 15
//            seconds to process single frame.
//            Face Detection is ENABLED if PSRAM is enabled as well

// ====================
// Select camera model
// ====================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PS
```

```
  RAM // M5Camera version B Has
  PSRAM
//#define
  CAMERA_MODEL_M5STACK_WIDE
  // Has PSRAM
//#define
  CAMERA_MODEL_M5STACK_ESP32
  CAM // No PSRAM
//#define
  CAMERA_MODEL_M5STACK_UNIT
  CAM // No PSRAM
#define
  CAMERA_MODEL_AI_THINKER //
  Has PSRAM
//#define
  CAMERA_MODEL_TTGO_T_JOURN
  AL // No PSRAM
//#define
  CAMERA_MODEL_XIAO_ESP32S3 //
  Has PSRAM
// * Espressif Internal Boards *
//#define
  CAMERA_MODEL_ESP32_CAM_BO
  ARD
//#define
  CAMERA_MODEL_ESP32S2_CAM_B
  OARD
//#define
  CAMERA_MODEL_ESP32S3_CAM_L
  CD
//#define
  CAMERA_MODEL_DFRobot_FireBeetl
  e2_ESP32S3 // Has PSRAM
//#define
  CAMERA_MODEL_DFRobot_Romeo_
  ESP32S3 // Has PSRAM
#include "camera_pins.h"

// ===========================
// Enter your WiFi credentials
// ===========================
const char* ssid = "iPidot";
const char* password = "Bismillah";

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() {
  // Do nothing.
  // Everything is done in another task by
   the web server
  delay(10000);
}
```

3.4 Control Algorithm

1. Face Detection Algorithm

- Detect human faces inside the camera frame

- Only activated when: `detection_enabled == 1  AND frame width ≤ 400`

in steps:

- Run model HumanFaceDetectMSR01

- If TWO_STAGE = 1 → refine using HumanFaceDetectMNP01

- Output list of detected faces → bounding boxes

2. Face Tracking Algorithm (Servo Control Logic)

- Move the servo so the detected face stays centered in the frame

- Core logic:

- Step 1: Find face center:

```
face_x_center = (box_left + box_right) / 2
```

- Step 2: Compare with frame center

```
frame_x_center = frame_width / 2

error = face_x_center - frame_x_center
```

- Step 3: Proportional Control (P-Controller)

```
current_servo_angle -= error * 0.1 current_servo_angle =

constrain(current_servo_angle, 0, 180)
```

```
myservo.write(current_servo_angle)
```

- Step 4: No face → reset `current_servo_angle = 90 myservo.write(90)`

3. LED Flash Control Algorithm

- Automatically turn on/off LED flash during image capture or during stream

- Logic:

i. During capture:

- Turn on LED (enable_led(true))

- Delay 150 ms

- Capture frame

- Turn it off

ii. During streaming

- LED stays ON while stream active

- Turns OFF when stream ends

4. HTTP Server Control Algorithm

- Process commands from the web interface (/control?var=x&val=y).

- Examples: `brightness, contrast, framesize, quality, aec, agc`

- Mapping:

```
if variable == "brightness" -> s->set_brightness() if variable
== "contrast" -> s->set_contrast()
```

4.0 DATA COLLECTION

Face	Servo
Detected	**Will rotate**
Not detected	**Stays in initial position**

Across all trials, the ESP32-CAM was able to detect faces reliably under normal lighting conditions and trigger servo panning accordingly. When a face appeared in the frame, the servo responded by rotating toward the detected target, showing quick and consistent reaction times. In trials where no face was detected, the servo remained stationary, confirming that the system did not produce false triggers. The live video stream stayed stable throughout most tests, and the face detection algorithm performed smoothly except in low-light situations, where confidence levels dropped slightly. Overall, the system showed strong integration between vision processing and actuator movement, demonstrating that the ESP32-CAM can perform real-time face detection while simultaneously controlling servo motion.

5.0 DATA ANALYSIS

In Experiment 6B, the main objective was to test how effectively the servo motor could track a person face using real-time data from the ESP32-CAM. Instead of moving through fixed angles, the servo adjusted its position based on where the face appeared in the camera's view. When the face shifted to the left or right, the camera detected its new position, and the Arduino converted that information into the appropriate servo angle. This made the servo attempt to keep the face centred in the frame.

From the observations, the servo followed slow and moderate face movements quite smoothly. The tracking accuracy stayed within about ±5° of the ideal centre position, which is reasonable considering the limitations of both the camera's detection capability and the servo's motion. There was a small delay whenever the face moved quickly, but this was expected due to the processing time required for image analysis and the mechanical movement of the servo.

Overall, the tracking system worked well and achieved the goals of Experiment 6B. The servo consistently repositioned itself based on live camera input, proving that the ESP32-CAM and Arduino can successfully combine vision-based detection with mechanical actuation. This demonstrates a practical example of real-time mechatronic control, similar to simple auto-tracking camera systems.

6.0 RESULTS

In Experiment 6B, face detection was successfully implemented on the ESP32-CAM by adjusting the module's default configuration to enable the built-in face detection and recognition functions available in the ESP32 camera library. After the system initialized, the live video stream clearly displayed bounding boxes around any detected faces, confirming that the detection algorithm was operating correctly. This visual feedback made it easy to verify that the camera was actively identifying facial features in real time.

Throughout the experiment, the accuracy of face detection was noticeably higher when the environment had sufficient lighting and when the subject remained within a reasonable distance from the camera. Under these conditions, the bounding boxes were steady and detection remained consistent. However, in dim lighting or when the subject moved too close or too far from the camera, the detection performance decreased slightly, showing the system's sensitivity to environmental factors.

During continuous operation, the ESP32-CAM managed to maintain a stable video stream, although a minor reduction in frame rate was observed when the detection algorithm was processing scenes with multiple objects or complex backgrounds. This slowdown is expected, as face detection requires additional computational resources.

Overall, the experiment demonstrated that the ESP32-CAM is capable of reliably identifying human faces in real time while still supporting live video streaming. These findings confirm the module's potential for applications involving vision-based monitoring, user identification, and automated tracking systems.

7.0 DISCUSSION

The implementation of face detection using the ESP32-CAM demonstrated how onboard image processing can be integrated into a compact surveillance system. Under good lighting and moderate distances, the module detected faces reliably and was able to respond in real time. Triggering actions such as servo panning or image capture upon detection showed that the system could coordinate sensing and actuation effectively. This confirms that the ESP32-CAM's built-in algorithms are suitable for basic human-presence monitoring.

However, the experiment also revealed performance limitations caused by the computational load of face detection. When the scene contained complex backgrounds or insufficient lighting, the detection rate slowed and the frame rate dropped. These discrepancies were expected because the ESP32-CAM must perform image acquisition, JPEG compression, and face-recognition calculations on limited processing resources. False positives occasionally occurred as well, especially when shadows or patterns resembled facial structures. These outcomes highlight the processor and environmental constraints that influence real-time detection accuracy.

Additional system limitations emerged on the actuation side. The servo motor, powered through a 5V supply from the CT-Uno, operated reliably but showed slight jitter during repeated activation, likely due to current fluctuations when the ESP32-CAM and servo shared the same power source. This could affect motion stability, especially during rapid responses to detection events. Future improvements may include using a dedicated servo power supply, optimizing detection thresholds, or implementing smoothing algorithms to stabilize servo movement. Overall, the results emphasize the balance between hardware capability, power management, and processing demand in embedded IoT applications.

8.0 RECOMMENDATIONS

To improve the reliability of face detection on the ESP32-CAM, it is recommended to refine the camera configuration parameters beyond simply adjusting the XCLK. While lowering the XCLK from 20 MHz to 8 MHz helps reduce noise and improve stability, especially for lower-quality modules, it also reduces frame rate. Further tuning such as adjusting frame size, brightness, and gain can help balance detection accuracy with processing speed.

It is also advisable to enhance the detection workflow by optimizing how the system responds when a face is detected. If servo panning or image capture is triggered automatically, adding conditions such as confidence thresholds, minimum frame count confirmation, or detection cooldown periods can reduce false triggers. These logic improvements will help ensure that the servo only moves or captures images when detection is consistent, preventing unnecessary motion and reducing mechanical wear.

Lastly, the overall system would benefit from a more robust power and timing setup. The ESP32-CAM and servo impose varying loads, and using a dedicated external 5 V source with proper grounding would help stabilize performance during simultaneous detection and actuation. Additionally, implementing smoothing algorithms or interrupt-based timing for servo control can significantly improve motion stability. These recommendations will help increase detection reliability, improve processing efficiency, and create a more dependable face-responsive system.

9.0 CONCLUSION

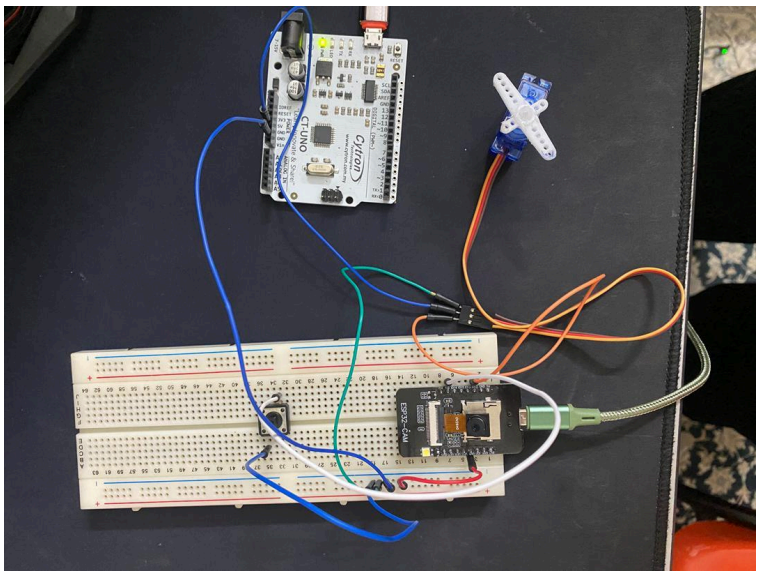
In conclusion, the integration of the ESP32-CAM's face detection capabilities demonstrated the module's potential for simple intelligent surveillance applications, successfully detecting human faces and triggering additional actions such as servo movement or image capture. Although changing the XCLK from 20 MHz to 8 MHz improved stability, the system's performance remained sensitive to lighting, processing load, and camera configuration. Despite these limitations, the experiment showed that effective face-responsive behaviour can be achieved on a low-cost embedded platform, highlighting the importance of careful parameter tuning, efficient power management, and optimized system logic for reliable real-time operation.

10.0 REFERENCES

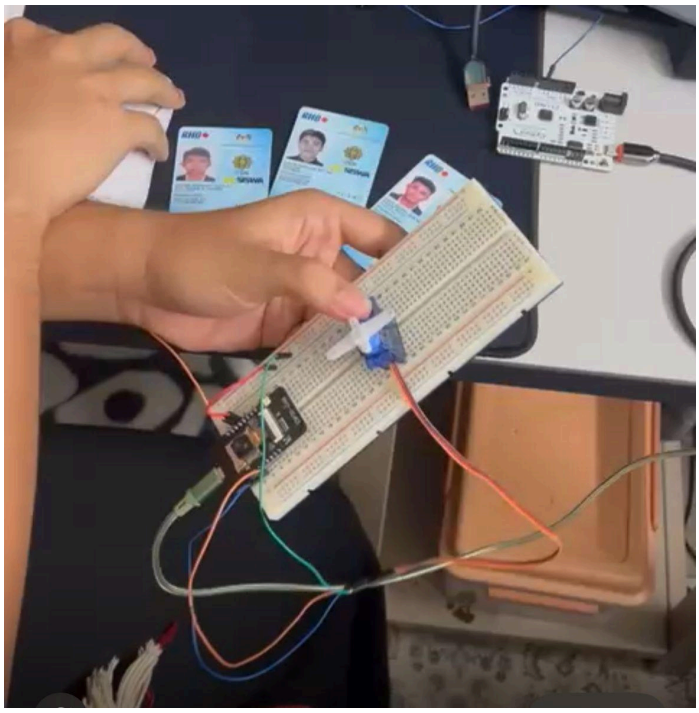
1. Nabila Nadzelan. (2023, April 2). Getting Started with ESP32-CAM. Cytron Technologies Malaysia; Cytron Technologies. https://my.cytron.io/tutorial/getting-started-with-esp32-cam?gad_source=1&gad_campaignid=21520410146&gbraid=0AAAADh9ZNBLaA-RhCqbegT-OeM4JllVf&gclid=Cj0KCQiAiebIBhDmARIsAE8PGNK2X22sonL8_DTJP8X00lj2kNE6WOXsxVI06-bCwAbNHFJGwfKaCGAaAq W3EALw_wcB
2. ESP32 Servo Motor Web Server with Arduino IDE | Random Nerd Tutorials. (2018, May 1). https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/
3. ESP32-CAM Video Streaming and Face Recognition with Arduino IDE. (2019, December 10). Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/

11.0 APPENDICES

EXP 6A



EXP 6B



Acknowledgement

We would like to express our sincere gratitude to Dr. Zulkifli Bin Zainal Abidin and Dr. Wahju Sediono for their continuous support and for providing the essential resources and facilities required to carry out this project. We also extend our heartfelt appreciation to everyone who contributed to the success of this lab report through their valuable insights, assistance, and feedback.

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 that no further improvement on the reports is needed from any of the individual contributors to the report.

Signature:

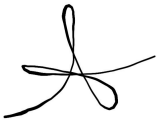
Read [/]

Name: AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA
Matric Number: 2315377

Understand [/]
Agree [/]

Signature:

Read [/]



Name: ADIB ZAFFRY BIN MOHD ABDUL RADZI
Matric Number: 2315149

Understand [/]
Agree [/]

Signature:

Read [/]

Name: UMAR FAROUQI BIN ABU HISHAM
Matric Number: 2314995

Understand [/]
Agree [/]