MECHATRONICS SYSTEM INTEGRATION

MCTA 3203

WEEK 6:

SMART SURVEILLANCE SYSTEM USING ESP32-CAM

SECTION 1

SEMESTER 1, 2025/20256

INSTRUCTOR:

ASSOC. PROF. EUR. ING. IR. TS. GS. INV. DR. ZULKIFLI BIN ZAINAL ABIDIN

DR. WAHJU SEDIONO

DATE OF SUBMISSION: 19TH NOVEMBER 2025

GROUP 8

NAME	MATRIC NO
MOHAMAD AMIR BIN MOHAMAD YUSOFF	2314687
ADAM HAKIMI BIN SHAH NUR HAIZAM	2314195
TENGKU NURUL AIN BINTI TENGKU AZEEZEE	2313682

Abstract

This experiment focuses on developing a smart surveillance system using the ESP32-CAM module integrated with a servo motor for horizontal camera panning. The ESP32-CAM enables live video streaming over Wi-Fi, while the servo motor allows controlled movement of the camera to simulate basic motion tracking. By programming the microcontroller and adjusting pulse width modulation (PWM) signals, the servo's rotation can be precisely managed. Through this experiment, we learned how to interface the ESP32-CAM for video streaming, implement servo motor control using Arduino IDE, and integrate manual input via a pushbutton for motion-triggered panning. The experiment successfully demonstrated how wireless communication and actuator control can be combined in a mechatronic system for practical surveillance applications.

Table Of Contents

Introduction

Controlling camera orientation and streaming live video are fundamental aspects of modern surveillance and IoT-enabled mechatronic systems, as they enable real-time monitoring and motion tracking. Such systems are widely used in applications including home security, automated monitoring, and robotic vision platforms. Direct control of a servo motor using a microcontroller requires careful management of position through Pulse Width Modulation (PWM), while video streaming requires proper configuration of the camera module and network communication.

In this experiment, the ESP32-CAM module was interfaced with a servo motor and a pushbutton to build a basic smart surveillance system. The ESP32-CAM provides live video streaming over Wi-Fi, and the servo motor allows horizontal panning of the camera. PWM signals were used to control the servo's rotation angle precisely, while the pushbutton enabled manual motion-triggered control.

The objectives of this experiment were to understand the working of the ESP32-CAM module, control a servo motor using Arduino GPIO pins and PWM, integrate manual input through a pushbutton, and implement a web-based interface for live video streaming. The hypothesis was that the servo would only pan when the pushbutton was pressed, and the ESP32-CAM would successfully stream live video to a connected device. This experiment provided hands-on experience integrating sensors, actuators, microcontrollers, and wireless communication, reinforcing theoretical concepts of PWM, servo control, and IoT-based mechatronic system design.

Task 1

Materials and Equipment

1. ESP32-CAM Module
2. Arduino Uno
3. USB-to-Serial Adapter (FTDI)
4. Servo Motor
5. Pushbutton
6. 5V Power Supply
7. Jumper Wires
8. Breadboard
9. Mounting Bracket or Servo Base

Experimental Setup

1. The experiment used an ESP32-CAM module integrated with an SG90 servo motor and a pushbutton to implement manual control of camera panning.
2. The ESP32-CAM was powered using a stable 5V 2A supply to prevent brownouts during servo movement and camera operation.
3. The SG90 servo motor was connected to GPIO 12 for horizontal rotation, with its VCC and GND lines connected to the 5V and GND pins of the ESP32-CAM.
4. A pushbutton was installed between GPIO 13 and GND, while an internal pull-up resistor was activated in software to ensure reliable button press detection.
5. The ESP32-CAM was programmed through an USB-to-Serial Adapter (FTDI), with IO0 held LOW during uploading to enter flashing mode.

6. After uploading the program, the ESP32-CAM connected to Wi-Fi and hosted a live video stream while the servo remained stationary unless the button was pressed.
7. The system was tested by viewing the video feed in a web browser and pressing the pushbutton to trigger servo panning, verifying correct coordination between hardware control and the camera stream.

Methodology

1. The ESP32-CAM was prepared by supplying 5V 2A through an external power source to ensure stable camera and servo performance.
2. The servo motor was wired to GPIO 12, with 5V and GND supplying power directly from the ESP32-CAM module.
3. A pushbutton was connected from GPIO 13 to GND, and the internal pull-up resistor was enabled in software to ensure a HIGH-to-LOW transition only when the button was pressed.
4. A serial adapter was used to upload the control program, and IO0 was grounded during flashing to place the ESP32-CAM into programming mode.
5. The control code was developed in Arduino IDE to initialize the camera, start the web server for video streaming, and read the digital input from the button.
6. PWM signals were generated on GPIO 12 to control servo rotation, but movement occurred only when a button press was detected.
7. Testing was performed by opening the ESP32-CAM's IP address in a web browser and simultaneously pressing and releasing the button to observe the servo movement.

8. Results were recorded to confirm that the servo only panned during button presses, demonstrating successful integration of manual control with the surveillance system.

Circuit Assembly

1. The ESP32-CAM module was connected to a regulated 5V 2A power supply to ensure stable camera and servo operation.
2. The servo motor was connected to GPIO 12 of the ESP32-CAM, with its VCC and GND wired directly to the module's 5V and GND pins.
3. A pushbutton was placed between GPIO 13 and GND, with an internal pull-up resistor activated in software to prevent false triggering.
4. Programming and serial monitoring were performed using an adapter, with IO0 held LOW during code uploading.
5. With this wiring configuration, the ESP32-CAM was able to generate PWM signals for controlling the servo motor, detect pushbutton presses for manual panning activation, and simultaneously perform live video streaming over Wi-Fi.

Programming Logic

1. The program initialized GPIO 12 as the PWM output for servo control and GPIO 13 as the digital input for the pushbutton, with the internal pull-up resistor enabled.
2. PWM signals were generated using the ledcWrite() function, where specific duty cycle values corresponded to the servo's rotation angles.
3. The pushbutton state was read using digitalRead(), allowing the servo to move only when the button was pressed (LOW state due to the pull-up configuration).

4. The panning direction was controlled by incrementing or decrementing the PWM value, reversing direction when the servo reached its defined minimum or maximum position.
5. Inside the main loop, the program continuously monitored the pushbutton, updated the servo position when pressed, and maintained uninterrupted video streaming over the local Wi-Fi network.
6. The program continued running until powered off, enabling consistent observation of servo panning triggered by manual button input while the live video feed remained active.

Code Used

Arduino Code:

```
#include <ESP32Servo.h>
#define BUTTON_PIN 13
#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:

   The ESP32 module is connected to a servo motor and a pushbutton to allow manual horizontal panning.

   The system uses the following pin assignments:

   1. Servo Motor: Control → GPIO 12
   2. Pushbutton: Input → GPIO 13

   The ESP32 provides PWM control signals to the servo motor, while a 5V power source supplies stable voltage for reliable movement and user interaction through the pushbutton.

2) Global Variables:

   Several global variables are defined to track servo movement, direction, and button state:

   a) const int BUTTON_PIN = 13; → Pushbutton input pin
   b) const int SERVO_PIN = 12; → Servo control pin
   c) int servoPos = 0; → Current servo angle (0°–180°)
   d) bool movingForward = true; → Direction of servo sweep (true = increasing angle)

These variables determine the servo's rotation angle, direction of panning, and how the system responds when the pushbutton is pressed.

3) Setup Function:

During initialization, the ESP32 configures all required pins and attaches the servo:

a) pinMode(BUTTON_PIN, INPUT_PULLUP) enables the internal pull-up resistor to ensure stable pushbutton input.
b) The servo is configured using:

   - myservo.setPeriodHertz(50) to set a 50 Hz PWM frequency.

   - myservo.attach(SERVO_PIN, 1000, 2000) to define the servo signal pulse range.
c) The initial servo position is set using myservo.write(servoPos).
d) Serial communication is started for monitoring with:

   - Serial.begin(115200);

   - Serial.println("Fast Bounce Servo Test Start")

This configuration prepares the ESP32 to detect button presses and generate proper PWM signals to control servo rotation.

4) Main Loop:

The loop() function repeats continuously and performs the following operations:

I. Button Check:

a) The pushbutton state is read using digitalRead(BUTTON_PIN).
b) If the button is pressed (state == LOW), the servo initiates horizontal panning.

c) If not pressed, the servo remains at its last position.

II. Servo Panning:

a. The servo angle changes in increments of 5° per loop:

- If movingForward == true, the angle increases.
- If the angle reaches 180°, the direction reverses.
- If the angle reaches 0°, the direction reverses again.

b. Updated angles are written to the servo using:

myservo.write(servoPos)

c. A 50 ms delay smooths the motion and prevents abrupt jerks.

III. Serial Monitoring

a. Optional feedback is printed to the Serial Monitor:

- Servo angle: <current_angle>

This allows real-time observation of the servo's exact position during panning.

IV. End of Loop:

a. The loop repeats indefinitely while the board is powered.

The servo only moves when the pushbutton is actively pressed, ensuring full manual control over panning.

5) PWM Control Principle:

The servo's rotation is controlled by varying the PWM pulse width:

a) Longer pulses (near 2000 µs) → Servo rotates toward 180°

b) Shorter pulses (near 1000 µs) → Servo rotates toward 0°

By incrementing or decrementing the servo angle in small steps, the system achieves a smooth and responsive back-and-forth panning motion activated entirely through the pushbutton.

<u>Data Collection.</u>

During the experiment, the movement of the servo motor was observed manually while the pushbutton was pressed. Each press activated the servo to sweep between 0° and 180° in steps of approximately 5°, as defined in the program. Slow and continuous button presses allowed the servo to complete repeated back-and-forth panning motion. The servo's response, direction reversal, and stability were visually inspected to confirm that the movement followed the programmed logic.

<u>Data Analysis</u>

From the observations, the servo consistently moved only when the pushbutton was pressed (LOW state due to pull-up configuration). When pressed, the servo angle increased until it reached the upper limit of 180°, after which the direction reversed automatically. Similarly, it swept backward until reaching 0°, then reversed direction again. The behavior verified correct reading of the pushbutton input, proper PWM generation using the ESP32Servo library, and smooth direction switching based on the defined boundaries.

Results

The experiment successfully demonstrated manual control of a servo motor using a pushbutton. The servo remained stationary when the button was not pressed and moved in a smooth back-and-forth manner when activated. Direction reversal at 0° and 180° occurred reliably.

The system behaved as expected: the pushbutton effectively enabled and disabled servo panning, confirming proper integration of digital input, PWM signal control, and mechanical response. The results validated that the ESP32 correctly executed the programmed logic for servo actuation based on user input.

Task 2

Materials and Equipment

1. ESP32-CAM Module
2. Arduino Uno
3. USB-to-Serial Adapter (FTDI) or CH340 USB to TTL Serial Cable
4. SG90 or MG90S Servo Motor
5. 5V Power Supply
6. Jumper Wires
7. Breadboard
8. Mounting Bracket or Servo Base

Experimental Setup

1. The experiment used an ESP32-CAM module equipped with an OV2640 camera and a servo motor to implement a basic face-triggered motion system.
2. The ESP32-CAM was powered using a 5V 2A external supply to ensure stable performance during continuous image processing and Wi-Fi streaming.
3. The servo motor was connected to GPIO 12 for rotation, while its VCC and GND pins were connected to the 5V and GND terminals of the ESP32-CAM.
4. The servo movement was triggered entirely by the ESP32-CAM's built-in face detection.
5. The ESP32-CAM was connected to a computer via an adapter for programming, with IO0 held LOW during the code upload process.
6. After uploading the program, the ESP32-CAM connected to Wi-Fi and hosted a web server that displayed a live video stream with face detection enabled.

7. The setup was tested by moving a face into the camera's field of view and observing whether the servo rotated whenever a face was detected, verifying correct integration between image processing and actuator response.

Methodology

1. The ESP32-CAM module was first prepared and powered with a 5V 2A supply to support simultaneous camera operation, video streaming, and servo movement.
2. The servo motor was wired to GPIO 12 and connected to the ESP32-CAM's 5V and GND pins to supply the required operating voltage.
3. Programming was done using an adapter, with IO0 grounded during code uploading to enter flash mode.
4. The Arduino IDE was used to upload a program that initialized the ESP32-CAM, enabled face detection, and activated the servo motor whenever a face appeared in the captured frame.
5. In the program, the camera continuously scanned for faces, and once a face was detected, a command was issued to rotate the servo to a predetermined angle or through a predefined sweep.
6. Testing was performed by accessing the ESP32-CAM's live video stream in a web browser and observing the servo's reaction whenever a face entered or exited the view.
7. Observations were recorded to confirm that the servo rotated only when a face was detected and remained still otherwise, demonstrating correct synchronization between machine vision and servo activation.

Circuit Assembly

1. The ESP32-CAM module was connected to a stable 5V 2A power supply to ensure reliable camera processing and Wi-Fi streaming. The servo motor was assembled as follows:
   - Signal → GPIO 12
   - VCC → 5V
   - GND → GND
2. The onboard OV2640 camera module was utilized for image processing.
3. The ESP32-CAM was interfaced to a computer with an adapter for code uploading and serial monitoring, with IO0 grounded during flashing.
4. With this wiring, the ESP32-CAM was able to simultaneously perform face detection, rotate the servo motor, and stream video over Wi-Fi.
5. The onboard OV2640 camera module was utilized for image processing.

Programming Logic

1. The ESP32-CAM program began by initializing the camera and enabling the built-in face detection function within the video stream. The servo motor was attached to GPIO 12 using the ESP32Servo library, with an initial angle defined.
2. During operation, each video frame was scanned for facial features. When the face-detection algorithm identified a face in the image, the program triggered the servo motor to rotate to a specific angle or perform a short sweep as a response action.
3. If no face was detected, the servo remained stationary at its default position.

4. The logic flow operated continuously inside the streaming process, ensuring that the servo reacted immediately when a face appeared.
5. Throughout the program, the ESP32-CAM maintained live video streaming over Wi-Fi, and servo activation was fully automated without requiring any manual input.
6. Serial monitoring could optionally be used to display detection events, servo activation messages, and system status.
7. The system continued to operate until manually stopped, allowing continuous observation of face detection and corresponding servo activation behavior.

Code Used

Arduino Code is uploaded at GitHub

Control Algorithm

1. Defining Pins:

   The ESP32-CAM module is connected to a servo motor to perform automatic horizontal panning whenever a face is detected.

   The system uses the following pin assignments:

   - Servo Motor: Control → GPIO 12

   The ESP32 generates PWM signals to control the servo motor, while a 5V power source provides stable voltage for reliable movement.

2. Global Variables:

   Several global variables are defined to track servo movement and face detection state:

a) const int SERVO_PIN = 12; for servo control pin

b) int servoPos = 0; for current servo angle (0°–180°)

c) bool movingForward = true; For direction of servo sweep (true = increasing angle)

d) bool faceDetected = false; For Face detection flag

These variables determine the servo's rotation angle, sweep direction, and whether it should rotate in response to face detection.

3. Setup Function:

During initialization, the ESP32 configures all required pins and attaches the servo:

a) myservo.setPeriodHertz(50); to set 50 Hz PWM frequency

b) myservo.attach(SERVO_PIN, 1000, 2000); to define servo pulse range

c) myservo.write(servoPos); to set initial servo position

Serial communication is started for monitoring:

a) Serial.begin(115200);

b) Serial.println("Face Detection Servo Test Start");

This prepares the ESP32 to generate proper PWM signals and allows real-time monitoring of servo movement.

4. Main Loop:

The loop() function repeats continuously and performs the following operations:

I. Face Detection Check:

a) The ESP32-CAM captures images and runs face detection algorithms.

b) If a face is detected, faceDetected = true.

c) If no face is detected, faceDetected = false.

II. Servo Panning:

a) The servo rotates in 5° increments whenever a face is detected:

- If movingForward == true, the angle increases.
- If the angle reaches 180°, the direction reverses.
- If the angle reaches 0°, the direction reverses again.

b) Updated angles are written to the servo using:

```
myservo.write(servoPos);
```

c) A 50 ms delay smooths the motion and prevents abrupt jerks.

III. Serial Monitoring

a) Optional feedback is printed to the Serial Monitor:

- Serial.print("Servo angle: ");
- Serial.println(servoPos);

This allows observation of the servo's position in real-time.

IV. End of Loop:

a) The loop repeats indefinitely while the board is powered.

b) The servo only moves when a face is detected, ensuring activation is entirely automatic and face-triggered.

5. PWM Control Principle:

   The servo's rotation is controlled by varying the PWM pulse width:

   d) Longer pulses (near 2000 µs) → Servo rotates toward 180°

   e) Shorter pulses (near 1000 µs) → Servo rotates toward 0°

   Incrementing or decrementing the servo angle in small steps produces smooth and responsive back-and-forth rotation whenever a face is detected.

Data Collection.

Face detection performance was observed using the live video stream hosted by the ESP32-CAM. When a face entered the camera's field of view, the servo motor automatically rotated. Observations included how quickly the servo responded to face movements, the smoothness of the servo rotation, and whether the bounding box in the video stream accurately tracked the face. The effect of different lighting conditions and distances from the camera on both face detection and servo response was also noted.

Data Analysis

Analysis of the system showed that the ESP32-CAM successfully detected faces and triggered the servo to rotate whenever a face was present. The servo moved smoothly in response to changes in face position, demonstrating effective integration between the camera's detection algorithm and the servo control logic. Detection accuracy decreased slightly under low lighting or rapid movements, causing minor delays in servo rotation, which is consistent with processing limitations of the ESP32-CAM. Overall, the system reliably linked face detection events to servo motion in real time.

<u>Results</u>

The ESP32-CAM system successfully performed automated face detection and triggered servo rotation. The bounding boxes consistently appeared around detected faces in the video stream, and the servo rotated correspondingly. The system maintained smooth video streaming while controlling the servo, demonstrating that both real-time image processing and motor actuation were functioning correctly. This confirms that the ESP32-CAM module can be effectively used in smart surveillance applications where camera rotation is activated by face detection events.

Discussion

1. Interpretation of Results and Their Implications

The experiment successfully demonstrated that microcontrollers can be used to integrate sensors and actuators in a mechatronic system.

In Task 1, the ESP32-CAM module controlled an SG90 servo motor, allowing horizontal panning only when the pushbutton was pressed. The servo responded reliably to PWM signals, moving smoothly between minimum and maximum angles. This confirms that GPIO input and PWM output can accurately control physical motion. The internal pull-up resistor ensured stable pushbutton readings, providing consistent user interaction.

In Task 2, the ESP32-CAM successfully detected faces in real time, highlighting them with bounding boxes on the live video feed. The optional servo activation in response to detected faces demonstrated the feasibility of automated motion tracking, suitable for basic IoT surveillance applications. The ESP32-CAM efficiently handled both video streaming and image processing simultaneously over a Wi-Fi network. Overall, the results confirm that simple pushbutton input and image-based sensor data can effectively drive actuator responses in IoT-based surveillance systems.

2. Discrepancies Between Expected and Observed Outcomes

Several minor discrepancies were observed during testing:

I. Servo Overshoot and Jitter:

In Task 1, the servo occasionally overshot its target due to inertia during rapid angle changes. Small angle increments at low PWM values caused slight jitter.

II. Face Detection Accuracy:

In Task 2, detection accuracy decreased under low lighting conditions or when subjects moved rapidly.

III. Servo Response Lag:

Servo activation triggered by face detection sometimes lagged, likely due to processing delays and limitations in the video frame rate.

Despite these minor discrepancies, the experiment results closely followed the expected behavior of servo and face detection control, demonstrating the ESP32-CAM's ability to accurately integrate sensor input and actuator response in an IoT-based surveillance system.

3. Sources of Error and Limitations of the Experiment

Several factors may have influenced the accuracy and performance of the system:

- Environmental Factors:

  Lighting conditions and facial orientation affected detection reliability.
- Mechanical Constraints:

  Friction or alignment issues in the servo bracket affected smooth motion.
- Hardware Replacement:

  Replacing the ESP32-CAM module between tasks may have caused minor performance differences.

Conclusion

The experiment successfully demonstrated the integration of microcontrollers, sensors, and actuators into a functional IoT-based surveillance system. Task 1 validated PWM motor control and GPIO input functionality through pushbutton-controlled servo panning, showing that physical motion could be reliably controlled by user input. Task 2 confirmed that the ESP32-CAM could accurately detect faces in real time and display them on a live video feed, highlighting the module's capability to process sensor data and transmit it over a Wi-Fi network simultaneously. The use of different ESP32-CAM modules between tasks emphasized the importance of careful power management and wiring to ensure consistent performance. Overall, the lab reinforced hands-on understanding of PWM control, sensor integration, microcontroller programming, and wireless communication in mechatronic systems, providing practical experience in building and troubleshooting basic IoT-based automation and surveillance applications.

<u>Recommendations</u>

To improve the accuracy and overall quality of future experiments, the following recommendations are suggested:

1. Improve Power Management

   Use a dedicated 5V 2A supply for the servo and camera to ensure stable operation and reduce jitter or unexpected resets.
2. Enhance Connectivity

   Position the ESP32-CAM closer to the router or use a Wi-Fi extender to minimize streaming delays and improve video consistency.
3. Implement Sensor-Triggered Automation

   Integrate a PIR motion sensor or enable ESP32-CAM's built-in face detection to automate camera movement, making the system more intelligent and less reliant on manual input.

References

[1] R. Santos and S. Santos, “How to Program / Upload Code to ESP32‑CAM AI‑Thinker (Arduino IDE),” *Random Nerd Tutorials*, 2020. [Online]. Available: https://randomnerdtutorials.com/program-upload-code-esp32-cam/ Random Nerd Tutorials+1

[2] Cytron Technologies Sdn. Bhd., “CH340 USB to TTL Serial Cable,” *Cytron My*, 2024. [Online]. Available: https://my.cytron.io/p-ch340-usb-to-ttl-serial-cable Cytron Technologies Singapore+1

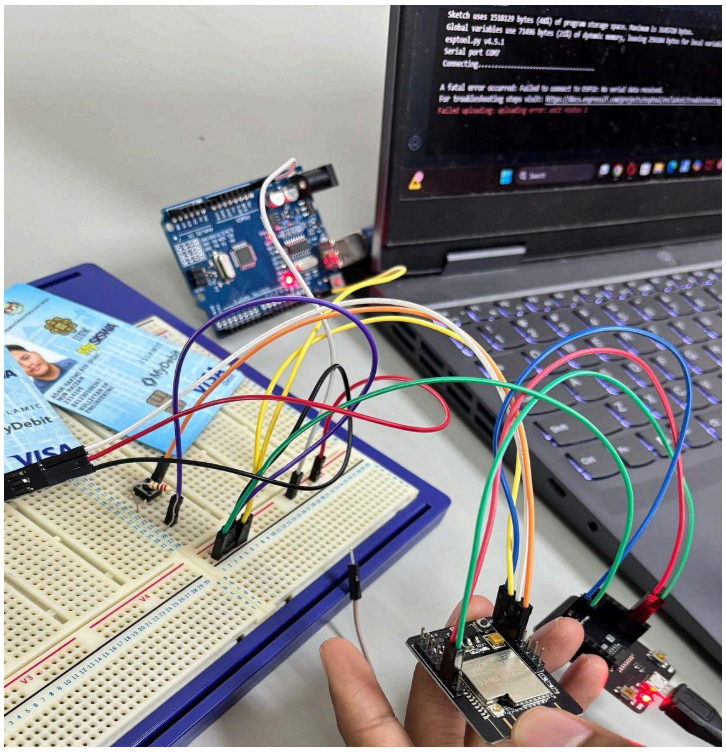
[3] A. Hussain, “Program ESP32‑CAM using FTDI adapter,” *Arduino‑er Blog*, 5 Sept. 2020. [Online]. Available: https://arduino-er.blogspot.com/2020/09/program-esp32-cam-using-ftdi-adapter.html arduino-er.blogspot.com

[4] “Program ESP32‑CAM using FTDI adapter,” YouTube, 19 Oct. 2020 (approx.). [Online]. Available: https://www.youtube.com/watch?v=D3MPBPGT3cw YouTube

[5] “Use a ESP32‑CAM Module to Stream HD Video Over Local Network,” *Core Electronics Guides*. [Online]. Available: https://core-electronics.com.au/guides/esp32-cam-set-up/

Appendices

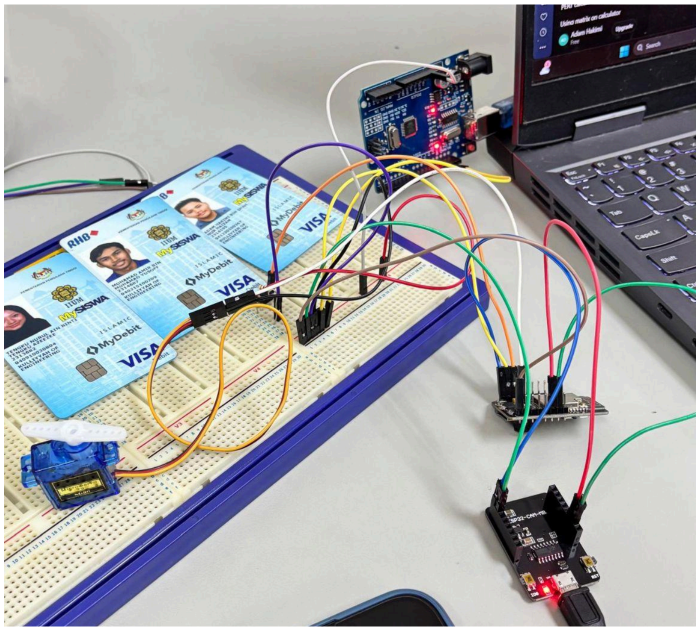
Diagram below shows the circuit for Task 1:



```
Sketch uses 1518129 bytes ([illegible]) of program storage space. Maximum is [illegible] bytes.
Global variables use [illegible] bytes ([illegible]) of dynamic memory, leaving [illegible] bytes for local
esptool.py v4.5.1
Serial port COM7
Connecting......................................

A fatal error occurred: Failed to connect to ESP32: No serial data received.
For troubleshooting steps visit: https://docs.espressif.com/projects/esptool/en/latest/
Failed uploading: uploading error: exit status 2
```

Diagram below shows the circuit for Task 2:



Acknowledgments

Alhamdulillah, all praise and gratitude be to Allah SWT for His blessings, guidance, and mercy that enabled us to successfully complete this project. Throughout the development process, we faced several challenges, but with His will and guidance, we managed to overcome them and complete the project successfully.

We would also like to express our heartfelt appreciation to our lecturer for their continuous guidance, support, and valuable feedback throughout this project. Their advice has greatly helped us in understanding both the theoretical and practical aspects of microcontroller systems and digital display design.

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.**

Signature: Amir	Read [/]
Name: Mohamad Amir Bin Mohamad Yusoff	Understand [/]
Matric Number: 2314687	Agree [/]
Signature: Adam	Read [/]
Name: Adam Hakimi Bin Shah Nur Haizam	Understand [/]
Matric Number: 2314195	Agree [/]
Signature: Ain	Read [/]
Name: Tengku Nurul Ain Binti Tengku Azeezee	Understand [/]
Matric Number: 2313682	Agree [/]