INTERNATIONAL JOURNAL OF RESEARCH CULTURE SOCIETY ISSN(O): 2456-6683
Monthly Peer-Reviewed, Refereed, Indexed Journal [Impact Factor: 9.241]
Volume - 8, Issue - 10, October - 2024

DOIs:10.2017/IJRCS/202410007 --:-- Research Paper / Article / Review

Wi-Fi based UAV transceiver: frame transmitter and channel data receiver

Soham Chatterjee [1], Sanjita Moyra [2], Abhijit Banerjee [3]
[1,2] M.Sc. Department of Computer Science, Institute of Management Study, EM Bypass, 93 Mukundapur, Kolkata-700099, Maulana Abul Kalam Azad University of Technology
[3] State Aided College Teacher, Department of Computer Science, New Alipore College, Block-L, New Alipore, Kolkata-700053, University of Calcutta

Abstract: *Increasing advancement in the concept of wireless network and imaging technology is shaping the world by its global application domain. Network providing intercontinental communication, Wi-Fi that wirelessly connects multiple devices or imaging technology that is even used in surveillance. At present embedded wireless network (WLAN) controller have capability of high bandwidth trans-reception over long distances and embedded imaging module having real-time operating firmware operates in emergencies. Existing producers of radio controllers have medium to long range transmission distance for RC bots, drones or submarines. FPV technology is highly in demand at present ruling the RC category. By thorough study of the existing concepts and all the domain related applications the devised idea is to use a compact, less bulky device. Finally, the capability of receiving control instruction and transmit frames & it`s status real-time to an android application suitable for every PWM controlled applications. With this study there is a scope for optimization of technology in terms of hardware, functional requirement and portability.*

Key Words: *RCA, ESP32-S,* navigationServer, *MIT, android.*

1. INTRODUCTION:

The earliest record of remotely controlled aircraft (RCA) can be traced back to world war I where A.M. Low contributed to early television and radio technology and was later used to develop British RCA during 1917-1918 to attack German aircraft [1]. Later Geoffrey de Havilland`s designed monoplane only flew under control on March 21st 1917 [2].

The first RCA having surveillance camera was developed and tested by Israeli intelligence during war in the middle east between 1967-1970 [3]. Being deployed at Suez Canal, it returned photos successfully and this RCA could launch and land easily. During Yom Kippur War of 1973 Egypt and Syria damaged Israeli fighter jets heavily by using Soviet surface to air missiles which is why Israel developed IAI Scout having real-time surveillance system [4][5][6].

It provided Israel support to retaliate on Syrian air defences and start of Lebanon war of 1982 [7].
UN Security Council`s panel of experts on Libya reported that a RCA Kargu 2 surveilled and strike down a target in Libya in the year 2020 in a military operation marks the beginning of armed RCA [8][9]. Later that year Turks developed Bayraktar TB2 which led Azerbaijan to success in war of Nagorno Karabakh against Armenia [10].

As of 2013 and afterwards RCA is also used in public domain for completion of different task and human aid which includes photography, surveillance, crowd monitoring, crop survey, search and rescue, environment monitoring, accident investigation and civil industry [11][12].

Parallelly technology like Webcam and IP Camera came up in 1991[13][14] and 1996[15][16] respectively. Since then, lots of improved technology came to existence. One such technology is ESP32-CAM which combines the ESP32 MCU and OV2640 camera was introduced in 2019. It gained a lot of recognition for its potential in aerial imaging, remote monitoring, IoT and drone technology [17][18].

Innovative applications of ESP32CAM emerged as developers utilized it for live video streaming from drones allowing real-time surveillance and monitoring, applications related to agriculture, environmental assessment, security and search & rescue. By combining with GPS modules, it helped in autonomous navigation and quality image capture. With support of advance software, it holds potential for obstacle avoidance and AI-based image processing [19][20][21].

In the current work, after broad study of various radio & IP cameras, radio & Wi-Fi transceivers we updated our Wi-Fi camera and receiver unit that was built in the previous work [22]. The updates in current design reduces the complexities that was present in the transceiver unit of the former work to achieve more modularity, portability and simplicity in design. Present unified system can now interact singularly i.e. frames will now be transmitted to smart cell-phone and control channel data will be sent from smart cell-phone to receiver directly.

2. METHODOLOGY:

A. ESP Cam & Controller Unit:

ESP32 developed by Espressif Technologies has major area of application in the current technological world. The receiver module is developed over the rare Ai Thinker ESP32-S CAM have Wi-Fi and BT on board for wireless applications. Currently used ESP Cam shown in Fig.1 model has 2 Mega Pixels camera along with the breakout board which is used for streaming frames real-time over the Wi-Fi connectivity. The flight can commence autonomously in the background and actions are directed by designed embedded software running on ESP32. Actions taken by ESP are:

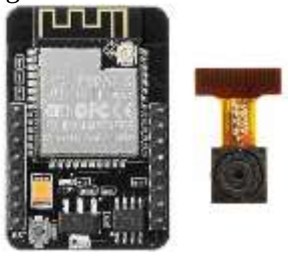


Fig.1

I.Receive Instruction (from Cell-phone Android Application):

This process begins with setting up of an Access Point (AP) of any particular IP Address as set by the programmer for e.g. 192.168.4.1. Then the stream and control server are set-up on port number 81 and 80 respectively. Instructions are of two types:

a. Navigation Control instruction:

Originally designed as an UAV control receiver unit, it includes 7 parameters given in Fig.2 and each one has its own function. The navigation is handled by navigation uri at ESP32 on port 80.

Channel Number	Channel Name	Channel Type	Operating Range
Channel 1	Throttle	PWM	1000µs - 2000µs
Channel 2	Yaw	PWM	1000µs - 2000µs
Channel 3	Pitch	PWM	1000µs - 2000µs
Channel 4	Roll	PWM	1000µs - 2000µs
Channel 5	Auxiliary 1	Digital Switch	3.3V / 0V
Channel 6	Auxiliary 2	Digital Switch	3.3V / 0V
Channel 7	Visor LED	Digital Switch	3.3V / 0V

Fig.2

Summing up, the receiver unit has 4 PWM Channels, 2 Auxiliary Switches and an extra Switching Channel for head lamp LED control at the hardware level.

b. Stream Control commands:

Stream control is handled by the stream uri. The stream server begins after set-up is completed on port 81 and streaming can only be started after the start stream http request from the client is received. Streaming is stopped by the stop stream http request from the client connected.

Other stream related commands are Resolution, Alignment (Horizontal & Vertical), Quality, Brightness, Contrast, Saturation and Special Effects which is directly controlled by the control server on port 80.

II.Send Frames & Status (to Cell-phone Android Application):

a. **Streaming** of frames is handled by the stream handler. Start or stop request is made on control server, then request is acknowledged. Frame is constructed in the buffer and is transported as response on the webserver handled by the stream handler on port 81.

The webserver is the part where the frames are sent to in the form of image stream html. Pixel Format, jpg buffer, buffer length, response are some of the notable elements of streaming.

b. Status regarding resolution, alignment, quality, brightness, contrast, saturation and special effects can be sent back to the client as acknowledgement. This acknowledgment is generated by the device based on above parameters and feedback response is exchanged during command instruction by cmd uri on port 80.

III.Flow of work:

The process showed in Fig.3 and Fig.4 where the setup and configuration of a Esp32 camera server is done. The first step is to define the libraries which is shown below. The pins for the throttle, yaw, pitch, roll, LED, ADC, AUX1, AUX2 and camera GPIOs are configured simultaneously. In the next step the IP address settings for the local IP, gateway and subnet are set along with the Wi-Fi Service Set Identifier & password. The two functions “startCameraServer” and “navigationServer” are declared including pin configurations and also the network connectivity.
List of libraries included: **a.** esp_camera.h, **b.** WiFi.h
List of Defined Pins:

ID	No.	ID	No.	ID	No.
THROTTLE_PIN	12	RESET_GPIO_NUM	-1	Y5_GPIO_NUM	21
YAW_PIN	13	XCLK_GPIO_NUM	0	Y4_GPIO_NUM	19
PITCH_PIN	15	SIOD_GPIO_NUM	26	Y3_GPIO_NUM	18
ROLL_PIN	14	SIOC_GPIO_NUM	27	Y2_GPIO_NUM	5
LED_PIN	4	Y9_GPIO_NUM	35	VSYNC_GPIO_NUM	25
AUX1_PIN	0	Y8_GPIO_NUM	34	HREF_GPIO_NUM	23
AUX2_PIN	16	Y7_GPIO_NUM	39	PCLK_GPIO_NUM	22
PWDN_GPIO_NUM	32	Y6_GPIO_NUM	36		

Fig.3

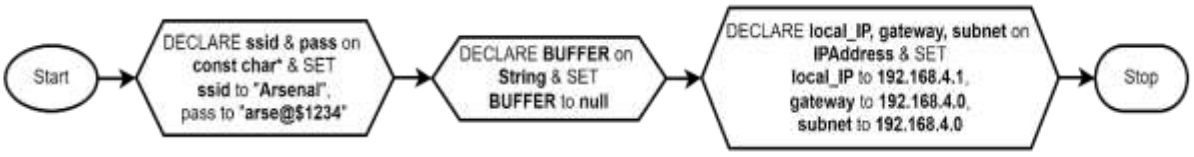

Start
DECLARE ssid & pass on const char* & SET ssid to "Arsenal", pass to "arse@$1234"
DECLARE BUFFER on String & SET BUFFER to null
DECLARE local_IP, gateway, subnet on IPAddress & SET local_IP to 192.168.4.1, gateway to 192.168.4.0, subnet to 192.168.4.0
Stop

Fig.4

The process in Fig.5 the setup and configuration of a Esp32 camera server is done with initializing the setup function. In the next step the GPIO pins for the camera initialize successfully followed by checking if there is no error. In the next step it sets the camera's frame size and configure the camera & control server for streaming.

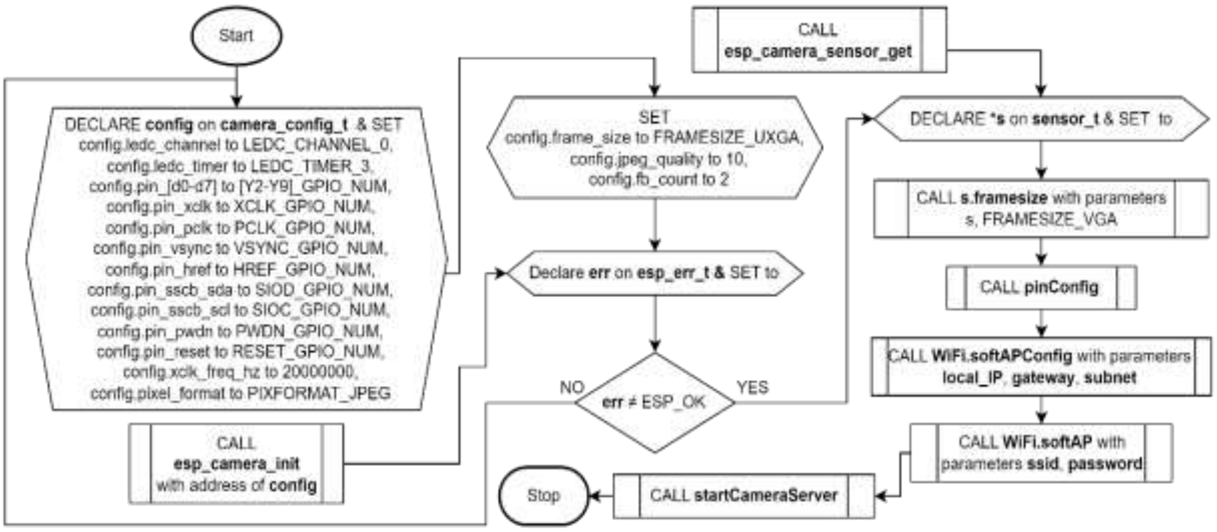
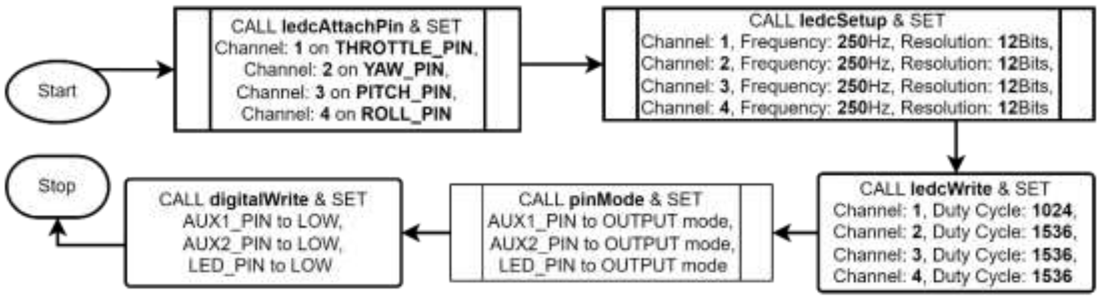
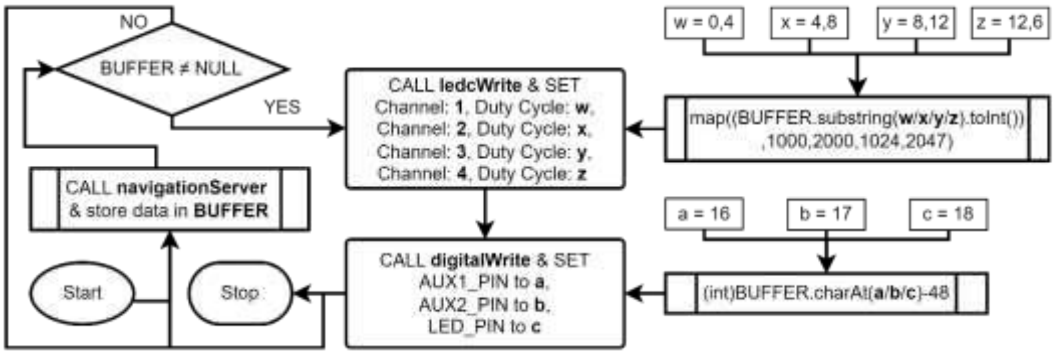

Start
DECLARE config on camera_config_t & SET config.ledc_channel to LEDC_CHANNEL_0, config.ledc_timer to LEDC_TIMER_3, config.pin_[d0-d7] to [Y2-Y9]_GPIO_NUM, config.pin_xclk to XCLK_GPIO_NUM, config.pin_pclk to PCLK_GPIO_NUM, config.pin_vsync to VSYNC_GPIO_NUM, config.pin_href to HREF_GPIO_NUM, config.pin_sscb_sda to SIOD_GPIO_NUM, config.pin_sscb_scl to SIOC_GPIO_NUM, config.pin_pwdn to PWDN_GPIO_NUM, config.pin_reset to RESET_GPIO_NUM, config.xclk_freq_hz to 20000000, config.pixel_format to PIXFORMAT_JPEG
CALL esp_camera_init with address of config
SET config.frame_size to FRAMESIZE_UXGA, config.jpeg_quality to 10, config.fb_count to 2
Declare err on esp_err_t & SET to
err ≠ ESP_OK
NO
YES
CALL esp_camera_sensor_get
DECLARE *s on sensor_t & SET to
CALL s.framesize with parameters s, FRAMESIZE_VGA
CALL pinConfig
CALL WiFi.softAPConfig with parameters local_IP, gateway, subnet
CALL WiFi.softAP with parameters ssid, password
CALL startCameraServer
Stop

Fig.5

When the “pinConfig” function begins in Fig.6, defines the throttle, yaw, pitch, and roll pins to PWM channels from 1 to 4 using “ledcAttachPin”. Then it sets up these PWM channels with a frequency of 250 Hz and a resolution of 12 bits using “ledcSetup”. The Initial PWM values are written with “ledcWrite” to these channels. The function then setup auxiliary pins which are AUX1_PIN and AUX2_PIN and an LED pin as outputs using “pinMode” and ensure their initial states to LOW using “digitalWrite”. After it defines the ADC pin as an input.


Start
CALL ledcAttachPin & SET Channel: 1 on THROTTLE_PIN, Channel: 2 on YAW_PIN, Channel: 3 on PITCH_PIN, Channel: 4 on ROLL_PIN
CALL ledcSetup & SET Channel: 1, Frequency: 250Hz, Resolution: 12Bits, Channel: 2, Frequency: 250Hz, Resolution: 12Bits, Channel: 3, Frequency: 250Hz, Resolution: 12Bits, Channel: 4, Frequency: 250Hz, Resolution: 12Bits
CALL ledcWrite & SET Channel: 1, Duty Cycle: 1024, Channel: 2, Duty Cycle: 1536, Channel: 3, Duty Cycle: 1536, Channel: 4, Duty Cycle: 1536
CALL pinMode & SET AUX1_PIN to OUTPUT mode, AUX2_PIN to OUTPUT mode, LED_PIN to OUTPUT mode
CALL digitalWrite & SET AUX1_PIN to LOW, AUX2_PIN to LOW, LED_PIN to LOW
Stop
Fig.6

In "loop" function in Fig.7, "navigationServer" function is called to load navigation data saving the result in the "BUFFER" variable. If "BUFFER" is not null, it reads "BUFFER" data to control the outputs. By using "BUFFER" substrings the "ledcWrite" is set PWM values for four channels. It also configures the states of three digital pins (LED_PIN, AUX1_PIN, and AUX2_PIN). After that it prints the throttle, yaw, pitch, roll, LED, AUX1 and AUX2 readings to the serial monitor for debugging.


NO
BUFFER ≠ NULL
YES
CALL navigationServer & store data in BUFFER
Start
Stop
w = 0,4
x = 4,8
y = 8,12
z = 12,6
map((BUFFER.substring(w/x/y/z).toInt()) ,1000,2000,1024,2047)
CALL ledcWrite & SET Channel: 1, Duty Cycle: w, Channel: 2, Duty Cycle: x, Channel: 3, Duty Cycle: y, Channel: 4, Duty Cycle: z
a = 16
b = 17
c = 18
(int)BUFFER.charAt(a/b/c)-48
CALL digitalWrite & SET AUX1_PIN to a, AUX2_PIN to b, LED_PIN to c
Fig.7

B. Transmitter Unit:

The android application-based Wi-Fi transceiver unit was developed on MIT App Inventor a block-based visual programming platform which is based on Java. Current android application is named Arsenal having version 1.0 after previous application UAV HUB 2.0.

I. Tools and extensions:

a. **Interface tools**: Label, Button, Slider, Switch & TextBox
b. **Layout:** HorizontalArrangement, VerticalArrangement & VerticalScrollArrangement
c. **Drawing & Animation:** Canvas, Ball
d. **Connectivity:** Web
e. **Sensors:** Clock
f. **Extension:** LayoutShadow, TaifunTools, CustomWebView, BrowsePromptHelper & DownloadHelper

II.Initialization:

a. The interface of the android application has two screens. The application starts executing "Screen1" which is the loading screen of the application as shown in Fig.8.

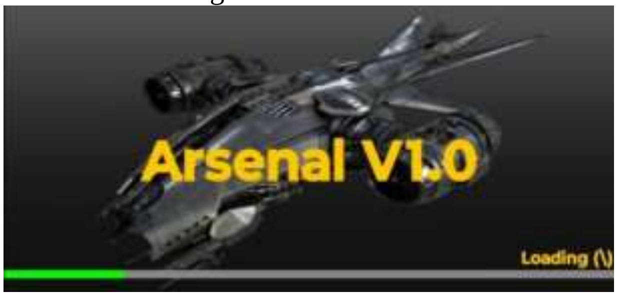


Fig. 8

The process in the Fig.9 begins by initializing variables like "cycle" and "screen clock". Subroutines like "HideStyleUI" and screen "orientation" are called to conceal stylistic elements and ensure correct visual display. An event listener is added to detect and respond to errors during setup.

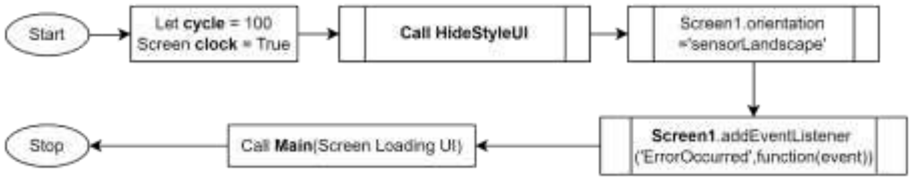


Fig. 9

The procedure in the main function includes an error-checking routine. It starts with definition of function called "errorOccurred" with parameters like "errorNumber" and message. It checks if "errorNumber" is "1101" or "1103". If no errors are detected, the function proceeds to the next phase of the application, transitioning to "Screen2" as shown in the flowchart Fig.10.

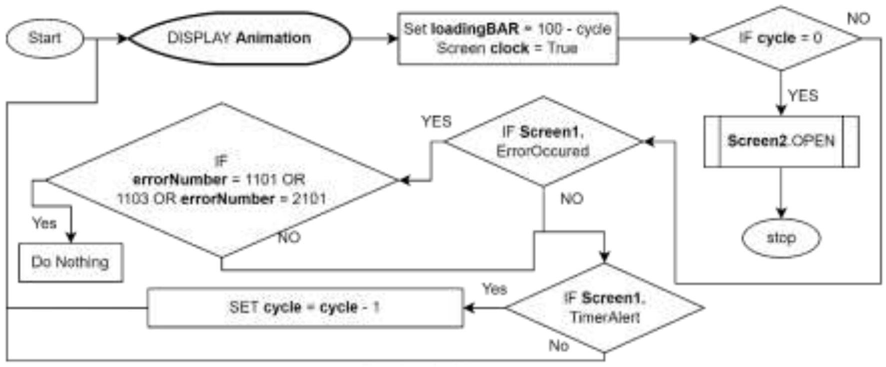


Fig. 10

b. "Screen2" loads as soon as "Screen1" has finished its execution. "Screen2" in Fig.11 contains all the features regarding controller and streaming.

- In the extreme left is the Joystick for controlling Throttle and Yaw across X and Y axis respectively and the extreme right is the Joystick for controlling Pitch and Roll across X and Y axis respectively.
- In the middle there are two separate sections, one for Auxiliary switches (AUX1, AUX2 & V LED) and other for Application control (Settings, QUIT & Refresh).
- On the very top is the screen label and then Navigation Status bar having Throttle, Yaw, Pitch & Roll and Auxiliary Status bar having AUX1, AUX2 & V LED showing the real-time data.

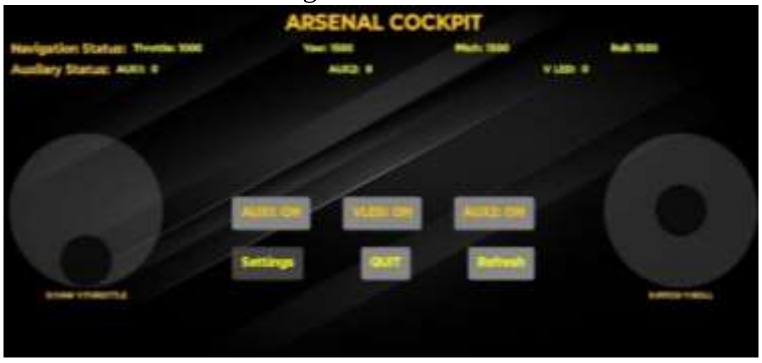


Fig. 11

The process in the Fig.12 starts by initializing:

IPA_COM = NULL: This variable is initialized to NULL, which indicates that no communication protocol has been established at the start.
Joystick1_COM = 0: The communication variable for Joystick1 is set to 0, a default state for the joystick input.
Joystick2_CenterX = 60: The centre X-coordinate for Joystick2 is set to 60, establishing a baseline for positioning.
Joystick2_CenterY = 60: Similarly, the Y-coordinate for Joystick2 is initialized to 60.
ThrottleStick1_RememberY = 120: This variable is set to 120, to recall the last Y-coordinate of ThrottleStick1.
yawMap = 180: The yaw mapping value is initialized to 180.
rollMap = 0: This variable is set to 0, indicating no initial roll or tilt.
var = NULL & **val = 0**.
Joystick1_CenterX = 60: The centre X-coordinate for Joystick1 is also set to 60, ensuring that both joysticks begin with similar initialization values.

ISSN(O): 2456-6683
[Impact Factor: 9.241]

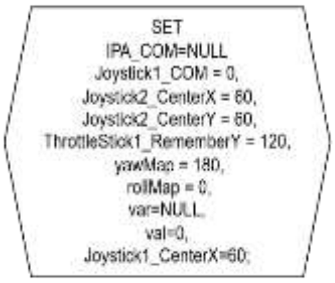


Fig.12

The setup procedure as shown in the Fig.13 with System Initialization with steps which include calling “HideSystemUI”, setting the screen orientation to “SensorLandscape”, selecting the initial colour scheme for the interface, positioning the joystick and configuring control parameters for navigation.

The process then sets up for web streaming where the first WebView container is initialized with ID 1. The ESP_WIFI_CAM function initializes a Wi-Fi camera, setting the stage for live streaming within the application. The ESP Stream_Layer is activated to handle data flow from the camera and another WebView container is initialized with ID 2.

The ESP Stream_Layer Setting configures key parameters of the stream layer such as shadow angle, distance and radius. The ESP_Navigation_Layer is for navigation, managing user interaction and movement through the application. Setting StreamLayerSettings, ShadowAngle = 0, ShadowDistance = 0, ShadowRadius = 0 defines the properties of the streaming layer for controlling shadow effects in the video stream.

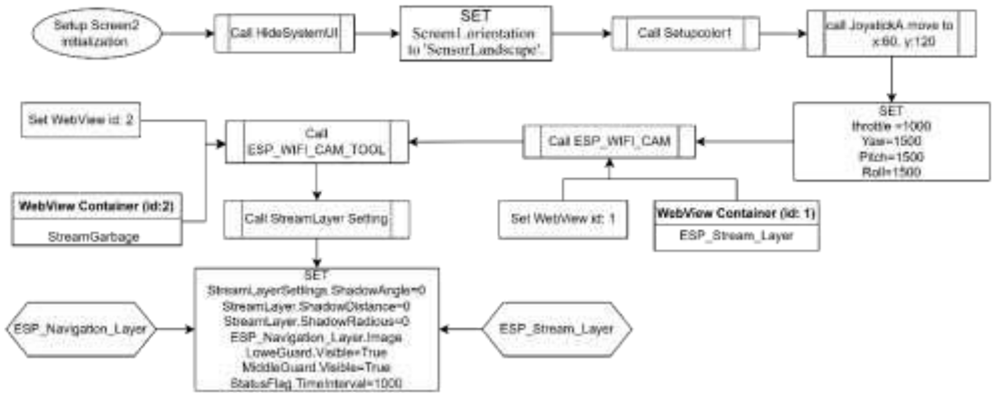

Setup Screen2 initialization
Call HideSystemUI
SET Screen1.orientation to 'SensorLandscape'
Call Setupcolor1
call JoystickA.move to x:60, y:120
SET throttle =1000 Yaw=1500 Pitch=1500 Roll=1500
Call ESP_WIFI_CAM
Call ESP_WIFI_CAM_TOOL
Set WebView id: 2
WebView Container (id:2)
StreamGarbage
Call StreamLayer Setting
Set WebView id: 1
WebView Container (id: 1)
ESP_Stream_Layer
SET StreamLayerSettings.ShadowAngle=0 StreamLayer.ShadowDistance=0 StreamLayer.ShadowRadious=0 ESP_Navigation_Layer.Image LoweGuard.Visible=True MiddleGuard.Visible=True StatusFlag.TimeInterval=1000
ESP_Navigation_Layer
ESP_Stream_Layer

Fig.13

III.Application model functions:

a. The JoyValue result is calculated by retrieving current and map coordinates and an offset value. The final step performs a calculation which involves subtracting “mapXY” from “CurrXY” and multiplying the result by 6, taking the absolute value, adding 1000, subtracting the offset, and rounding the final value as shown in Fig.14.

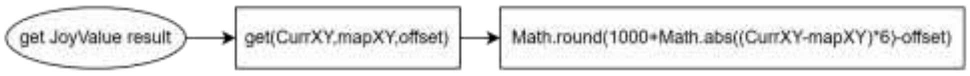


Fig.14

b. “navTool” controls input axes for movement. The throttle check ensures that the throttle input does not exceed a specified upper limit. If true, the system sets the throttle to 2000. If false, the process sets the throttle value to 1000. Same checking is done for yaw, pitch and roll as shown in the Fig.15.

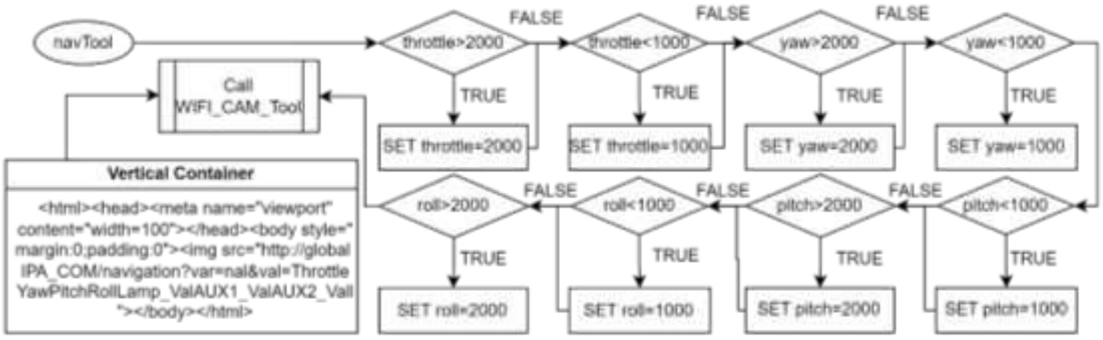


Fig.15

c. The process in Fig.16 shows the JoystickA dragged function where it checks the if current X>180, if it's true then it sets the value of current X to 180. Similarly, the checking is done for current Y axis. In the next step the value of current X and Y is used calculate the Throttle and Yaw value when the JoystickA dragged to a position and then it sets the current Y value to the previous position where its dragged. The process concludes with a call to the "navTool". Similar process is done in Fig.17 for JoystickB where it sets the value of X and Y and move the current X and Y value to calculate the position for Roll and Pitch. The process concludes with a call to the "navTool".

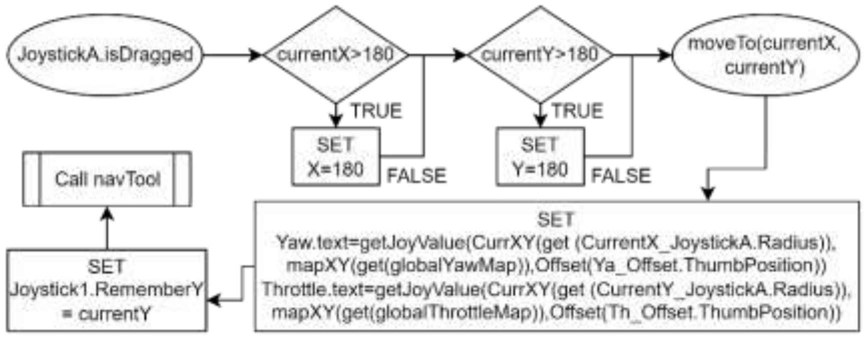

JoystickA.isDragged
currentX>180
currentY>180
moveTo(currentX, currentY)
TRUE
TRUE
SET X=180
FALSE
SET Y=180
FALSE
Call navTool
SET Joystick1.RememberY = currentY
SET Yaw.text=getJoyValue(CurrXY(get (CurrentX_JoystickA.Radius)), mapXY(get(globalYawMap)),Offset(Ya_Offset.ThumbPosition)) Throttle.text=getJoyValue(CurrXY(get (CurrentY_JoystickA.Radius)), mapXY(get(globalThrottleMap)),Offset(Th_Offset.ThumbPosition))

Fig.16

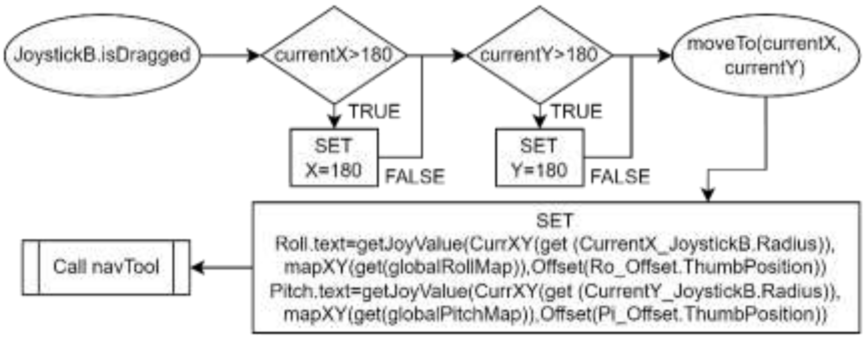

JoystickB.isDragged
currentX>180
currentY>180
moveTo(currentX, currentY)
TRUE
TRUE
SET X=180
FALSE
SET Y=180
FALSE
Call navTool
SET Roll.text=getJoyValue(CurrXY(get (CurrentX_JoystickB.Radius)), mapXY(get(globalRollMap)),Offset(Ro_Offset.ThumbPosition)) Pitch.text=getJoyValue(CurrXY(get (CurrentY_JoystickB.Radius)), mapXY(get(globalPitchMap)),Offset(Pi_Offset.ThumbPosition))

Fig.17

d. The process shows the management of joystick inputs by initializing variables and conditional control based on user input. It begins when the joystick is released or touched up. Two important variables are initialized: var JoystickX = Joystick.CenterX and var JoystickY = Joystick.getRememberY() - Joystick.Radius(). The system then evaluates conditions based on the joystick's position. If the conditions are met the system updates control variables by adjusting yaw dynamically based on the thumb position. The throttle is set using function calls and variables. Throttle = get(joyValue) to determines the throttle level. If the conditions are not met, the process proceeds to the next step. The process concludes with a call to the "navTool". As shown in Fig.18.

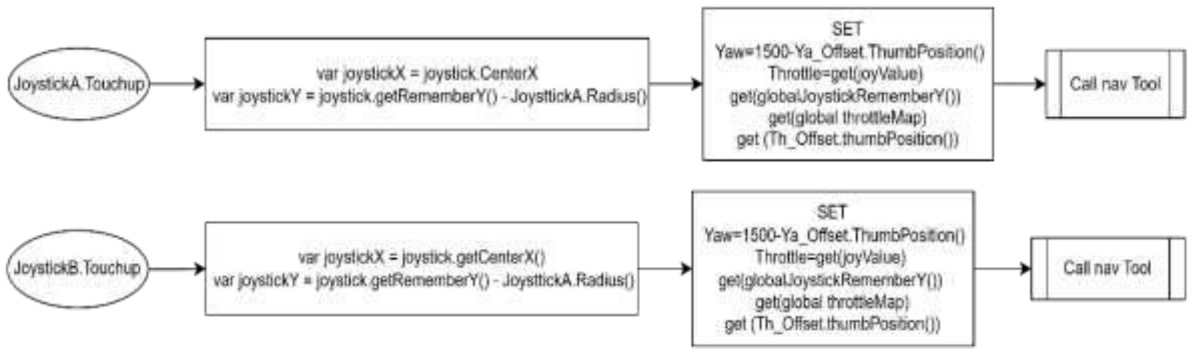

JoystickA.Touchup
var joystickX = joystick.CenterX var joystickY = joystick.getRememberY() - JoystickA.Radius()
SET Yaw=1500-Ya_Offset.ThumbPosition() Throttle=get(joyValue) get(globalJoystickRememberY()) get(global throttleMap) get (Th_Offset.thumbPosition())
Call nav Tool
JoystickB.Touchup
var joystickX = joystick.getCenterX() var joystickY = joystick.getRememberY() - JoystickA.Radius()
SET Yaw=1500-Ya_Offset.ThumbPosition() Trottle=get(joyValue) get(globalJoystickRememberY()) get(global throttleMap) get (Th_Offset.thumbPosition())
Call nav Tool

Fig. 18

e. The function checks if AUX1 is 'ON' and if not, sets its inner text to 'AUX1: ON', background colour to #808080, and variable AUX1_Val to 0. If true it sets its inner text to 'AUX1: OFF', background colour to #AAAAAA, and variable AUX1_Val to 1. The process concludes with a call to "navTool". The AUX1 switch determines whether the drone is armed or disarmed allowing it to control the aircraft. Similarly, it is done for AUX2 to determine the aircraft's light status as shown in Fig.19.

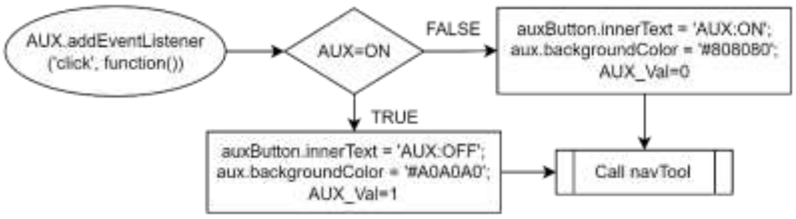

AUX.addEventListener ('click', function())
AUX=ON
FALSE
auxButton.innerText = 'AUX:ON'; aux.backgroundColor = '#808080'; AUX_Val=0
TRUE
auxButton.innerText = 'AUX:OFF'; aux.backgroundColor = '#A0A0A0'; AUX_Val=1
Call navTool

Fig.19

f. The “visorLamp” function as shown in Fig.20 is to determine whether the lamp is currently "off" or "on" state. If the inner text is not equal to "LED_OFF" the system interprets the lamp as in the "on" state and proceeds with actions such as setting the inner text to "LED_OFF", changing the background colour to #808080 and setting the variable Lamp_Val to 0. If the inner text is equal to "LED_OFF" the system updates the inner text to "LED_ON" and changes the background colour to #00AA00 also sets the variable Lamp_Val to 1. The final action is to call a function called “navTool” after the condition check and state update are completed.

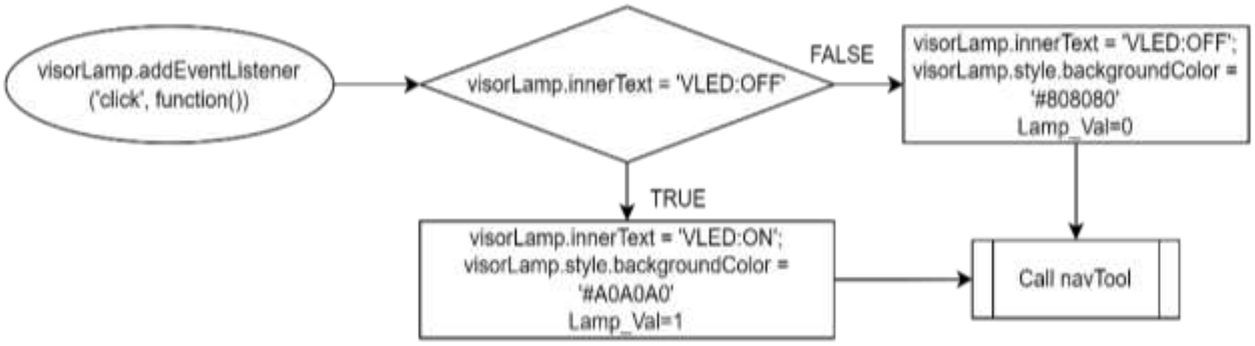

visorLamp.addEventListener ('click', function())
visorLamp.innerText = 'VLED:OFF'
FALSE
visorLamp.innerText = 'VLED:OFF'; visorLamp.style.backgroundColor = '#808080' Lamp_Val=0
TRUE
visorLamp.innerText = 'VLED:ON'; visorLamp.style.backgroundColor = '#A0A0A0' Lamp_Val=1
Call navTool

Fig.20

g. On selecting Settings button, the menu options are displayed on the left i.e. Communication & Camera and one for closing the menu. On the right Communication Settings menu is displayed as default. This menu contains IP Address, Offsets, Channel Direction & Throttle Calibration Settings.

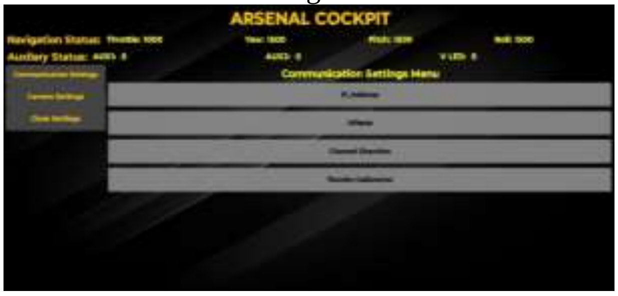


Fig.21

h. On selecting the IP Address button on the communication settings menu, a tab gets opened. IP Address can be provided to the text box and can be set using the Set button. That is going to be the current IP Address to which instructions are to be delivered. The tab can be closed by Back button.

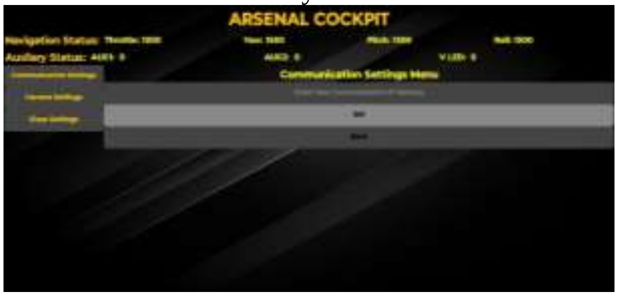


Fig.22

The procedure represents “COMIPSetButton” function which triggers the following action: setting the global variable IPA_COM to the value of COM_IPA. This setup ensures that when the “COMIPSetButton” is clicked, the global variable IPA_COM is updated based on the value of COM_IPA as shown in Fig.23.

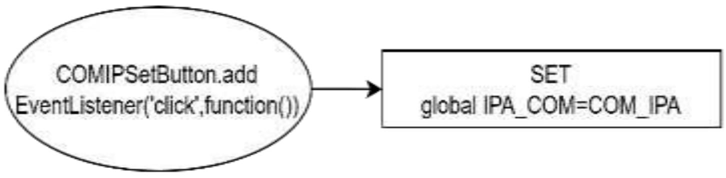


Fig.23

i. On selecting the Channel Direction button on the communication settings menu, we get into the channel direction control. The four channels i.e. Throttle Yaw, Pitch & Roll can be mapped for forward or reverse direction. This process doesn't require any instructions to be delivered as they are added to the joystick before transmitting.

Fig.24

The Channel Map Switch Logic is triggered by the event Throttle/Yaw/Pitch/Roll i.e. "ChannelMapSwitch.onChanged" which initiates actions within the system. The system analyses the state of ChannelMapSwitch.On that determines the application's response. If the switch is off, the system confirms that ChannelMapSwitch.On is false and updates ChannelMapLabel.Text to "ChannelLayout: -". It sets the variable ChannelMap to 0 indicating a disabled throttle mapping. When the switch is on it updates ChannelMapSwitch.On to true and changes the ChannelMapLabel.Text to "ChannelLayout: +" as shown in Fig.25.

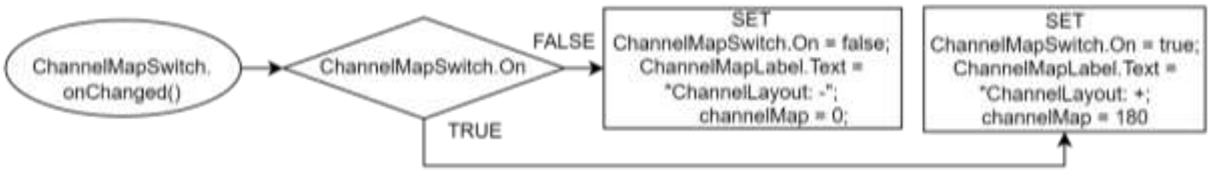

ChannelMapSwitch. onChanged()
ChannelMapSwitch.On
FALSE
TRUE
SET
ChannelMapSwitch.On = false;
ChannelMapLabel.Text = "ChannelLayout: -";
channelMap = 0;
SET
ChannelMapSwitch.On = true;
ChannelMapLabel.Text = "ChannelLayout: +;
channelMap = 180

Fig.25

j. On selecting the Offsets button on the communication settings menu, we get into the offset control. The four channels i.e. Throttle Yaw, Pitch & Roll can be adjusted for offsets if there is any error in any channel or in the receiver end. The maximum adjustment can be between x-50 to x+50. These processes don't require any instructions to be delivered as they are added to the joystick before transmitting.

Fig.26

The event is occurred when the throttle offset is changed. The "Th_Offset_Value = Th.Offset + Th.ThumbPosition" step adds the current Th.Offset value to the input ThumbPosition, indicating the new throttle position depending on human interaction with the throttle control button. Same event is generated when the (yaw, pitch, roll) individual offset is changed as shown in Fig.27.

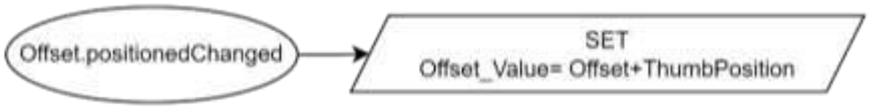


Fig.27

k. On selecting the Throttle Calibration button on the communication settings menu, we get into the throttle channel calibration settings. The Throttle channel can be calibrated for better flight or throttle response. This process is added as an instruction to be delivered by the transmitter.

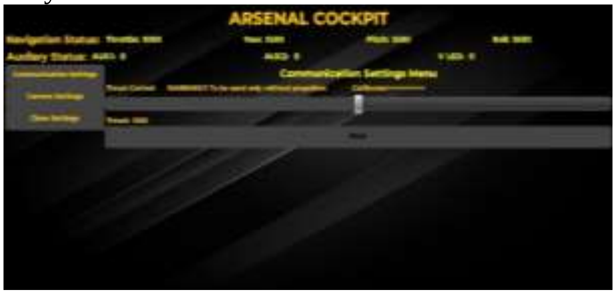
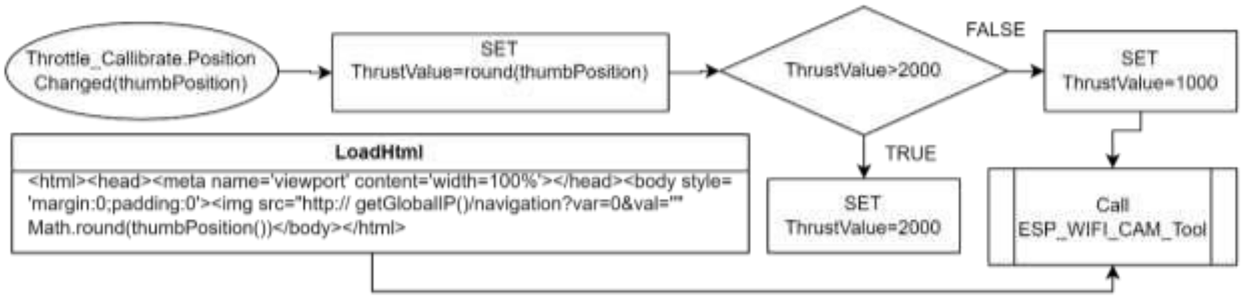


Fig.28

The calibration process starts when the Throttle_Calibrate_Position ChangedEvent(thumbPosition) event is triggered, which is activated when the user moves or adjusts the throttle control's thumb position. The system sets ThrustValue to the rounded value of thumbPosition. If ThrustValue > 2000, the system sets ThrustValue to 2000. If ThrustValue is less than or equal to 2000, the system sets ThrustValue to 1000. The ESP_WIFI_CAM_Tool is executed which interacts with the ESP-32 Wi-Fi camera module to validate throttle calibration as shown in Fig.29.



```
<html><head><meta name='viewport' content='width=100%'></head><body style=
'margin:0;padding:0'><img src="http:// getGlobalIP()/navigation?var=0&val=""
Math.round(thumbPosition())</body></html>
```

Fig.29

l. On selecting Camera Settings button, the other menu options are displayed on the left. On the right Camera Settings menu is displayed. This menu contains Stream Control, Resolution, Basic Settings & Alignment.

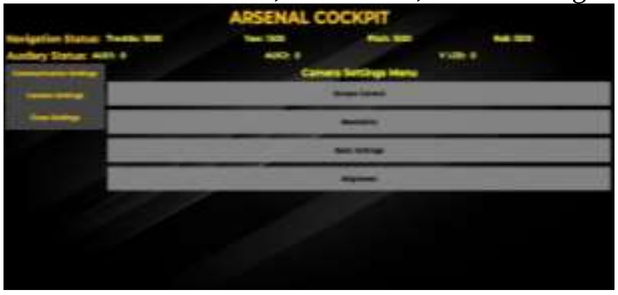


Fig.30

m. The Stream Control tab is opened by the Stream Control button from the Camera Settings menu. The Stream can be started or stopped from the current tab. This process generates an instruction to be delivered by the transmitter.

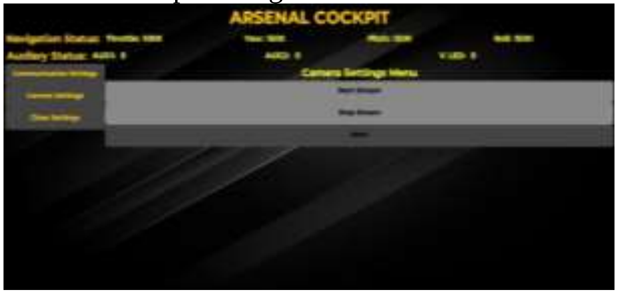


Fig.31

The StartStreamButton initiates the streaming service. This process involves setting the ESP_Navigation_Layer.Image to false and calling the ESP_WIFI_CAM function. The HTML page is loaded and display the stream from the specified global IP address as shown in the Fig.32.

ISSN(O): 2456-6683
[Impact Factor: 9.241]

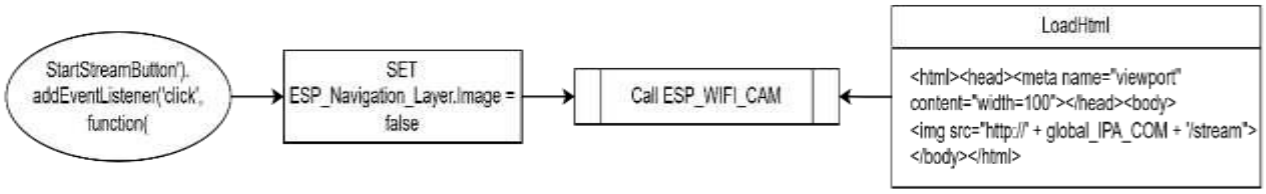

StartStreamButton').
addEventListener('click',
function(
SET
ESP_Navigation_Layer.Image =
false
Call ESP_WIFI_CAM
LoadHtml
<html><head><meta name="viewport"
content="width=100"></head><body>
<img src="http://" + global_IPA_COM + '/stream">
</body></html>

Fig.32

n. The StopStreamButton sets the ESP_Navigation_Layer.Image to Black.png. The ESP_WIFI_CAM function is called and the HTML page is loaded, which includes viewport settings for mobile responsiveness and an iframe sourcing content from an IP address concatenated with "/stream". This stops the current stream and prepares the interface for the next action as shown in Fig.33.

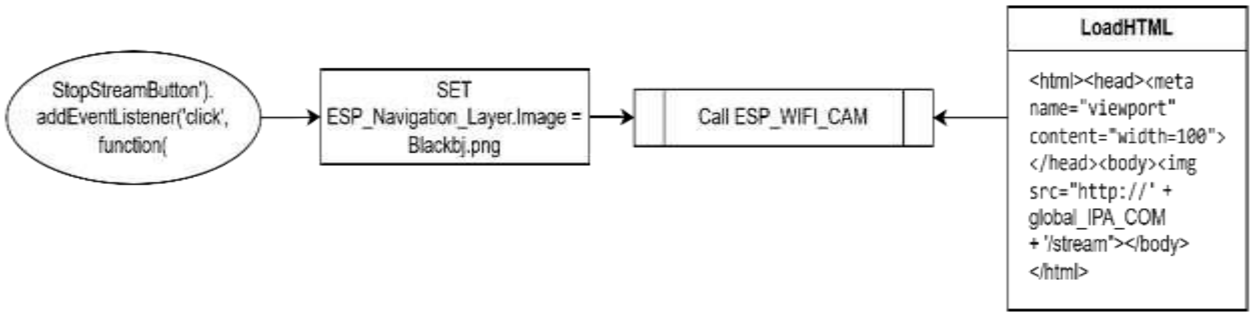


Fig.33

o. The Resolution selection tab is opened by the Resolution button from the Camera Settings menu. The Resolution can be changed to higher or lower resolution from the current tab. This process generates an instruction to be delivered by the transmitter.

Fig.34

- **QQVGA Button**: the process begins with the event "QQVGAButton.addEventListener('click', function()". Set the global variable val to 0. The function executeHTML is called.
- **QVGA Button**: The process starts with the event "QVGAButton.addEventListener('click', function()". Set the global variable to val 4. The function executeHTML is called.
- **HQVGA Button**: The process begins with the event "HQVGAButton.addEventListener('click', function()". Set the global variable to val 3. The function executeHTML is called.
- **CIF_Button**: The process starts with the event "CIF_Button.addEventListener('click', function()". Set the global variable to val 5. The function executeHTML is called..
- **VGA Button**: The process begins with the event "VGAButton.addEventListener('click', function()". Set the global variable to val 6. The function executeHTML is called.
- **SVGA Button**: The process starts with the event "SVGAButton.addEventListener('click', function()". Set the global variable to val 7. The function executeHTML is called.
- **XGA Button:** The process begins with the event "XGAButton.addEventListener('click', function()". Set the global variable to val 8. The function executeHTML is called.
- **SXGA Button:** The process starts with the event"SXGAButton.addEventListener('click', function()". Sets the global variable to val 9. The function executeHTML is called.

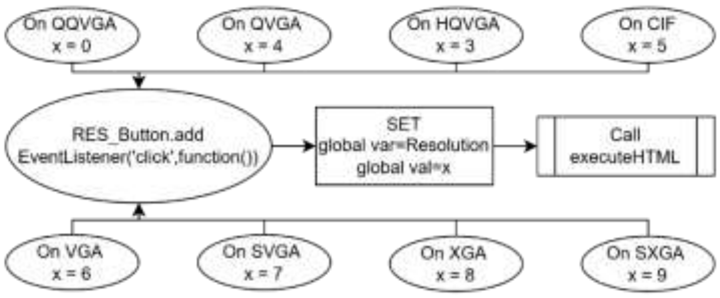


Fig.35

p. The Alignment tab is opened by the Alignment button from the Camera Settings menu. The Alignment settings i.e. vertical flipping or horizontal mirroring can be done or undone from the current tab. This process generates an instruction to be delivered by the transmitter.

Fig.36

The “VFlipSwitch.changed” function triggers actions based on VFlipSwitch = ON. If true, it sets VFlipSwitch.On to true, updates VFlipSwitch.text to "ON", and sets global var to 1. If false it sets VFlipSwitch.On to false, updates VFlipSwitch.text to "Off" and sets global var to 0. This setup ensures that when the VFlipSwitch is changed the system updates the switch’s state and text and sets a global variable to reflect the switch’s status as shown in Fig.37

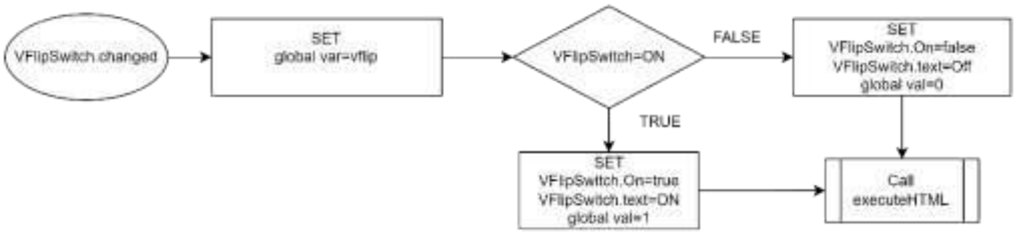


Fig.37

The “HMirrorSwitch.changed” function triggers actions based on the condition HMirror = ON. If true, it sets HMirrorSwitch.On to true, updates HMirrorSwitch.text to "ON", and sets global val to 1, while if false, it sets HMirrorSwitch.On to false, updates HMirrorSwitch.text to "Off", and sets global val to 0. This setup ensures that when the HMirrorSwitch is changed, the system updates the switch’s state and text and sets a global variable to reflect the switch’s status as in the Fig.38.

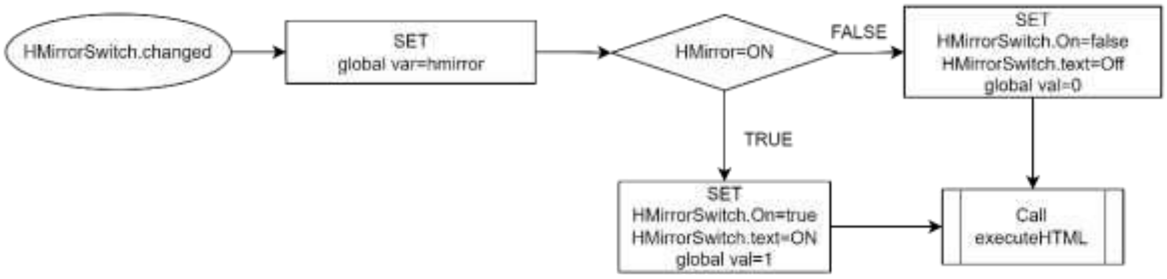


Fig.38

q. The Basic Settings tab is opened by the Basic Settings button from the Camera Settings menu. The basic settings i.e. Quality, Brightness, Contrast & Saturation can be controlled from the current tab. This process generates an instruction to be delivered by the transmitter.

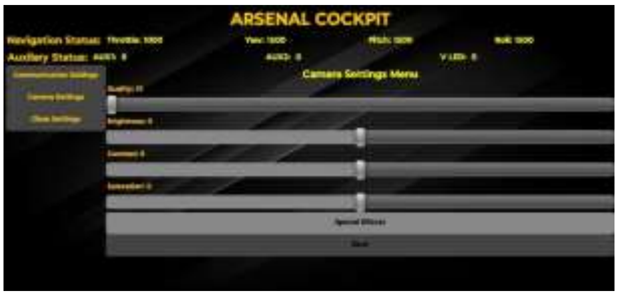


Fig.39

"QualitySlider.PositionChanged(thumbPosition)" triggers actions like setting global var to the value of quality, updating the Quality label to the rounded value of QualitySlider.ThumbPosition and calling the executeHTML function. The system updates the relevant variables and labels and executes the HTML code as shown in Fig.40.

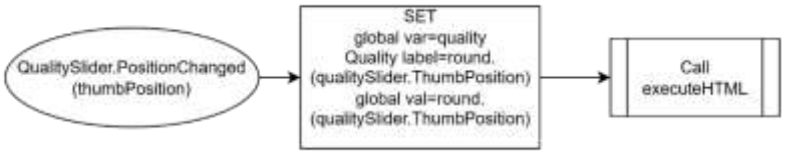

QualitySlider.PositionChanged (thumbPosition)
SET
global var=quality
Quality label=round.
(qualitySlider.ThumbPosition)
global val=round.
(qualitySlider.ThumbPosition)
Call executeHTML

Fig.40

"BrightnessSlider.PositionChanged(thumbPosition)" triggers the following actions: setting global var to the value of Brightness, updating the Brightness label to the rounded value of BrightnessSlider.ThumbPosition, and setting global val to the rounded value of BrightnessSlider.ThumbPosition. Finally, it calls the executeHTML function. This setup ensures that when the position of the BrightnessSlider changes, the system updates the relevant variables and labels, and executes the HTML code. as shown in Fig.41.

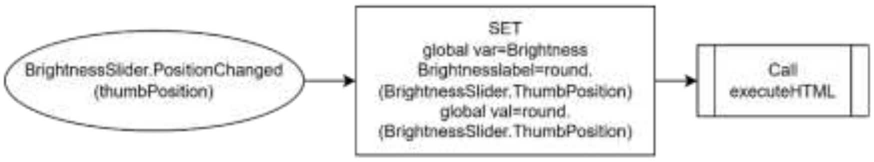

BrightnessSlider.PositionChanged (thumbPosition)
SET
global var=Brightness
Brightnesslabel=round.
(BrightnessSlider.ThumbPosition)
global val=round.
(BrightnessSlider.ThumbPosition)
Call executeHTML

Fig.41

The process starts with the event "ContrastSlider.PositionChanged (thumbPosition)". The global variable is set using the formula globalVar = round(ContrastSlider.ThumbPosition). The function executeHTML is called to update the HTML content based on the new contrast value. as shown in Fig.42.

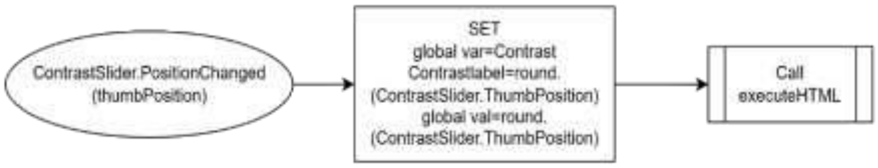

ContrastSlider.PositionChanged (thumbPosition)
SET
global var=Contrast
Contrastlabel=round.
(ContrastSlider.ThumbPosition)
global val=round.
(ContrastSlider.ThumbPosition)
Call executeHTML

Fig.42

The process begins with the event "SaturationSlider.PositionChanged (thumbPosition)." A global variable is set using the formula globalVar = round(SaturationSlider.ThumbPosition). The function executeHTML is called to update the HTML content based on the new saturation value. as shown in Fig.43.

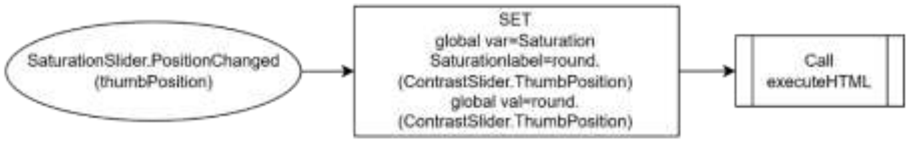

SaturationSlider.PositionChanged (thumbPosition)
SET
global var=Saturation
Saturationlabel=round.
(ContrastSlider.ThumbPosition)
global val=round.
(ContrastSlider.ThumbPosition)
Call executeHTML

Fig.43

r. The Special Effects tab is opened by the Special Effects button from the Basic Settings tab. The special effects like No Effect, Invert, Gray Scale, Red Tint, Green Tint, Blue Tint & Sepia can be applied to the frames that are streaming from the current tab. This process generates an instruction to be delivered by the transmitter.

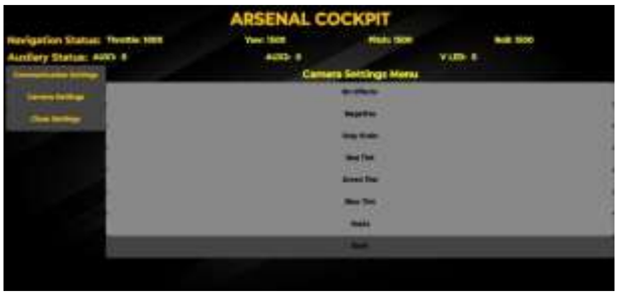


Fig.44

The process begins with the event “NoEffectClickButton”. The global value is set to 0 using the formula global val = 0. The function executeHTML is called. Similarly, for (GrayScale, RedTint, GreenTint, BlueTint, SepiaTint) Buttons, global value (1,2,3,4,5,6) is updated accordingly as shown in Fig.45.

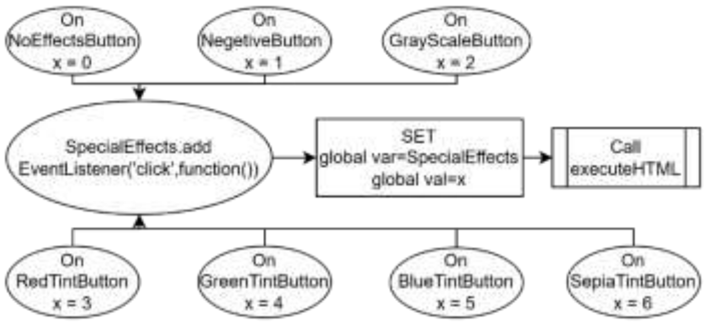
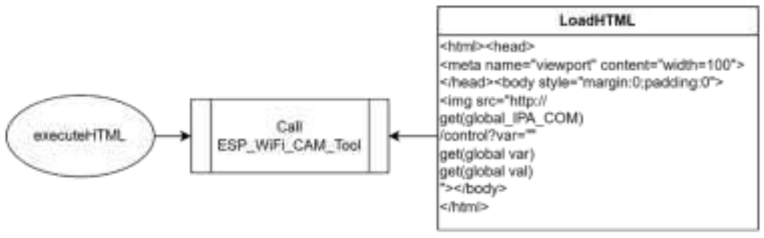


Fig.45

IV.Network control:

The executeHTML function used to update various elements on the web page such as images, text, or styles based on the values of global variables. when a user adjusts a slider or clicks a button, the global variable **val** and **var** is set as per users’ selection that are picked up by evet listeners and executeHTML updates the HTML to reflect this change. By calling executeHTML the application ensures that the user interface remains interactive and responsive. executeHTML is often called after setting global variables or interacting with other tools (like ESP_WiFi_CAM_Tool) as shown in Fig.46



```
<html><head>
<meta name="viewport" content="width=100">
</head><body style="margin:0;padding:0">
<img src="http://
get(global_IPA_COM)
/control?var=""
get(global var)
get(global val)
"></body>
</html>
```

Fig.46

V.Error detection and masking:

The errors detection scheme in Fig.47 describes a decision-making process for handling error in the application. It starts with the assumption that an error has occurred and checks if the error number is either 1101 or 1103. If this condition is met or not met, the process mask any errors that displays “Unable to get a response with the specified URL: http://192.168.4.1/”.

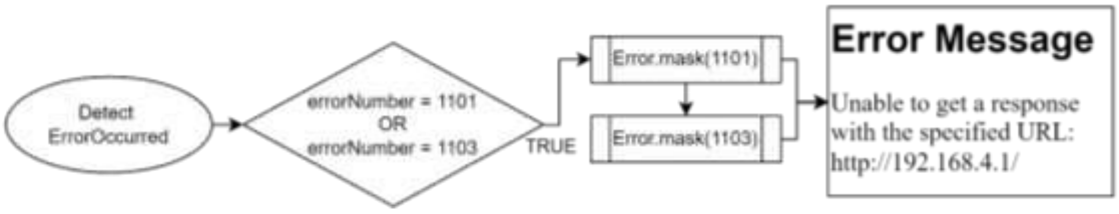

Detect ErrorOccurred
errorNumber = 1101 OR errorNumber = 1103
TRUE
Error.mask(1101)
Error.mask(1103)
Error Message
Unable to get a response with the specified URL: http://192.168.4.1/

Fig.47

3. RESULT AND DISCURSSION:

Based on the conducted study the results found out were differing in both terms of design and functionality. The final design is given in Fig.48 and its output is shown in Fig.49. A brief comparison of results is given below:

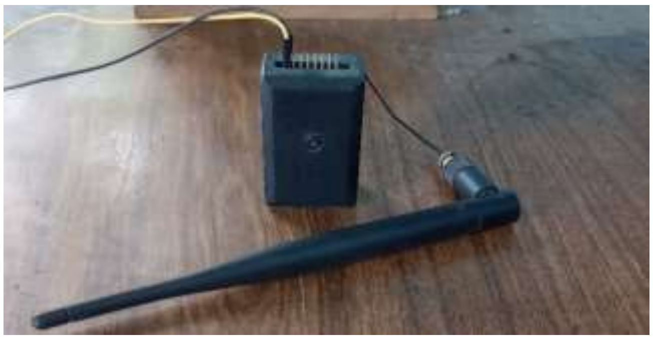


Fig.48

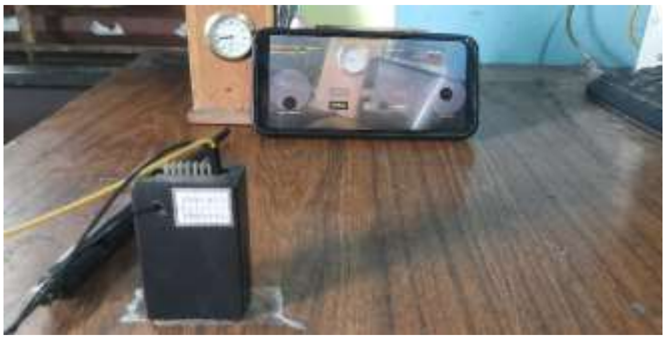


Fig.49

A. Reduction of hardware:

Previously the system had ESP-07 for receiving flight channel data like Throttle, Yaw, Pitch, Roll, AUX1 and AUX2 from the android application. The frame stream was generated by ESP32-S CAM and transmitted to the android application. Overall, there was a delay in response due to crowding on channel. Hardware used in the previous project includes:

I. Ai Thinker ESP32-S CAM Wi-Fi Module with OV2640 Camera Fig.50
II. Ai Thinker ESP-07 ESP8266 Serial Wi-Fi Module Fig.51
III. MT76813DBI ESP8266 Serial Wi-Fi wireless Gain Antenna Fig.52
IV. ESP8266 Adapter Plate Serial Wireless WIFI Module Fig.53
V. AMS1117 3.3V Power Supply Module Fig.54

Fig.50

Fig.51

Fig.52

Fig.53

Fig.54

Circuit was complex for which the device had issues at functioning and antenna had less communication range for it uses twin 3dBi antenna.

In the current system ESP32-S CAM acts as both transmitter of frames and receiver of flight channel data. Overall, there was less delay in response due to no crowding on channel. Hardware used in the current project includes:

I. Ai Thinker ESP32-S CAM Wi-Fi Module with OV2640 Camera Fig.xx
II. 698-960 MHz and 1710-2690 MHz / 5dBi Gain Dual Band 3G / 4G LTE Antenna Fig.xx

Fig.55

Fig.56

Circuit is simpler in this case for which device is quick to respond and function. Now, the antenna also has wider communication range for it uses 5dBi antenna.

B. Application design & optimization:

I. In the previous system the Android application had its Screen 1 in Fig.57 take IP address for channel data and a container of Screen 2 in Fig.58 take IP address for camera.

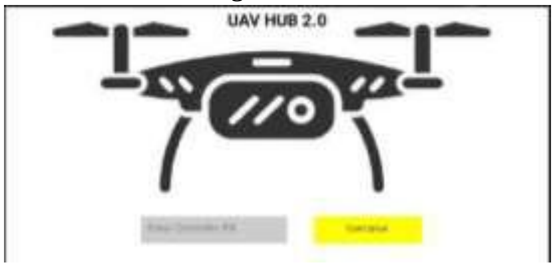


Fig.57

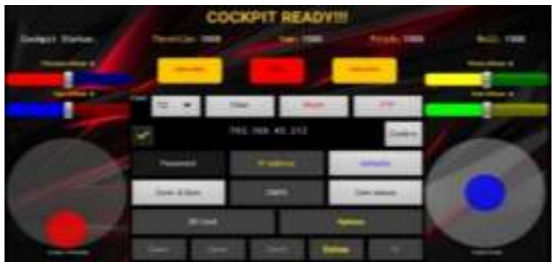


Fig.58

In the current system however Android application had its Screen 2 in Fig.59 take a single IP address for channel data and for camera.

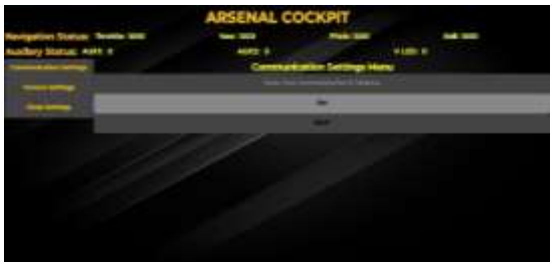


Fig.59

II. In the previous system the Android application had no separate interface for communication and camera settings. Additionaly only 4 channel offset sliders were provided to adjust as shown in Fig.60.

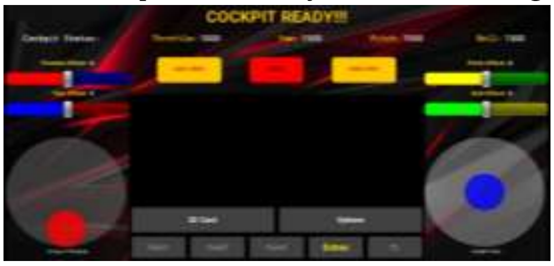


Fig.60

In the current system the Android application has separate interface for communication shown in Fig.61 and camera settings shown in Fig.62.

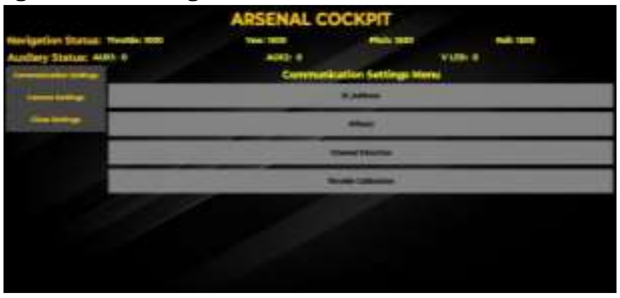


Fig.61

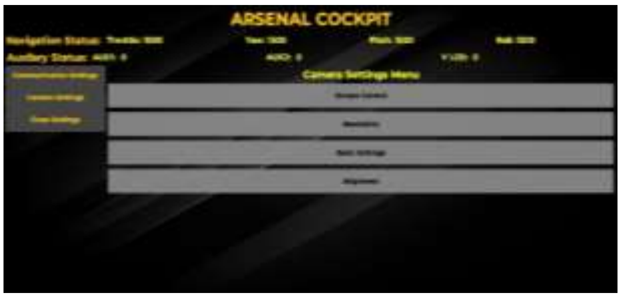


Fig.62

III. In the previous system the video stream had a container attached for streaming as shown in Fig.63, whereas the present system streams in the background and the controller functions i.e. Joystick, Buttons and display functions like statuses are displayed in the foreground as shown in Fig.64 of the Android application.

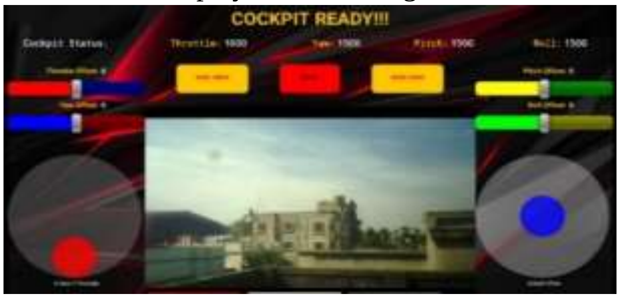


Fig.63

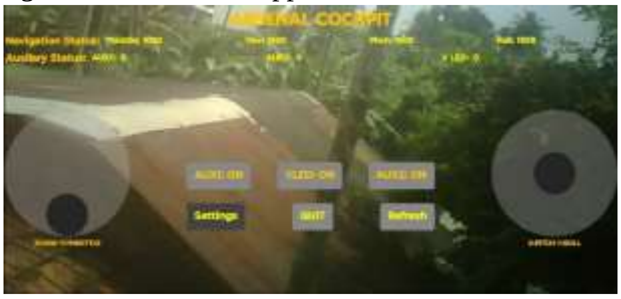


Fig.64

C. Cost effectivity:

Cost of each devices required to build the previous system is shown below in Fig.65

Sl.No.	Device Name	Quantity	Cost (in INR)
1	Ai Thinker ESP32-S CAM Wi-Fi Module with OV2640	1	849.00
2	Ai Thinker ESP-07 ESP8266 Serial Wi-Fi Module	1	229.00
3	MT76813DBI ESP8266 Serial Wi-Fi wireless Gain Antenna	2	61.00
4	ESP8266 Adapter Plate Serial Wireless WIFI Module	1	28.00
5	AMS1117 3.3V Power Supply Module	1	11.00

So, the total cost of the system is 1239 INR as per price on Robu (Dated:28/09/2024).
Cost of each devices required to build the current system is shown below in Fig.66

Sl.No.	Device Name	Quantity	Cost (in INR)
1	Ai Thinker ESP32-S CAM Wi-Fi Module with OV2640	1	849.00
2	698-960 & 1710-2690MHz/5dBi Gain Dual Band 3G/4G LTE Antenna	1	199.00

Fig.66

So, the total cost of the system is 1048 INR as per price on Robu (Dated:28/09/2024).
As we can see, the cost is also reduced by 191 INR after modifications done on the hardware.

4. CONCLUSION:

With the study conducted above there are several key points like scope of related work, limitations and future scope of the present work are to be concluded. The current work was taken up as to update the past work. The frame transmitter, controller and wireless receiver is updated to achieve modularity, portability and cost effectiveness. Firstly, the ESP32 based wireless receiver being affordable makes it almost an ideal alternative for developers and professionals at entry level for creating advanced drone systems. The dual core processor makes it`s multitasking efficient and processes for both camera and flight are carried out with ease. Secondly, the modular design enables ready to switch upgrades or modifications. This increases the need for plug and play devices having fostered innovation and continuous improvement of wireless controllers in UAV technology making it a compelling choice for modern aerial applications. Finally, the Wi-Fi based UAV transceiver device approaches to combine several technologies into one to achieve one stop solution for several problem. Modern users, in the public domain prefer compact feature rich smartphone transceiver over a bulky single action device.

The Wi-Fi based UAV transceiver device at this point has limitations on onboard memory restricting complex computational task like advanced image processing or data analysis in real time. The number of onboard pins available plus some limitation related to clock sharing makes it hard to make the data transfer port multi-functional and number of channels had to be reduced to 7.

Wi-Fi based UAV transceiver in the future can be extended with more I/O ports providing more channels for interfaced transmission, additional PSRAM for complex tasks like night vision, facial recognition based on image processing or artificial intelligence. It can also be extended to communicate with other equipment on board to provide better experience to an individual.

5. ACKNOWLEDGEMENT:

The open-source embedded processors of ESP (Esspresif Systems) and integrated development environment (IDE) of Arduino, MIT (Application Development Environment) and tools of others organization are used due course of this work for which we are highly thankful. Special thanks to New Alipore College, University of Calcutta for providing laboratory support during the development of work.

Prototyping development kits and boards are brought from Robu.in. Device image credit: Robu.

REFFERENCES:

1. Taylor, John W. R, “Jane's Pocket Book of Remotely Piloted Vehicles”.
2. Professor A. M. Low FLIGHT, "The First Guided Missile", 3 October 1952, pp. 436.
3. Dunstan, Simon, 2013, “Israeli Fortifications of the October War 1973”, ISBN 9781782004318, Osprey Publishing. pp. 16, “The War of Attrition was also notable for the first use of UAVs, or unmanned aerial vehicles carrying reconnaissance cameras in combat”.
4. Tom Scheve, “A Brief History of UAVs”, 22 July 2008. “How the MQ-9 Reaper Works”, howstuffworks.com.
5. "Russia Buys a Bunch of Israeli UAVs", 9 April 2009, strategypage.com.
6. Azoulai Yuval, “Unmanned combat vehicles shaping future warfare”, 24 October 2011, en.globes.co.il, Article No. 1000691790.
7. Charles Levinson, “Israeli Robots Remake Battlefield”, 13 January 2010, The Wall Street Journal, p. A10.
8. David Hambling, “Drones may have attacked humans fully autonomously for the first time”, 27 May 2021, New York Post.
9. Paula Froelich, “Killer drone 'hunted down a human target' without being told to”, 29 May 2021, New York Post.
10. Brian Fung, “Why drone makers have declared war on the word drone”, 16 August 2013, The Washington Post.
11. Matt McFarland, “In Switzerland, police find a use for drones”, 17 September 2014, The Washington Post.

12. M. Saska, V. Vonasek et.el., “Coordination and Navigation of Heterogeneous UAVs-UGVs Teams Localized by a Hawk-Eye Approach”, Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.
13. “Low-cost, high-speed SGI Indy comes with camera”, 13 August 1993, Machine, Design, ProQuest 217148786, Vol. 65, no. 16, pp. 84.
14. “Video input becoming workstation standard”, 22 July 1993, Electronic Design, ISSN 0013-4872, Vol. 41, no. 15, pp. 30.
15. “Interview with Martin Gren, inventor of the network camera”, SDM Magazine, 18 October 2011.
16. John Adams, “Martin Gren: IP CCTV's Founding Father”, 8 December 2015, Security Electronics and Networks.
17. J. Díaz, “Introduction to ESP32-CAM: A Smart Camera Module”, 2019.
18. A. López, “Getting Started with ESP32-CAM for Drone Applications”, 2020.
19. P. Sharma, “Using ESP32-CAM in Drone Projects: A Comprehensive Guide”, 2021.
20. R. Gupta, “Enhancing Drone Capabilities with ESP32-CAM: Trends and Innovations”, 2022.
21. N. Patel, “Future Directions for ESP32-CAM in Aerial Robotics”, 2023.
22. S. Chatterjee, S. Moyra & A. Banerjee, “A Model Design in UAV Technology: A Multi-controller-based Wi-Fi Imaging Drone”, IJRASET, Volume 11 Issue X Oct 2023.