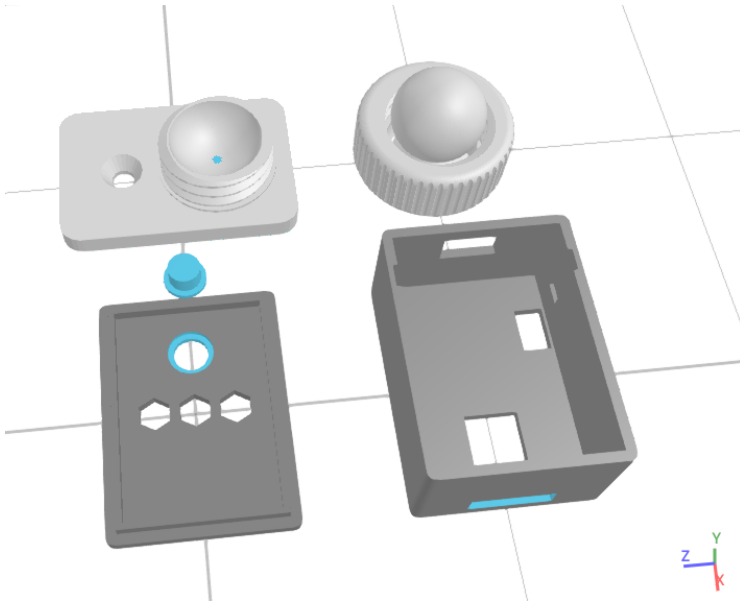


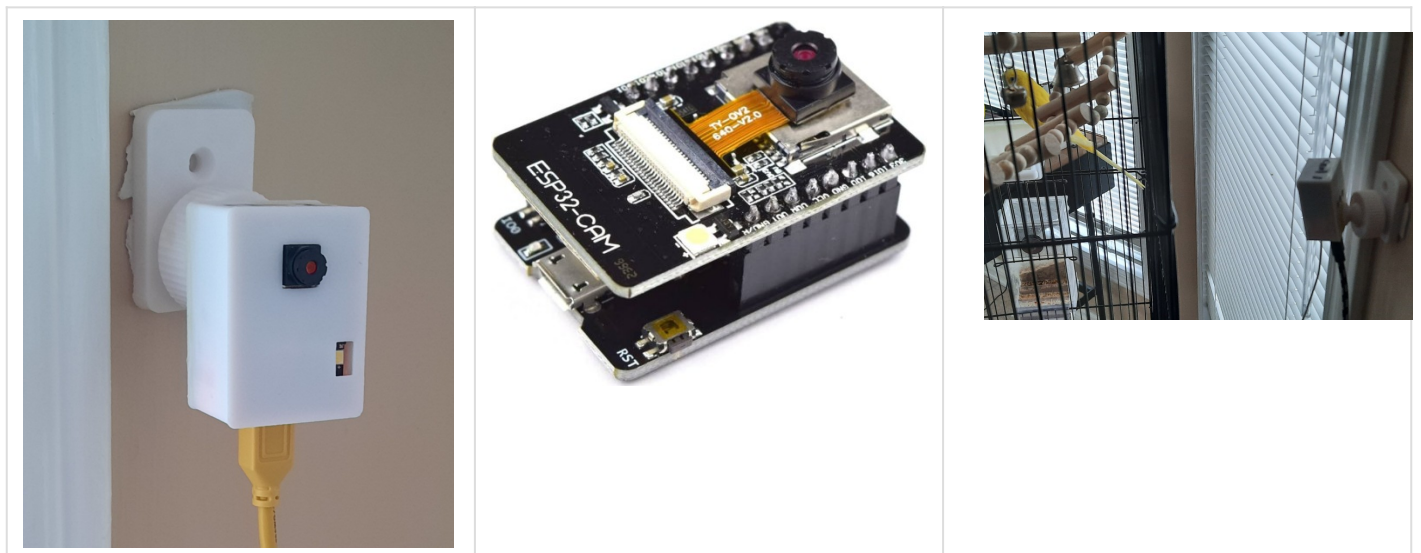
Simple Surveillance Camera

This probably sounds dumb but I wanted to make sure my birds were not starving to death anytime I needed to travel. The solution was 2 very inexpensive (less than \$10) ESP32 CAM modules. These are pretty much plug and play regarding hardware. I just needed to print a case and write some code (code at bottom of this page), and the images are sent to my server in 5 minute intervals. For night time there is a flash built into the ESP32 CAM which actually works well, even in larger areas.

Here is the 3d printer file for the case: [esp32_case.zip](#)



The printed case assembled:



And here is what I see on my webpage:



So there you have it - the ultimate cheap-ass bird cam!

In case you want to do this yourself here's the code I used:

```
#include <WiFi.h>
#include <EEPROM.h>
#include <esp_sleep.h>
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"
#include "esp_camera.h"

int activeConnection = 1;

const String ssid1 = "xxxx";
const String password1 = "xxxx";
```

```

// silo
const String ssid2 = "xxxxx";
const String password2 = "xxxxxx";

String source = "camera1";

String serverName = "xx.xxx.ca";
String serverPath = "/xxxx/setphoto.php?source=" + source;

const int serverSendInterval = 5; // 5 minutes between sending a photo update to the server

const int serverPort = 80;

#define LED_BUILTIN 4
#define EEPROM_SIZE 4
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;
long secondsOfHour = 0;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool serverFailed = false;
int wifiConnectionAttempts = 0;

WiFiClient client;

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.grab_mode = CAMERA_GRAB_LATEST;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = 3;

pinMode(LED_BUILTIN, OUTPUT);

// camera init
esp_err_t err = esp_camera_init( & config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
}

//Init EEPROM
EEPROM.begin(EEPROM_SIZE);

activeConnection = EEPROM.read(eepromActiveConnection);

```

```
Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
    activeConnection = 1;
}

if (activeConnection == 0) {
    activeConnection = 1;
}

if (activeConnection > 2) {
    activeConnection = 2;
}

Serial.print("activeConnection: ");
Serial.println(activeConnection);

float pingData = EEPROM.readFloat(eepromPingsAddress);
if (isnan(pingData)) {
    pingData = 0;
}
totalServerPings = pingData;
EEPROM.end();
EEPROM.begin(EEPROM_SIZE);
float pingFailData = EEPROM.readFloat(eepromFailedPingsAddress);
if (isnan(pingFailData)) {
    pingFailData = 0;
}
totalServerPingFails = pingFailData;
EEPROM.end();

if (!connectToWiFi()) {
    delay(2000);
    WiFi.disconnect();
    delay(1000);
    if (activeConnection == 1) {
        activeConnection = 2;
    } else {
        activeConnection = 1;
    }
}
```

```
    connectToWiFi();
}

}

void loop() {

    delay(serverSendInterval * 60 * 1000);

    secondsOfHour += serverSendInterval * 60;

    // after 24 hours reset this integer
    if (secondsOfHour > 86400) {
        secondsOfHour = 1;
        ESP.restart();
    }

    sendPhoto();

}

void sendPhoto() {

    String getAll;
    String getBody;

    camera_fb_t * fb = NULL;
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500); // allow time for led to fully illuminate
    fb = esp_camera_fb_get();
    delay(50);
    digitalWrite(LED_BUILTIN, LOW);

    if (!fb) {
        Serial.println("Camera capture failed");
        delay(1000);
        ESP.restart();
    }

    setWifiSleepMode(false);
```

```
delay(5000);
```

```
Serial.println("Connecting to server: " + serverName);
```

```
if (client.connect(serverName.c_str(), serverPort)) {
```

```
    Serial.println("Connection successful!");
```

```
    String head = "--ImpresstoDocs\r\nContent-Disposition: form-data; name=\"imageFile\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
```

```
    String tail = "\r\n--ImpresstoDocs--\r\n";
```

```
    uint32_t imageLen = fb -> len;
```

```
    uint32_t extraLen = head.length() + tail.length();
```

```
    uint32_t totalLen = imageLen + extraLen;
```

```
    client.println("POST " + serverPath + " HTTP/1.1");
```

```
    client.println("Host: " + serverName);
```

```
    client.println("Content-Length: " + String(totalLen));
```

```
    client.println("Content-Type: multipart/form-data; boundary=ImpresstoDocs");
```

```
    client.println();
```

```
    client.print(head);
```

```
    uint8_t * fbBuf = fb -> buf;
```

```
    size_t fbLen = fb -> len;
```

```
    for (size_t n = 0; n < fbLen; n = n + 1024) {
```

```
        if (n + 1024 < fbLen) {
```

```
            client.write(fbBuf, 1024);
```

```
            fbBuf += 1024;
```

```
        } else if (fbLen % 1024 > 0) {
```

```
            size_t remainder = fbLen % 1024;
```

```
            client.write(fbBuf, remainder);
```

```
        }
```

```
    }
```

```
    client.print(tail);
```

```
    esp_camera_fb_return(fb);
```

```
    int timeoutTimer = 10000;
```

```
    long startTimer = millis();
```

```
    boolean state = false;
```

```

while ((startTimer + timeoutTimer) > millis()) {
  Serial.print(".");
  delay(100);
  while (client.available()) {
    char c = client.read();
    if (c == '\n') {
      if (getAll.length() == 0) {
        state = true;
      }
      getAll = "";
    } else if (c != '\r') {
      getAll += String(c);
    }
    if (state == true) {
      getBody += String(c);
    }
    startTimer = millis();
  }
  if (getBody.length() > 0) {
    break;
  }
}
Serial.println();
client.stop();
Serial.println(getBody);

setWifiSleepMode(true);
} else {
  getBody = "Connection to " + serverName + " failed.";
  Serial.println(getBody);
  client.stop();

  setWifiSleepMode(true);

}
}

bool connectToWiFi() {

```



```
String activeSsid = "";
String activePassword = "";

if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
} else if (activeConnection == 2) {
    activeSsid = ssid2;
    activePassword = password2;
}

Serial.print("Connecting to WiFi: ");
Serial.println(activeSsid);

WiFi.begin(activeSsid, activePassword);

while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
    delay(500);
    Serial.print(".");
    wifiConnectionAttempts++;
}

wifiConnectionAttempts = 0;

if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
    wifiConnected = true;

    EEPROM.begin(EEPROM_SIZE);
    EEPROM.write(eepromActiveConnection, activeConnection);
    EEPROM.commit();
    EEPROM.end();

    Serial.print("set activeConnection to : ");
    Serial.println(activeConnection);

    return true;
```

```

    } else {
        Serial.print("Connection to ");
        Serial.print(activeSsid);
        Serial.println(" failed. Trying alternative");
        return false;
    }
}

/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

    wifiSleeping = sleepMode;

    if (sleepMode) {
        WiFi.disconnect();
        WiFi.setSleep(true);
        delay(1000);
        Serial.print("sleep wifi status: ");
        Serial.println(wl_status_to_string(WiFi.status()));
    } else {
        WiFi.setSleep(false);
        WiFi.reconnect();
        delay(2000);
        Serial.print("awaken wifi status: ");
        Serial.println(wl_status_to_string(WiFi.status()));
        // Check if the connection is still active. if not trigger wait for it to come back online
        if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
            Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
            WiFi.disconnect();
            wifiPaused = true;
            wifiConnected = false;
            connectToWiFi();
        }
    }
}

/**

```

```

* record server ping success in long term memory
*/
void recordPingSuccess() {
    totalServerPings++;
    EEPROM.begin(EEPROM_SIZE);
    EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
    EEPROM.commit();
    EEPROM.end();
    wifiConnected = true;
    serverFailed = false;
}

/**
* record server ping fails in long term memory
*/
void recordPingFailure() {
    totalServerPingFails++;
    EEPROM.begin(EEPROM_SIZE);
    EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
    EEPROM.commit();
    EEPROM.end();
    wifiConnected = false;
    serverFailed = true;
}

/**
* ESP32 wifi card statuses
* @param status
* @return string
*/
String wl_status_to_string(wl_status_t status) {

    String response = "";

    switch (status) {
    case WL_NO_SHIELD:
        response = "WL_NO_SHIELD";
        break;
    case WL_IDLE_STATUS:
        response = "WL_IDLE_STATUS";

```

```
    break;
case WL_NO_SSID_AVAIL:
    response = "WL_NO_SSID_AVAIL";
    break;
case WL_SCAN_COMPLETED:
    response = "WL_SCAN_COMPLETED";
    break;
case WL_CONNECTED:
    response = "WL_CONNECTED";
    break;
case WL_CONNECT_FAILED:
    response = "WL_CONNECT_FAILED";
    break;
case WL_CONNECTION_LOST:
    response = "WL_CONNECTION_LOST";
    break;
case WL_DISCONNECTED:
    response = "WL_DISCONNECTED";
    break;
}

return response;
}
```

Revision #10

Created 5 March 2024 05:22:32 by peter drinnan

Updated 7 March 2024 12:00:42 by peter drinnan