Advanced Chicken House

Written by Jiahao Chen 20537, Hoi-Fung Sam 30875, Jianhua Wu 20608

1.Introduction

Aim of the project:

In this project, we are going to make a house for chickens in order for scientists to study their living habits. The basic idea of this chickens' house is to first, capture motion of the chicken by PIR motion sensor. Second, the PIR sensor transmits a signal to the camera and then the camera takes a photo of the chicken. In the end, the camera that is connected to Wifi will send the photo to the user via email. The energy supply of this project comes from a solar panel, which means the project is totally environment friendly.

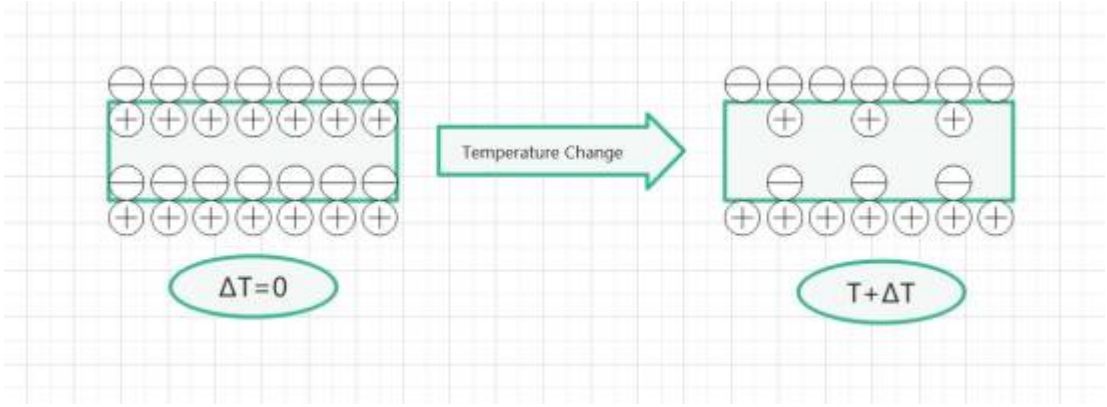
2.Methods and Materials

2.1 Materials:

1. breadboard
2. female jumper wires
3. male to female jumper wires
4. HC-SR501 PIR Sensor
5. ESP32-camera
6. FTDI programmer
7. solar panel
8. battery
9. Adafruit Universal USB / DC / Solar Lithium Ion/Polymer charger - bq24074

2.1.1 HC-SR501 PIR Sensor

PIR stands for Passive Infrared Sensor. A PIR sensor consists of two parts: Pyroelectric Sensor and Fresnel Lens.
Pyroelectric infrared sensors can detect infrared rays emitted by warm objects like people or certain animals and convert them into electrical signals for output. When the crystals at the two slots are heated, positive differential charges (charges of equal quantity but opposite signs) will be generated at both ends of the crystal slots. This electric polarization phenomenon due to thermal changes is called the pyroelectric effect. Usually, the bound charges generated by the spontaneous polarization of the crystal are neutralized by the free electrons attached to the surface of the crystal from the air, and the spontaneous polarization moment cannot be shown. When the temperature changes, the positive and negative charges in the crystal structure will be shifted, and the phenomenon of running off charges occurs on the crystal surface. To be more specific, the state of charge depletion is proportional to the degree of polarization.



The pyroelectric sensor is sensitive to temperature. It is made of ceramic oxide or piezoelectric crystal elements. The two surfaces of the element are made of electrodes. When the temperature changes by ΔT, the pyroelectric effect will generate a charge of ΔQ on the two electrodes. A weak voltage ΔV is generated between the electrodes. The charge of ΔQ generated by the pyroelectric effect will combine with the ions in the air and disappear. When the ambient temperature is stable, as ΔT=0, the sensor has no output. When the human or chicken body enters the detection area, ΔT is generated due to the difference between the human body temperature and the ambient temperature. If the human or chicken body does not move after entering the detection area, the temperature does not change, which means the sensor has no output. So it is the principle of RIP sensor detecting human body or animal activity.

Pir Sensor Output
There are three Pinout: GND, OUT, VCC.
VCC is the power supply of the sensor. In our project, we connect the 5V to this pin.
Output is the TTL logic pin that transfers the detecting motion. When the motion of a warm body is strong, it goes HIGH. On the contrary, it goes LOW when no motion is detected.
GND stands for GROUND, it works with VCC and comes up with a close circuit.

Advantages:

1. It does not emit any type of radiation.
2. The power consumption of the device is small, which is easy to install in the chicken house.
3. Inexpensive.

Disadvantages:

1. Easy to be interfered with by various heat sources and light sources.
2. Vulnerable to interference from radio frequency radiation.
3. When the ambient temperature is close to the human body temperature, the detection and sensitivity will drop obviously, sometimes resulting in short-term failure. For example, the body temperature of chicken is 41 ℃, the temperature of my hometown Nanjing, China in summer is up to 40 ℃. The temperature difference is too low for the sensor to detect the motion of chicken.

Lens:
In order to make the detection area larger, a special but low cost technology called Fresnel Lens is used. The Fresnel Lens condenses light, which provides a larger IR range for the sensor. The cover of the PIR sensor consists of multiple small Fresnel Lens.

2.1.2 ESP32-camera

The ESP32-CAM is a popular development board designed for applications requiring a camera module. It includes multiple data pins, a combined Wi-Fi and camera module, and a microSD card slot for easy storage. In this project we mainly use the Wi-Fi function and following pins: Power Pins (5V, 3.3V and GND), general purpose input/output pins (GPIO12), serial pins (U0R, U0T) and GPIO0.

2.1.3 Adafruit Universal USB / DC / Solar Lithium Ion/Polymer charger - bq24074

2.2 Method:

2.2.1 Wire connection to upload the code

ESP32CAM - FTDI Programmer
U0R - TXD
U0T - RXD
GND - GND

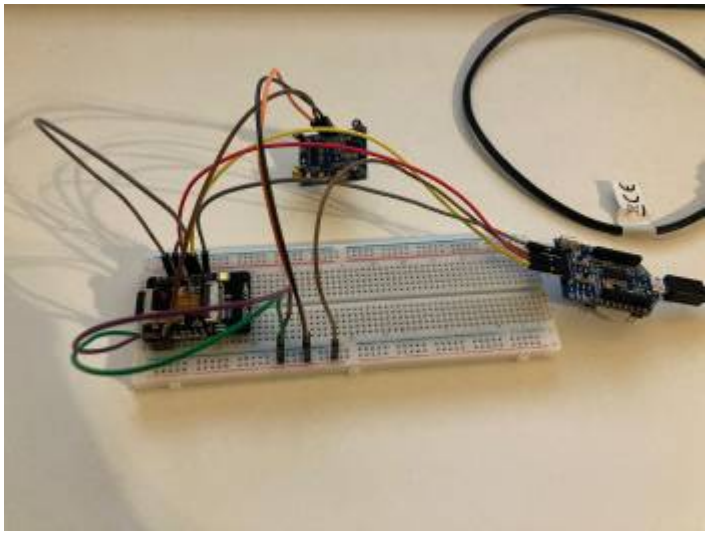
ESP32CAM - Breadboard
5V - positive rail of the breadboard
GND - negative rail of the breadboard

ESP32CAM - PIR
GPIO12 - OUT

PIR - Breadboard
VCC - positive rail of the breadboard
GND - negative rail of the breadboard

FTDI Programmer - Breadboard
VCC - positive rail of the breadboard

Flashing mode
GPIO0 - GND

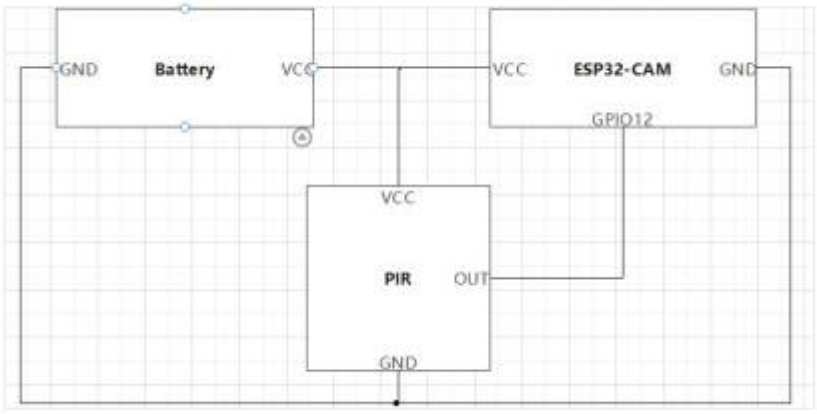


2.2.2 The principle of connecting the ESP32-CAM and FTDI Programmer:

Because ESP32-CAM does not have a built-in programmer, it is necessary to upload the code to ESP32 before connecting the board and PIR sensor. Here we use FTDI as an information transfer station for code transmission.
To transfer code or data from the computer to the ESP32 using the FTDI converter, you need to establish UART communication between the FTDI chip and the ESP32. UART (Universal Asynchronous Receiver/Transmitter) is a popular serial communication protocol that allows two devices to send and receive data asynchronously.

2.2.3 How to connect and transmit data between FTDI and ESP32-CAM:

Connect the TX pin of the FTDI to the RX (U0R) pin of the ESP32. This connection allows data transmitted from the FTDI to be received by the ESP32.
Connect the RX pin of the FTDI to the TX (U0T) pin of the ESP32. This connection allows data transmitted from the ESP32 to be received by the FTDI.
Connect VCC to 5V and GND to GND to form a circuit to supply current.
Finally, connect GPIO0 and another GND pin. The purpose of this step is to change the boot mode of the ESP32. After the ESP32, FTDI and computer are all connected, press the “Reset” button on the ESP32, and then the board will enter programming mode. At this point, with the Arduino running, the code data can be sent to the board by TX and RX and saved. After the Arduino shows finished, disconnect GPIO0 and GND and press the reset button again to return the board to normal working mode and use.



2.2.4 Wire connection in operation

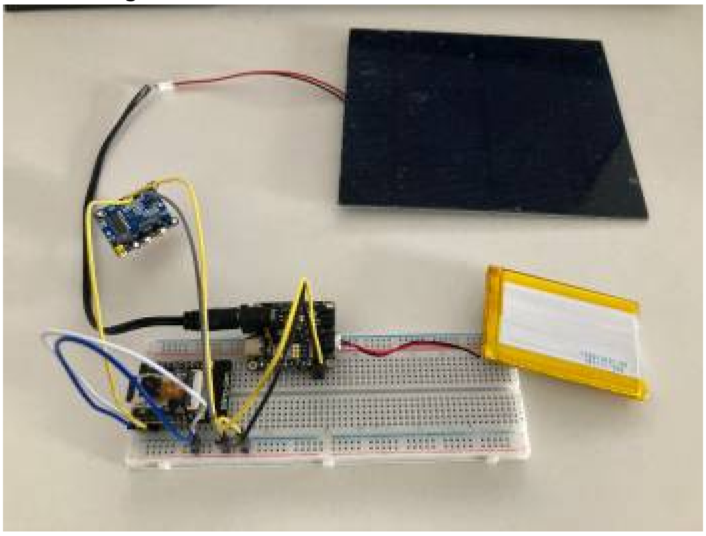
ESP32CAM - Breadboard
5V - positive rail of the breadboard
GND - negative rail of the breadboard

ESP32CAM - PIR
GPIO12 - OUT

PIR - Breadboard
VCC - positive rail of the breadboard

GND - negative rail of the breadboard

Lithium Ion/Polymer charger - Breadboard
LiPo - positive rail of the breadboard
GND - negative rail of the breadboard



3.Code

References for this project are from:
Arduino IDE + ESP32 CAM ESP32-CAM Capture and Send Photos Via Email using an SMTP Server and PIR
Arduino IDE + ESP32 CAM Capture and Send Photos Via Email using an SMTP Server and PIR (Update)

```
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>> 05 ESP32 Cam Capture and Send Photos Via Email with
PIR (Without LED Flash)
/*********
  Rui Santos
  Complete instructions at
https://RandomNerdTutorials.com/esp32-cam-projects-ebook/

  Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files.
  The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
*********/

/*
 * Uteh Str
 *
 * References for this project are from :
https://randomnerdtutorials.com/esp32-cam-send-photos-email/
 * I made some modifications and combinations of the above references to
create this project.
 */
```

```
/* ========================================== Including the libraries */
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>
/* ========================================== */

/* ========================================== Defining variables for Email */
/*
 * Specifically for Gmail users :
 * - To send Email using Gmail use port 465 (SSL) and SMTP Server
smtp.gmail.com
 * - Especially for the Sender's Gmail account, so that ESP32 CAM can log
into the sender's Gmail account,
 *   the Sender's Gmail account must activate 2-Step Verification then get
"App Passwords". The method is in the video. Watch carefully.
 */
#define emailSenderAccount      "amc491561@gmail.com@gmail.com"
#define emailSenderAppPassword  "vsvnciqerihrhurx"
#define smtpServer              "smtp.gmail.com"
#define smtpServerPort          465
#define emailSubject            "ESP32-CAM Photo Captured"
#define emailRecipient          "YOUR_EMAIL_RECIPIENT@gmail.com"
/* ========================================== */

/* ========================================== Defining the Camera Model and
GPIO */
#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1
  #define XCLK_GPIO_NUM      0
  #define SIOD_GPIO_NUM     26
  #define SIOC_GPIO_NUM     27

  #define Y9_GPIO_NUM       35
  #define Y8_GPIO_NUM       34
  #define Y7_GPIO_NUM       39
  #define Y6_GPIO_NUM       36
  #define Y5_GPIO_NUM       21
  #define Y4_GPIO_NUM       19
  #define Y3_GPIO_NUM       18
  #define Y2_GPIO_NUM        5
  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22
```

```
#else
  #error "Camera model not selected"
#endif
/* ======================================== */

#define FILE_PHOTO "/photo.jpg" //--> Photo File Name to save in SPIFFS

#define pin_Pir 12 //--> PIR Motion Detector PIN

/* ======================================== Variables for network */
// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "iotlab";
const char* password = "iotlab18";
/* ======================================== */

SMTPData smtpData; //--> The Email Sending data object contains config and
data to send

/*
_____________________________________________________________________________
____ Function to read PIR sensor value (HIGH/1 OR LOW/0) */
bool PIR_State() {
  bool PRS = digitalRead(pin_Pir);
  return PRS;
}
/*
_____________________________________________________________________________
____ */

/*
_____________________________________________________________________________
____ Function to check if photos are saved correctly in SPIFFSl */
bool checkPhoto(fs::FS &fs) {
  File f_pic = fs.open(FILE_PHOTO);
  unsigned int pic_sz = f_pic.size();
  Serial.printf("File name: %s | size: %d\n", FILE_PHOTO, pic_sz);
  return (pic_sz > 100);
  f_pic.close();
}
/*
_____________________________________________________________________________
____ */

/*
_____________________________________________________________________________
____ Subroutine for formatting SPIFFS */
// This subroutine is used in case of failure to write or save the image
file to SPIFFS.
void SPIFFS_format() {
  bool formatted = SPIFFS.format();
  Serial.println();
```

```
  Serial.println("Format SPIFFS...");
  if(formatted){
    Serial.println("\n\nSuccess formatting");
  }else{
    Serial.println("\n\nError formatting");
  }
  Serial.println();
}
/*
_____ */

/*
_____ Subroutine for Capture Photo and Save it to SPIFFS */
void capturePhotoSaveSpiffs( void ) {
  camera_fb_t * fb = NULL; //--> pointer
  bool Status_save_photo = 0; //--> Boolean indicating if the picture has
been taken correctly

  /* ---------------------------------------- Take a photo with the camera
*/
  Serial.println();
  Serial.println("Taking a photo...");
  do {
    delay(2000);
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed.");
      Serial.println("Carry out the re-capture process...");
    }
  } while ( !fb );
  Serial.println("Take photo successfully.");
  /* ---------------------------------------- */

  /* ---------------------------------------- Save photos to SPIFFS */
  do {
    /* :::::::::::::::::::::::::::::::::::::::::::::::: Photo file name */
    Serial.printf("Picture file name: %s\n", FILE_PHOTO);
    File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);
    /* :::::::::::::::::::::::::::::::::::::::::::::::: */

    /* :::::::::::::::::::::::::::::::::::::::::::::::: Insert the data in
the photo file */
    if (!file) {
      Serial.println("Failed to open file in writing mode.");
      SPIFFS_format();
      capturePhotoSaveSpiffs();
      return;
    }
```

```
    else {
      file.write(fb->buf, fb->len); // payload (image), payload length
      Serial.print("The picture has been saved in ");
      Serial.print(FILE_PHOTO);
      Serial.print(" - Size: ");
      Serial.print(file.size());
      Serial.println(" bytes.");
    }
    /* :::::::::::::::::::::::::::::::::::::::::::::::: */

    file.close(); //--> Close the file

    /* :::::::::::::::::::::::::::::::::::::::::::::::: check if file has
been correctly saved in SPIFFS */
    Serial.println("Checking if the picture file has been saved correctly in
SPIFFS...");
    Status_save_photo = checkPhoto(SPIFFS);
    if (Status_save_photo == 1) {
      Serial.println("The picture file has been saved correctly in
SPIFFS.");
    } else {
      Serial.println("The picture file is not saved correctly in SPIFFS.");
      Serial.println("Carry out the re-save process...");
      Serial.println();
    }
    /* :::::::::::::::::::::::::::::::::::::::::::::::: */
  } while (!Status_save_photo);
  /* ------------------------------------- */

  esp_camera_fb_return(fb); //--> return the frame buffer back to the driver
for reuse.
}
/*
_____________________________________________________________________________
____ */

/*
_____________________________________________________________________________
____ Subroutine to get the Email sending status */
// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {
  Serial.println(msg.info()); //--> Print the current status
}
/*
_____________________________________________________________________________
____ */

/*
_____________________________________________________________________________
____ Subroutine for send photos via Email */
void sendPhoto( void ) {
```

```
  Serial.println("Sending email...");

  // Set the SMTP Server Email host, port, account and password
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderAppPassword);

  // Set the sender name and Email
  smtpData.setSender("ESP32-CAM UTEH STR PIR Sensor", emailSenderAccount);

  // Set Email priority or importance High, Normal, Low or 1 to 5 (1 is
highest)
  smtpData.setPriority("High");

  // Set the subject
  smtpData.setSubject(emailSubject);

  // Set the email message in HTML format
  smtpData.setMessage("<h2>Photo captured with ESP32-CAM and attached in
this email.</h2>", true);
  // Set the email message in text format
  //smtpData.setMessage("Photo captured with ESP32-CAM and attached in this
email.", false);

  // Add recipients, can add more than one recipient
  smtpData.addRecipient(emailRecipient);
  //smtpData.addRecipient(emailRecipient2);

  // Add attach files from SPIFFS
  smtpData.addAttachFile(FILE_PHOTO, "image/jpg");

  // Set the storage type to attach files in your email (SPIFFS)
  smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

  // sendCallback
  smtpData.setSendCallback(sendCallback);

  // Start sending Email, can be set callback function to track the status
  if (!MailClient.sendMail(smtpData))
  Serial.println("Error sending Email, " + MailClient.smtpErrorReason());

  // Clear all data from Email object to free memory
  smtpData.empty();

  // The LED Flash flashes 1 time with a duration per 1 second,
  // which means that the process of sending photos via email has been
completed (regardless of whether the photo was successfully sent or not).
  delay(2000);
}
/*
```

```
____ */

/*
_____________________________________________________________________________
____ VOID SETUP() */
void setup() {
  // put your setup code here, to run once:
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //--> disable brownout detector

  Serial.begin(115200);
  Serial.println();

  pinMode(pin_Pir, INPUT);

  /* ---------------------------------------- Loop to stabilize the PIR
sensor at first power on. */
  /*
   * I created this loop because from the tests I did that when the PIR
sensor starts to turn on,
   * the PIR sensor takes at least 30 seconds to be able to detect movement
or objects stably or with little noise.
   * I don't know if it's because of the quality factor of the PIR sensor I
have.
   * From this source:
https://lastminuteengineers.com/pir-sensor-arduino-tutorial/,
   * indeed the PIR sensor takes 30-60 seconds from the time it is turned on
to be able to detect objects or movements properly.
   */
  Serial.println("Wait 60 seconds for the PIR sensor to stabilize.");
  Serial.println("Count down :");
  for(int i = 29; i > -1; i--) {
    Serial.print(i);
    Serial.println(" second");
    delay(1000);
  }
  Serial.println("The time to stabilize the PIR sensor is complete.");
  Serial.println();
  /* ---------------------------------------- */

  /* ---------------------------------------- Connect to Wi-Fi */
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Successfully connected to ");
  Serial.println(ssid);
  /* ---------------------------------------- */
```

```
  /* ---------------------------------------- Print ESP32 Local IP Address
*/
  Serial.print("IP Address: http://");
  Serial.println(WiFi.localIP());
  Serial.println();
  /* ---------------------------------------- */

  /* ---------------------------------------- Starting to mount SPIFFS */
  Serial.println("Starting to mount SPIFFS...");
  if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    Serial.println("ESP32 Cam Restart...");
    ESP.restart();
  }
  else {
    Serial.println("SPIFFS mounted successfully");
  }
  /* ---------------------------------------- */

  /* ---------------------------------------- Camera configuration. */
  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_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA; //--> FRAMESIZE_ +
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    /*
     * From source
https://randomnerdtutorials.com/esp32-cam-ov2640-camera-settings/ :
     * - The image quality (jpeg_quality) can be a number between 0 and 63.
     * - Higher numbers mean lower quality.
```

```
     * - Lower numbers mean higher quality.
     * - Very low numbers for image quality, specially at higher resolution
can make the ESP32-CAM to crash or it may not be able to take the photos
properly.
     */
    config.jpeg_quality = 20;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
  /* ---------------------------------------- */

  /* ---------------------------------------- Initialize camera */
  Serial.println();
  Serial.println("Camera initialization...");
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    Serial.println("ESP32 Cam Restart...");
    ESP.restart();
  }
  Serial.print("Camera initialization was successful.");
  Serial.println();
  /* ---------------------------------------- */
}
/*
_____________________________________________________________________________
____ */

/*
_____________________________________________________________________________
____ VOID LOOP() */
void loop() {
  // put your main code here, to run repeatedly:

  // If the PIR sensor data = 1 means that objects and movements are
detected, it will be an indicator to start the image capture process and the
process of sending images via email.
  if(PIR_State() == 1) {
    capturePhotoSaveSpiffs(); //--> Calling the capturePhotoSaveSpiffs()
subroutine.
    sendPhoto(); //--> Calling the sendPhoto() subroutine.
  }
  delay(1);
}
/*
_____________________________________________________________________________
____ */
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
<<<<<<<<<<<<<<<<<<<<<<
```

3.1 Explanation of code

Import the libraires

A library is a collection of pre-written code and functions that provide additional functionality to the Arduino sketches.
ESP32_MailClient.h allows ESP32 to send emails with attachments via SMTP servers (Simple Mail Transfer Protocol), here is the procedure to install the ESP-Mail-Client library in Arduino.
WiFi.h allows ESP32 to connect to the local WiFi network which is useful for projects needed to connect Arduiono to the internet to send and receive data.
<SPIFFS.h> is used to access and manage the SPIFFS(SPI Flash File System)on ESP32 board. SPIFFS is a lightweight file system that uses the flash memory of the board to store and retrieve files.
FS.h allows to read, write, and manage files on different types of storage, such as SPIFFS, SD cards, and internal EEPROM.

```
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>
```

Define email

Define the email account and password that the ESP32-CAM would login as well as the recipient's email
Define the email provider SMTP setting, the SMTP settings for different email providers are different.

```
#define emailSenderAccount      "amc491561@gmail.com@gmail.com"
#define emailSenderAppPassword  "vsvnciqerihrhurx"
#define smtpServer              "smtp.gmail.com"
#define smtpServerPort          465
#define emailSubject            "ESP32-CAM Photo Captured"
#define emailRecipient          "YOUR_EMAIL_RECIPIENT@gmail.com"
```

Insert the WiFi name and password

```
// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "iotlab";
const char* password = "iotlab18";
```

The taken photo will be temporarily saved in a flash memory SPIFFS under the name 'photo.jpg'. Define the PIR sensor OUT pin connect to GPIO12

```
#define FILE_PHOTO "/photo.jpg" //--> Photo File Name to save in SPIFFS

#define pin_Pir 12 //--> PIR Motion Detector PIN
```

Pin definition of ESP32-CAM AI-Thinker

Each ESP32 Camera development board uses different GPIOs to connect to the camera.

```
/* ======================================== Defining the Camera Model and
GPIO */
#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1
  #define XCLK_GPIO_NUM      0
  #define SIOD_GPIO_NUM     26
  #define SIOC_GPIO_NUM     27

  #define Y9_GPIO_NUM       35
  #define Y8_GPIO_NUM       34
  #define Y7_GPIO_NUM       39
  #define Y6_GPIO_NUM       36
  #define Y5_GPIO_NUM       21
  #define Y4_GPIO_NUM       19
  #define Y3_GPIO_NUM       18
  #define Y2_GPIO_NUM        5
  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22
```

'capturePhotoSaveSpiffs()' function

The function captures an image using the camera and saves it to the SPIFFS. If any error occurs during the process, it attempts to recover and reattempt the capture and save operation.

'sendPhoto()' function

The function configures the email settings, attaches the captured photo, sends the email using the configured SMTP server, and then clears the data from the 'smtpData' object

'setup()' function

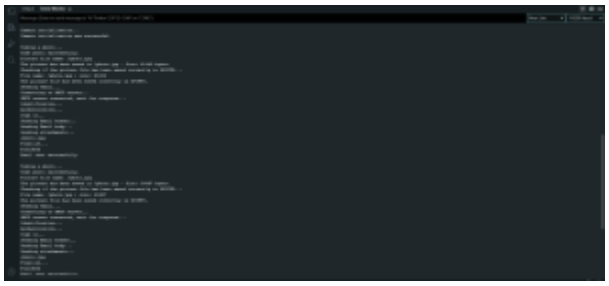
The function is responsible for setting up the ESP32-CAM by disabling the brownout detector meaning it avoids ESP32 resetting in case of small voltage fluctuations.
Set the PIR sensor pin as an input pin.
The PIR sensor needs some time to settle and calibrate after power-up before it can accurately detect movements. Therefore here it counts down for 30 seconds before connecting to WiFi.
Define the camera configuration setting to determine the resolution, JPEG quality, and frame buffer count.

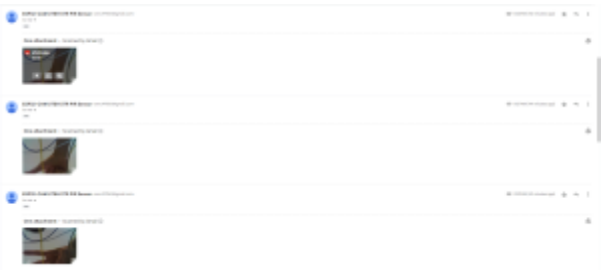
'loop()' function

If the PIR sensor data = 1, means that motions are detected, then the camera starts taking photos and saving to SPIFFS. After that, it sends images via email

4.Results and Discussion

4.1 Real-Time Clock(RTC)





Photos are captured and sent successfully. However, it needs some improvement. The first problem is shown in the result photo, the file name of each photo sent by ESP32 are the same, which is 'photo.jpg'. Although the time can be noticed from the time of the email, it is not accurate when there is a sudden WiFi connection loss and connect in 30 minutes later. Also it is more convenient when people want to download the photos in the computer with the photo capture time as the file name. Therefore, the ESP32 Real-Time Clock(RTC) is used below to get the current time and use it as part of the file name.

A rough modification of the code is here

First is to add the RTC libraries

```
#include "time.h"
```

modify 'capturePhotoSaveSpiffs()' function

```
void capturePhotoSaveSpiffs(void) {
  camera_fb_t *fb = NULL;
```

```
  bool statusSavePhoto = false;

  Serial.println();
  Serial.println("Taking a photo...");

  do {
    delay(2000);
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed.");
      Serial.println("Retrying...");
    }
  } while (!fb);

  Serial.println("Photo captured successfully.");

  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    esp_camera_fb_return(fb);
    return;
  }

  char time_str[30];
  sprintf(time_str, "%04d-%02d-%02d_%02d-%02d-%02d", timeinfo.tm_year +
1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
          timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);

  String photoFileName = "/photo_" + String(time_str) + ".jpg";

  File file = SPIFFS.open(photoFileName, FILE_WRITE);

  if (!file) {
    Serial.println("Failed to open file in writing mode.");
    SPIFFS_format();
    esp_camera_fb_return(fb);
    capturePhotoSaveSpiffs();
    return;
  } else {
    file.write(fb->buf, fb->len);
    Serial.print("The picture has been saved as ");
    Serial.print(photoFileName);
    Serial.print(" - Size: ");
    Serial.print(file.size());
    Serial.println(" bytes.");
  }

  file.close();

  Serial.println("Checking if the picture file has been saved correctly in
SPIFFS...");
```

```
  statusSavePhoto = checkPhoto(SPIFFS);
  if (statusSavePhoto) {
    Serial.println("The picture file has been saved correctly in SPIFFS.");
    FILE_PHOTO = photoFileName; // Set the new file name to the global
variable FILE_PHOTO
  } else {
    Serial.println("The picture file is not saved correctly in SPIFFS.");
    Serial.println("Retrying...");
    esp_camera_fb_return(fb);
    capturePhotoSaveSpiffs();
    return;
  }

  esp_camera_fb_return(fb);
}
```

modify 'setup()' function

```
void setup() {
  // put your setup code here, to run once:
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //--> disable brownout detector

  Serial.begin(115200);
  Serial.println();

  pinMode(pin_Pir, INPUT);

  /* ------------------------------------- Loop to stabilize the PIR
sensor at first power on. */
  /*
   * I created this loop because from the tests I did that when the PIR
sensor starts to turn on,
   * the PIR sensor takes at least 30 seconds to be able to detect movement
or objects stably or with little noise.
   * I don't know if it's because of the quality factor of the PIR sensor I
have.
   * From this source:
https://lastminuteengineers.com/pir-sensor-arduino-tutorial/,
   * indeed the PIR sensor takes 30-60 seconds from the time it is turned on
to be able to detect objects or movements properly.
   */
  Serial.println("Wait 60 seconds for the PIR sensor to stabilize.");
  Serial.println("Count down :");
  for(int i = 29; i > -1; i--) {
    Serial.print(i);
    Serial.println(" second");
    delay(1000);
  }
  Serial.println("The time to stabilize the PIR sensor is complete.");
  Serial.println();
```

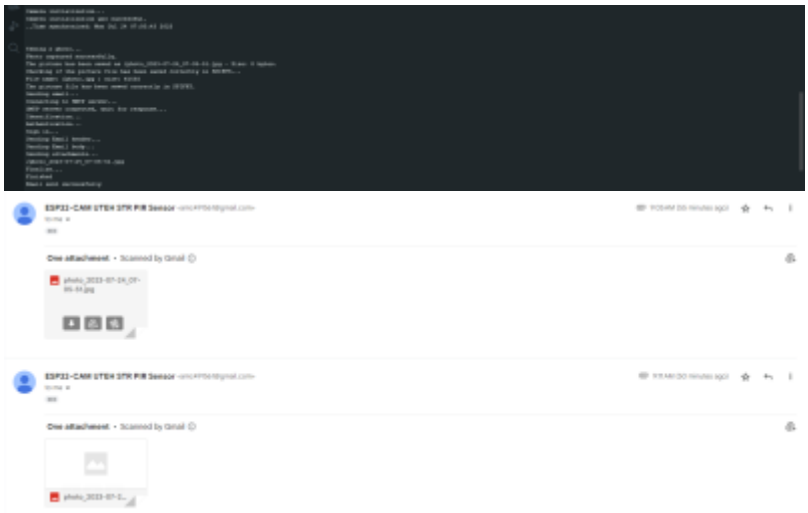
```
  /* ---------------------------------------- */

  /* ---------------------------------------- Connect to Wi-Fi */
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Successfully connected to ");
  Serial.println(ssid);
  /* ---------------------------------------- */

  /* ---------------------------------------- Print ESP32 Local IP Address
*/
  Serial.print("IP Address: http://");
  Serial.println(WiFi.localIP());
  Serial.println();
  /* ---------------------------------------- */

  /* ---------------------------------------- Starting to mount SPIFFS */
  Serial.println("Starting to mount SPIFFS...");
  if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    Serial.println("ESP32 Cam Restart...");
    ESP.restart();
  }
  else {
    Serial.println("SPIFFS mounted successfully");
  }
  /* ---------------------------------------- */

  /* ---------------------------------------- Camera configuration. */
  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_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
```

```
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA; //--> FRAMESIZE_ +
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    /*
     * From source
https://randomnerdtutorials.com/esp32-cam-ov2640-camera-settings/ :
     * - The image quality (jpeg_quality) can be a number between 0 and 63.
     * - Higher numbers mean lower quality.
     * - Lower numbers mean higher quality.
     * - Very low numbers for image quality, specially at higher resolution
can make the ESP32-CAM to crash or it may not be able to take the photos
properly.
     */
    config.jpeg_quality = 20;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
  /* ---------------------------------------- */

  /* ---------------------------------------- Initialize camera */
  Serial.println();
  Serial.println("Camera initialization...");
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    Serial.println("ESP32 Cam Restart...");
    ESP.restart();
  }
  Serial.print("Camera initialization was successful.");
  Serial.println();
  /* ---------------------------------------- */
  // Initialize RTC
  time_t now;
  configTime(0, 0, "pool.ntp.org");
  while (true) {
    time(&now);
    struct tm *timeinfo = localtime(&now);
    if (timeinfo->tm_year >= (2023 - 1900)) {
      break;
    }
    Serial.print(".");
    delay(1000);
```

```
  }
  Serial.println("Time synchronized: " + String(ctime(&now)));
}
```

Finally change the declaration

```
String FILE_PHOTO = "/photo.jpg";
```



As shown in the result photo, the name of the photo is now changed to the format “photo_YYYY-MM-DD_HH-MM-SS.jpg” according to the time of capture.

While the code is not sufficiently developed, it only modifies the file name, some more improvement on the code has to be done.

4.2 Deep Sleep Mode

We put our ESP32-Cam in Deep Sleep mode in order to reduce the power consumption. So that the battery will last longer. In deep sleep mode, CPU and WIFi activities are cut off, but the Ultra Low Power(ULP) co-processor is still powered on. Some of the ESP32 pins can be used by the ULP co-processor during deep sleep.
The original concept of this project is using GPIO12 as an external wake up pin, which when the pir sensor detects motion, it sends a digital output signal high (logical 1) to the wake up pin to trigger esp32-cam to wake up.
While finally we found that it is not suitable to do it in this project. The reason is that when the ESP32 is in deep sleep mode, it effectively “shuts down” the processor, and the code execution stops. It does not maintain the previous state or continue running the code where it left off. Instead, it will restart the entire program from the beginning and start execution from the ’setup()’ function again, as if you pressed the reset button. This means it counts down to stabilize, connect to WiFi, mount SPIFFS, initialize the camera and take photos every time when it detects motion. Which will not take photos efficiently.
While without using the deep sleep mode, it counts down to stabilize, connect to WiFi, mount SPIFFS and initialize the camera once, and take photos whenever it detects motion.

5.Reference