Smart Solutions (LOTI.05.060) Project - Video Surveilance

Roberts Oskars Komarovskis

May 2021

Overview

The goal of this project is to create a project that will allow to look at the room remotely. The main components of the project are - ESP32-CAM module, Arduino (any kind), servo motor, and a computer. General diagram of the elements and connections for the project can be seen in Figure 1. ESP32-CAM acts as a server, which uploads video feed in local network. This video feed can be opened through the browser or it can be accessed through python with opencv package. The computer end has a single python script that displays the video stream, reads the pressed keys, and sends characters to the ESP32-CAM module based on the pressed key. ESP32-CAM module reads the characters sent from computer and either sets one of the two GPIOs high. If the key is released, both outputs are low. Arduino reads the digital GPIOs and based on the value turns the servo motor.

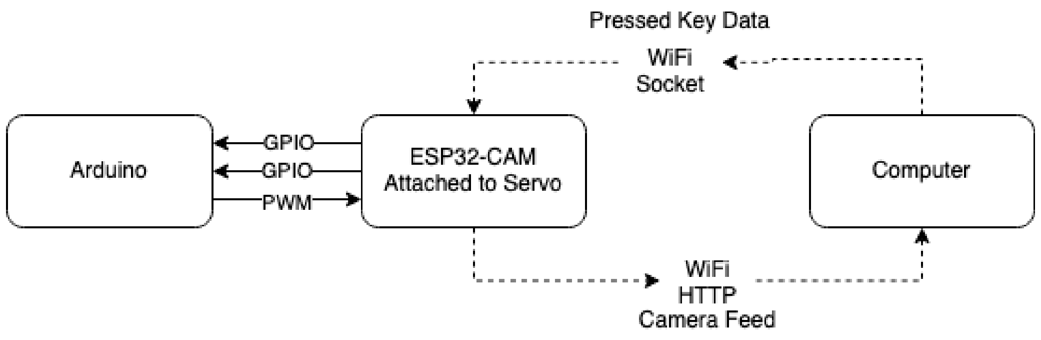
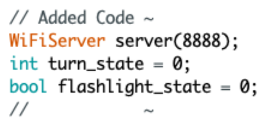


Figure 1: Block diagram of the remote surveillance project

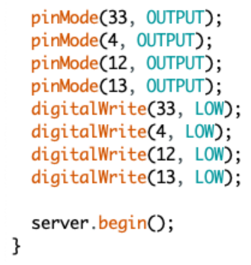
ESP32-CAM Module

ESP32-CAM configuration is the main part of the project. FTDI programmer was used to upload the code. Tutorial was followed to configure both Arduino IDE and the module itself. [1] This tutorial provides detailed description of ESP32-CAM pinout, connections to the programmer, device setup in the Arduino IDE, and other information. Source files were taken from the examples in the Arduino IDE (this is also covered in the tutorial). This source code was slightly modified to communicate with the computer and the Arduino. Following images provides description of the changes that were done.



```
// Added Code ~
WiFiServer server(8888);
int turn_state = 0;
bool flashlight_state = 0;
//            ~
```

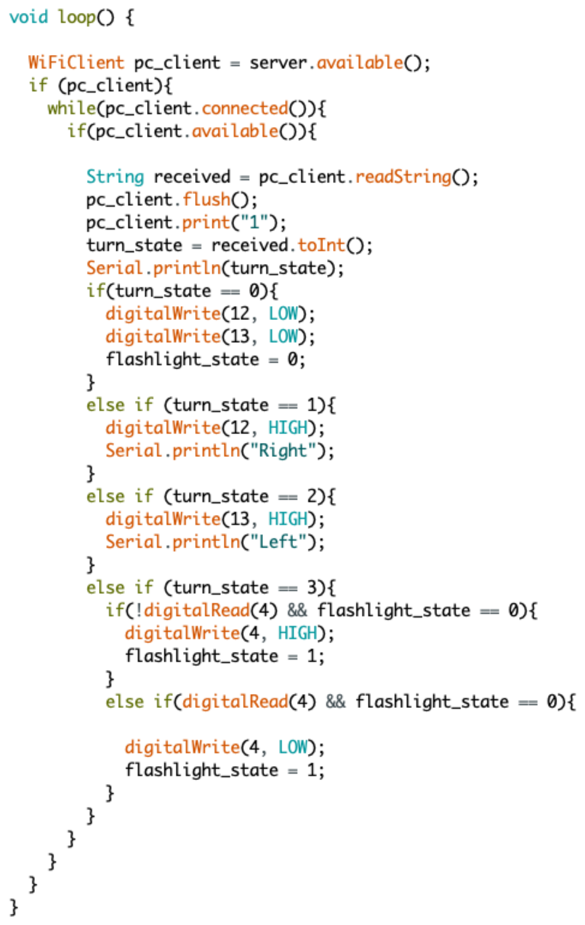
Figure 2: These are the initial definition lines before the "void setup()" in the "CameraWebServer.ino" file



```
  pinMode(33, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(33, LOW);
  digitalWrite(4, LOW);
  digitalWrite(12, LOW);
  digitalWrite(13, LOW);

  server.begin();
}
```

Figure 3: Added code in the "void setup()" in the "CameraWebServer.ino" file to configure the GPIOs of the ESP32, and also start the wifi server for receiving messages from the computer



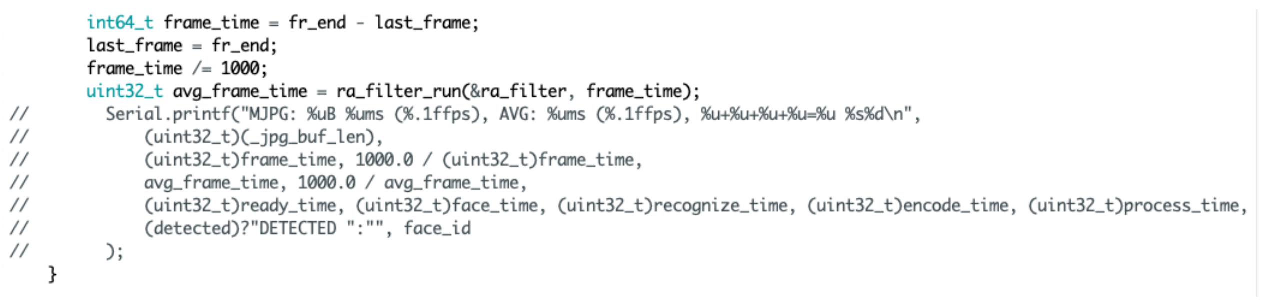
```
void loop() {

  WiFiClient pc_client = server.available();
  if (pc_client){
    while(pc_client.connected()){
      if(pc_client.available()){

        String received = pc_client.readString();
        pc_client.flush();
        pc_client.print("1");
        turn_state = received.toInt();
        Serial.println(turn_state);
        if(turn_state == 0){
          digitalWrite(12, LOW);
          digitalWrite(13, LOW);
          flashlight_state = 0;
        }
        else if (turn_state == 1){
          digitalWrite(12, HIGH);
          Serial.println("Right");
        }
        else if (turn_state == 2){
          digitalWrite(13, HIGH);
          Serial.println("Left");
        }
        else if (turn_state == 3){
          if(!digitalRead(4) && flashlight_state == 0){
            digitalWrite(4, HIGH);
            flashlight_state = 1;
          }
          else if(digitalRead(4) && flashlight_state == 0){

            digitalWrite(4, LOW);
            flashlight_state = 1;
          }
        }
      }
    }
  }
}
```

Figure 4: Main loop of the "CameraWebServer.ino" file. It reads data from the computer and sends backs acknowledge character. GPIO state are set based on the received data.

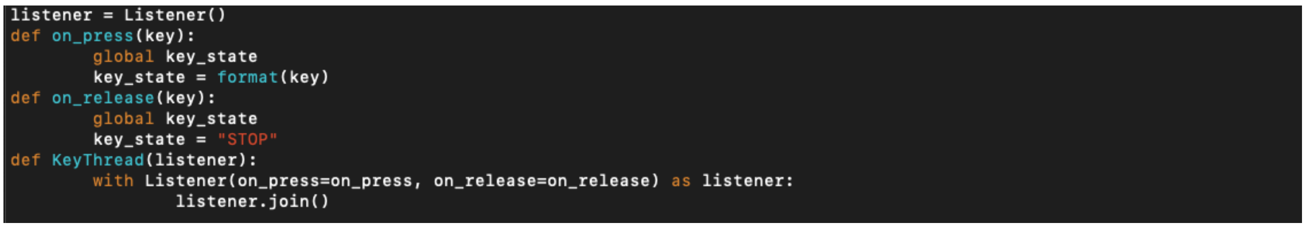


```
        int64_t frame_time = fr_end - last_frame;
        last_frame = fr_end;
        frame_time /= 1000;
        uint32_t avg_frame_time = ra_filter_run(&ra_filter, frame_time);
//        Serial.printf("MJPG: %uB %ums (%.1ffps), AVG: %ums (%.1ffps), %u+%u+%u+%u=%u %s%d\n",
//            (uint32_t)(_jpg_buf_len),
//            (uint32_t)frame_time, 1000.0 / (uint32_t)frame_time,
//            avg_frame_time, 1000.0 / avg_frame_time,
//            (uint32_t)ready_time, (uint32_t)face_time, (uint32_t)recognize_time, (uint32_t)encode_time, (uint32_t)process_time,
//            (detected)?"DETECTED ":"", face_id
//        );
    }
```

Figure 5: These lines was commented out in the "app_httpd.cpp" file to avoid populating the Serial monitor

Computer

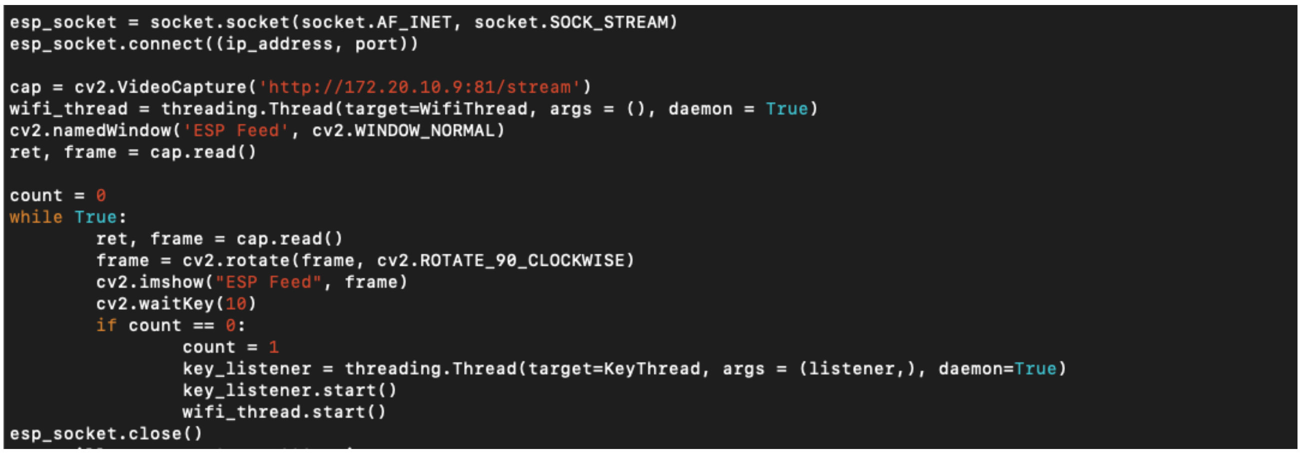
Computer code is done with python3 language. Main functions of the script is to look at the camera feed, and record the key presses and send data to the ESP32-CAM module based on the pressed key. Key listener code, which can be seen in Figure 6. was taken from online source. [2] If nothing is pressed or if some key is released, "0" is sent to the ESP32-CAM. If right key is pressed "1" is sent to the EPS-32CAM. If left key pressed, "2" is sent to the ESP-32CAM. After each sent message, program waits for response.



```
listener = Listener()
def on_press(key):
        global key_state
        key_state = format(key)
def on_release(key):
        global key_state
        key_state = "STOP"
def KeyThread(listener):
        with Listener(on_press=on_press, on_release=on_release) as listener:
                listener.join()
```

Figure 6: Code for defining key listening functions and variables

Main code of the computer script can be seen in figure 7. which starts the socket and connects to it. Cap defines the source for the video feed. Wifi thread is started which will conduct the sending and receiving characters from the ESP32-CAM. The first loop iteration defines and starts the wifi thread, and also starts the key listener. This has to be done after the cv2.waitKey() as the listener impacts its functionality.



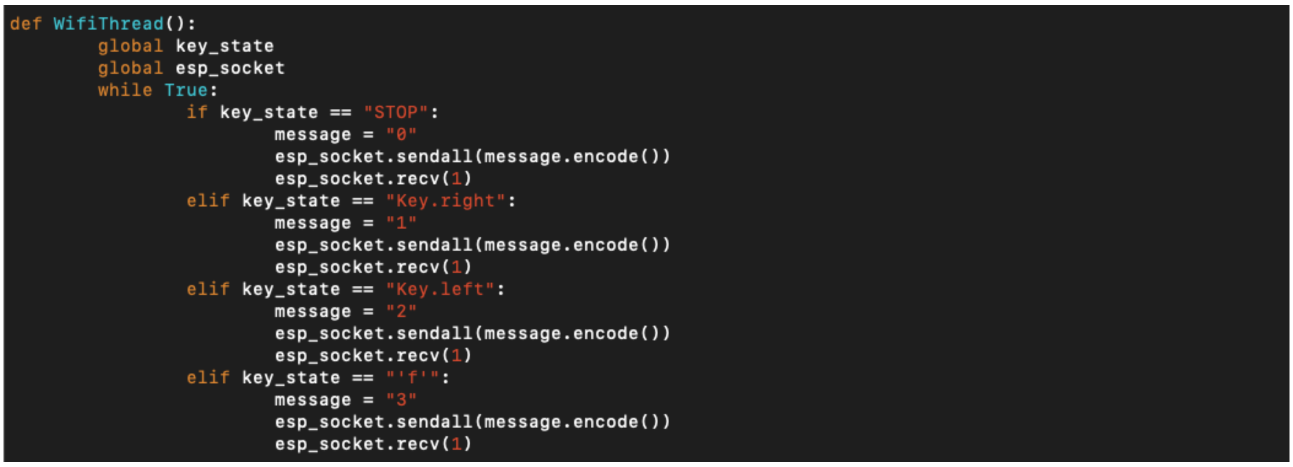
```
esp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
esp_socket.connect((ip_address, port))

cap = cv2.VideoCapture('http://172.20.10.9:81/stream')
wifi_thread = threading.Thread(target=WifiThread, args = (), daemon = True)
cv2.namedWindow('ESP Feed', cv2.WINDOW_NORMAL)
ret, frame = cap.read()

count = 0
while True:
        ret, frame = cap.read()
        frame = cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)
        cv2.imshow("ESP Feed", frame)
        cv2.waitKey(10)
        if count == 0:
                count = 1
                key_listener = threading.Thread(target=KeyThread, args = (listener,), daemon=True)
                key_listener.start()
                wifi_thread.start()
esp_socket.close()
```

Figure 7: Creation of different important variables and main loop which reads the video feed

Figure 8. contains the thread functions which deals with communication between the EPS32-CAM. The function looks at the global variable "key_state", which is updated by the key listener. Based on its value, different characters are sent to the ESP32-CAM.

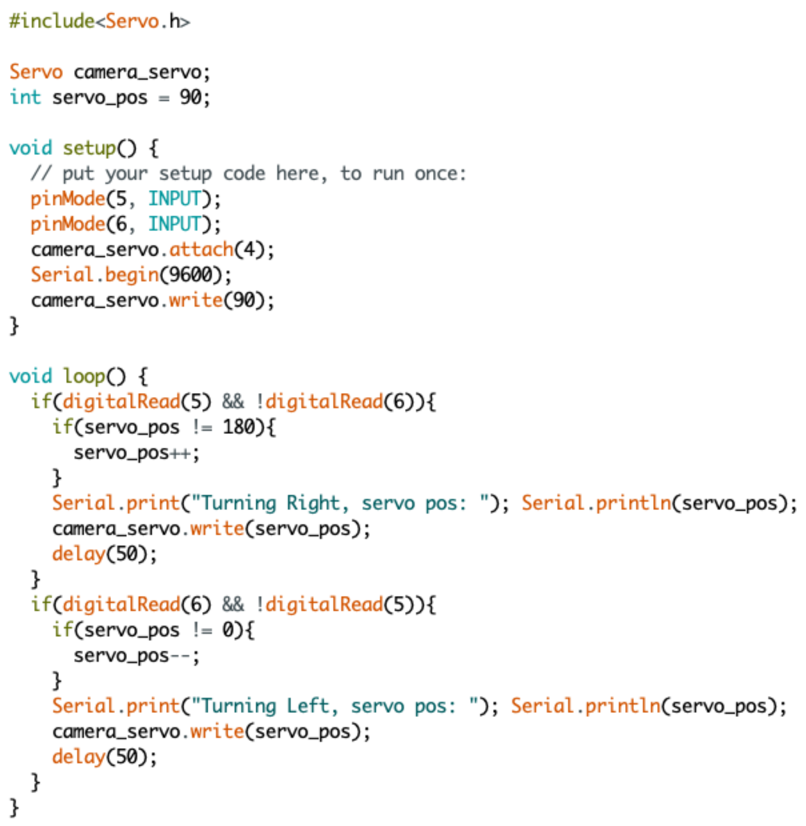


```
def WifiThread():
        global key_state
        global esp_socket
        while True:
                if key_state == "STOP":
                        message = "0"
                        esp_socket.sendall(message.encode())
                        esp_socket.recv(1)
                elif key_state == "Key.right":
                        message = "1"
                        esp_socket.sendall(message.encode())
                        esp_socket.recv(1)
                elif key_state == "Key.left":
                        message = "2"
                        esp_socket.sendall(message.encode())
                        esp_socket.recv(1)
                elif key_state == "'f'":
                        message = "3"
                        esp_socket.sendall(message.encode())
                        esp_socket.recv(1)
```

Figure 8: Wifi thread which deals with communicating with the EPS32-CAM module

Arduino

Arduino code is fairly simple. Servo.h library was used to control the servo motor. The arduino reads the digital values of the ESP32-CAM general outputs. If one of the outputs is high, turn servo to the right. If the other one is high, turn to the left. If both are low, don't turn. The Arduino code can be seen in Figure 9.



```
#include<Servo.h>

Servo camera_servo;
int servo_pos = 90;

void setup() {
  // put your setup code here, to run once:
  pinMode(5, INPUT);
  pinMode(6, INPUT);
  camera_servo.attach(4);
  Serial.begin(9600);
  camera_servo.write(90);
}

void loop() {
  if(digitalRead(5) && !digitalRead(6)){
    if(servo_pos != 180){
      servo_pos++;
    }
    Serial.print("Turning Right, servo pos: "); Serial.println(servo_pos);
    camera_servo.write(servo_pos);
    delay(50);
  }
  if(digitalRead(6) && !digitalRead(5)){
    if(servo_pos != 0){
      servo_pos--;
    }
    Serial.print("Turning Left, servo pos: "); Serial.println(servo_pos);
    camera_servo.write(servo_pos);
    delay(50);
  }
}
```

Figure 9: Arduino code for controlling the servo

Electrical characteristics of ESP32-CAM

ESP32 operates at 3.3V. The module has a LDO, which can convert 5V to 3.3V, so 5V can be supplied by the 5V pin. There is also a 3.3V pin, which can be used to supply 3.3V directly. If Ardunio pins are used as input (high-impedance), then level converters are not necessary. However, if other functions are implemented between the Arduino and the ESP32-CAM then the level converter has to be used. ESP32-CAM also has an on-board flashlight, which can be controlled with the IO4. Schematics can be seen in Figure 10., which were taken from an online source. [3]

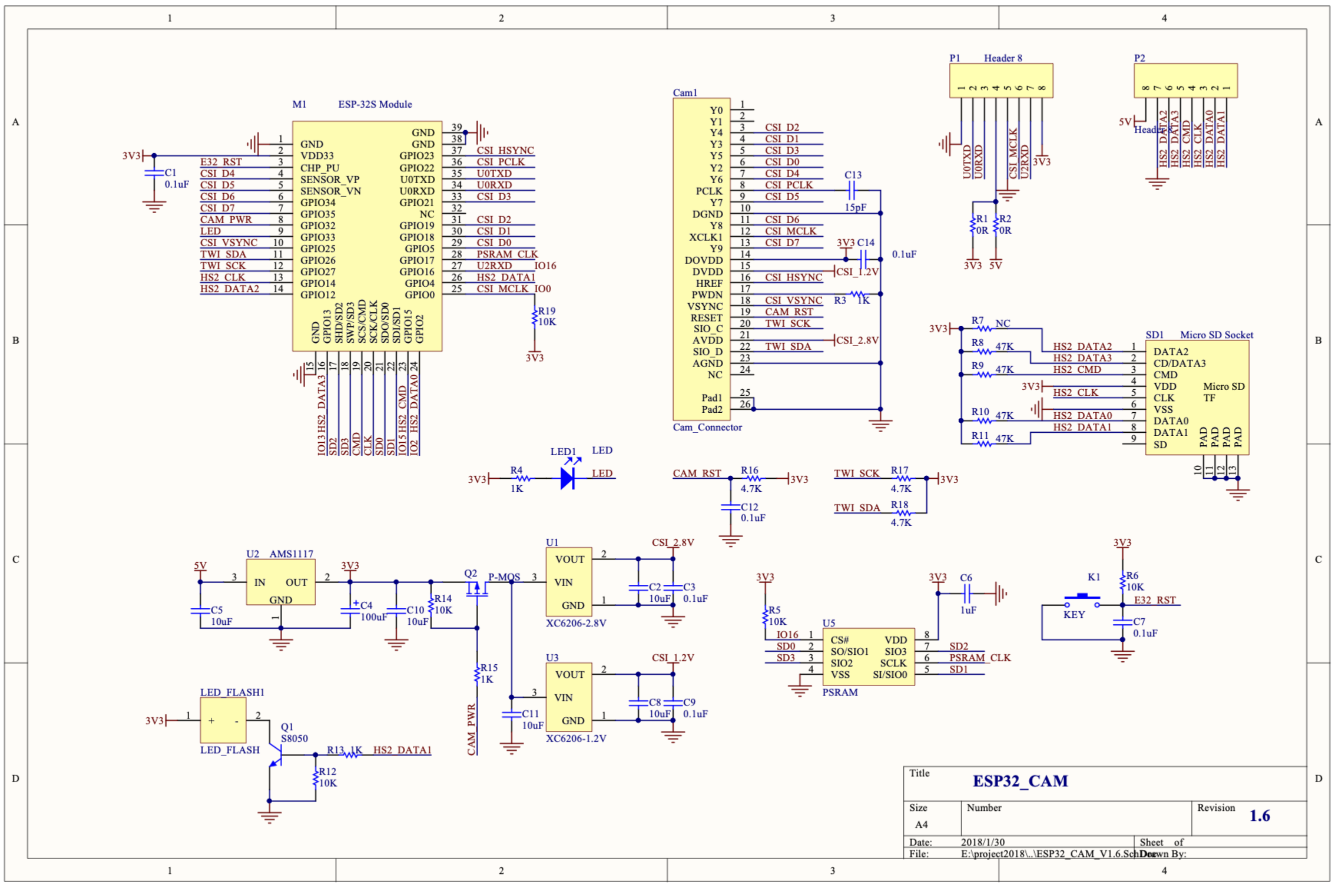


Figure 10: Schematics of the ESP32-CAM module

References

[1] , "ESP32-CAM Video Streaming and Face Recognition with Arduino IDE." https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/.

[2] , "How to Detect Key Presses In Python." https://nitratine.net/blog/post/how-to-detect-key-presses-in-python/, 2020.

[3] , "Schematics of the ESP32-CAM module." https://raw.githubusercontent.com/SeeedDocument/