PHENIKAA UNIVERSITY

FALCULTY OF ELECTRONIC ENGINEERING

BASIC PROGRAMMING FOR ELECTRONICS

FINAL PROJECT REPORT

Project: Smart System utilize passive Infrared sensor to Control Fan and Light in smart home

1. Duong Doan Tung

2. Nguyen Trong Huy Hoang

3. Le Hoang Nam

HA NOI, 12/2022

TABLE OF CONTENTS

WORKLOAD DISTRIBUTION

Full name	Task
Duong Doan Tung	○ Python, Labview Programming ○ Write report
Nguyen Trong Huy Hoang	○ Setup hardware ○ Decorate, house design ○ Presentation script ○ Hardware schematic design
Le Hoang Nam	○ Edit video ○ Build hardware ○ Performance Analysis ○ Presentation design

Project include:

1. Project report
2. Presentation file and video
3. Python files
4. Labview files (21 subVIs included)

Full project material can be found here:

https://drive.google.com/drive/folders/15Gyvd-AxRdvlOK2FSbTaDAAlVZT9CAch?usp=sharing

Project repository: https://github.com/dtungpka/I.C.F.L

Special thanks to our teacher Dr. Huy Minh Le for his help in this project

Section 1: ABSTRACT

Educators and researchers worldwide are using National Instruments products to automate routine tasks, accomplish new objectives, replace outdated and expensive equipment, and demonstrate students the potential of high technology. Engineers have used virtual instrumentation for more than 25 years to bring the power of flexible software and PC technology to test, control, and design applications making accurate analog and digital measurements. To demonstrate Labview's potential and Labview's capabilities in real-world applications, in this paper, we created a Smart System that utilizes passive Infrared sensor to Control Fan and Light in smart home. This system is capable of controlling Light Emitting Diodes (LED) and DC motors through several inputs.

Section 2: PROJECT OVERVIEW

1. Inspiration

From cities and transportation to the resources we farm, emerging Internet of Thing (IoT) technologies are set to make everyday processes more integrated and easier. As the sensors, data storage, the Internet, and analytics become faster, cheaper, better, and more integrated together, users will be able to rely on smart system more. IoT devices will have a significant impact on many aspects of our lives including how we live, drive, and farm animals and crops.

The Internet of Things technology establishes a connection between all things and the Internet via sensing devices and implements intelligent identification and management. The information sensing devices include infrared sensors, GPS and laser scanner devices. They are all connected to the Internet to implement remote perception and control. IOT is widely applied in intelligent transportation, environment protection, government work, public security, smart home, intelligent fire control, industrial monitoring, elderly care, personal health, etc.

With that in mind, we set our main objective to create a smart IoT system for smart home applications and called it **Smart System utilize passive Infrared sensor to Control Fan and Light in smart home (S.S.I.C.F.L)**, this is an upgrade from mid-term project **using passive Infrared sensor to Control Fan and Light in smart home (I.C.F.L)**.

2. Basic Idea

The system consists of 2 devices: a computer and DAQ USB-6255 device.

The computer will acquire sensor data, process and send signal to control the DAQ device. DAQ device is used to read Passive InfRared sensor (PIR) data, control LED and fan.

On computer, we can control DAQ device in 3 modes:

- Schedule mode: create schedules for fan and LED (FLED)[1] to work automatically at a defined time stamp
- Manual mode: allows users to control FLED manually in 2 ways:
    - On Labview Front panel
    - By finger using Optical Recognition Module (ORM)
- Realtime mode: Read data from PIR sensor and control FLED automatically.

[1] Fan and LED

Section 3: PROJECT DETAILS

1. Hardware

1.1. Electronic parts

1.1.1. USB-6255

USB-6255 is an 80 AI (16-Bit, 1.25 MS/s), 2 AO (2.86 MS/s), 24 DIO USB Multifunction I/O Device—The USB-6255 offers analog I/O, digital I/O, two 32-bit counters/timers, and analog and digital triggering. The device delivers low-cost, reliable DAQ capabilities in a wide range of applications from simple applications in laboratory automation, research, design verification/test, and manufacturing test.

Figure 1. USB-6255 Device

1.1.2. HC-SR501

HC-SR501 is based on infrared technology, automatic control module, using Germany imported LHI778 probe design, high sensitivity, high reliability, ultra-low-voltage operating mode, widely used in various auto-sensing electrical equipment, especially for battery-powered automatic controlled products.

Figure 2. HC-SR501

1.1.3. L298N

The L298N is an integrated monolithic circuit in a 15- lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic level sand drive inductive loads such as relays, solenoids, DC and stepping motors

1.1.4. Light Emitting Diode

A light-emitting diode (LED) is a semiconductor device that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons.

1.1.5. DC Motor

A DC motor is any of a class of rotary electrical motors that converts direct current (DC) electrical energy into mechanical energy. The most common types rely on the forces produced by induced magnetic fields due to flowing current in the coil.

1.2. Wiring diagram

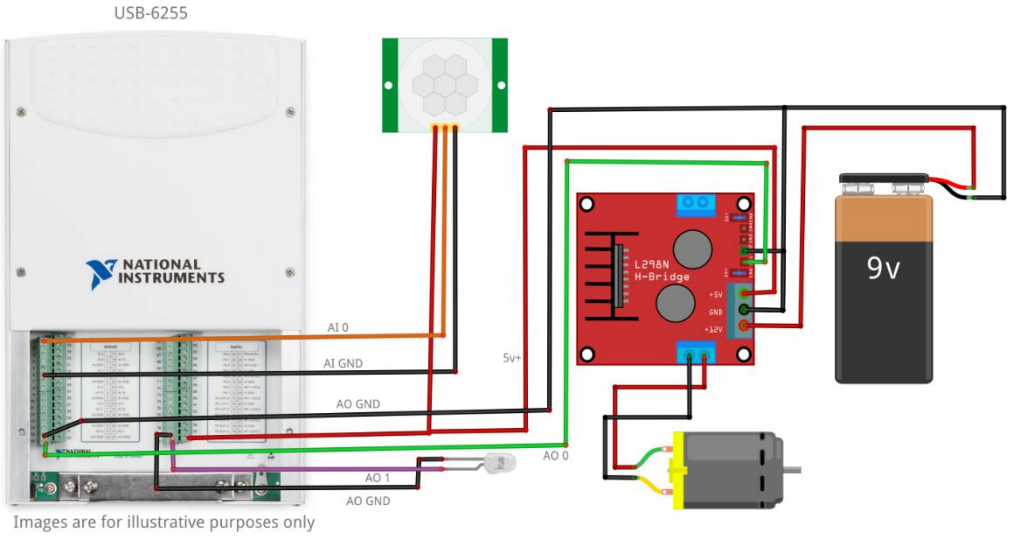


Figure 3. Circuit schematic

We created a simple circuit that reads the PIR sensor input in Terminal 1: AI0 of USB-6255. FLED are controlled by Analog output terminal AO0, AO1 respectively. We use LM298 DC Motor Driver Controller to amplify the signal from DAQ to 9 volt and use it to control the Fan.

1.3. House model design

We designed a 3-floor house and built it using formex and stick. There are a total of 6 LED lights with 2 lights on each floor and a fan on the balcony on the 3rd floor. We also decorate it with 3 solar panels on top.

Figure 4. House design

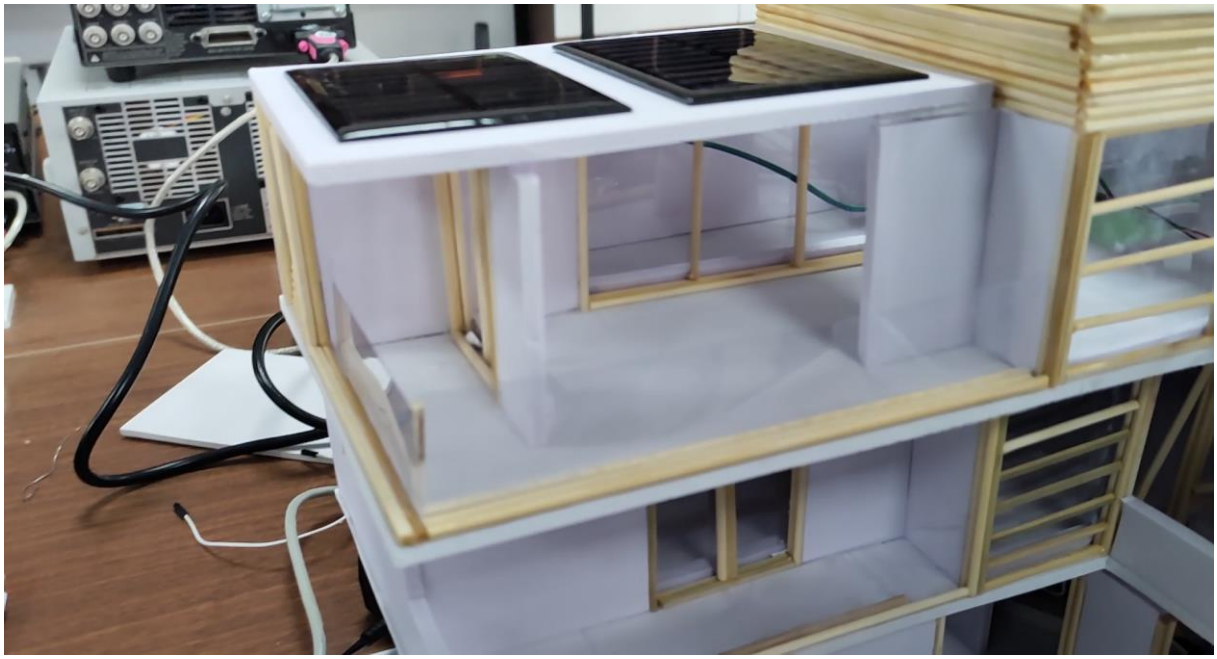


Figure 5. House close view

2. Software

2.1. Labview

2.1.1. User Interface

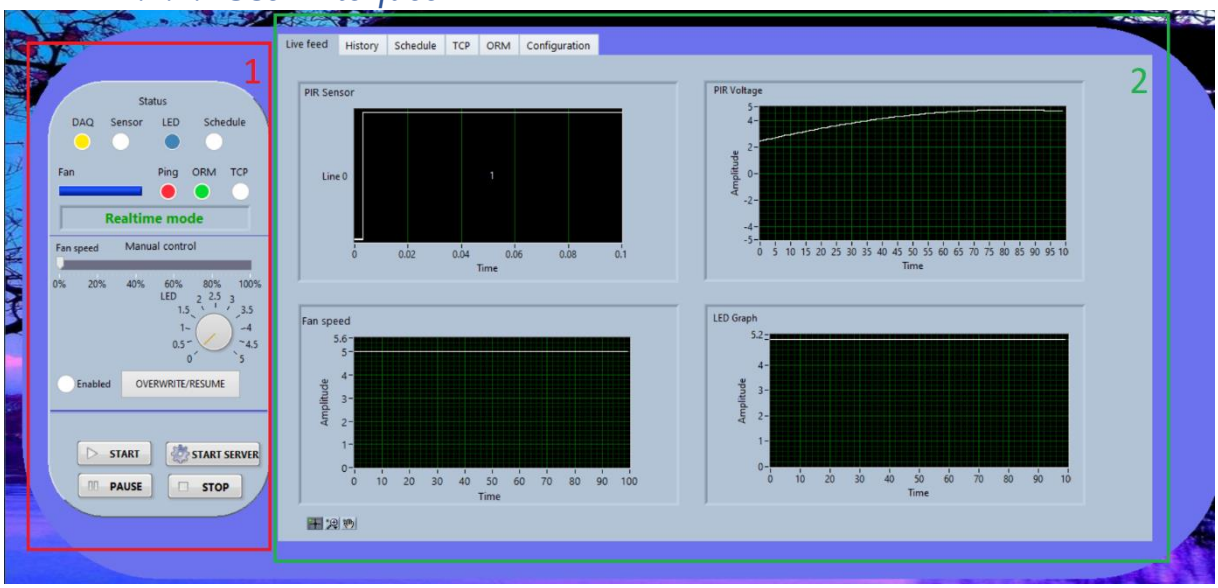


Figure 6. Labview user interface on live feed tab

The user interface (front panel) in UI Main.vi file can control every function of this project. It's contained 2 parts[1]:

- Control panel (1) display status[2] of the program and control FLED manually.
- Tabs (2) contain live feed data from DAQ, TCP server and client information, ORM module and schedule mode control.

2.1.1.1. Manual control

User can control FLED manually by drag and drop the Fan speed slider and LED knob in Control panel. When those values changed, enable light would turn yellow if manual mode not enabled yet. Click on OVERWRITE/RESUME will turn on manual mode and apply change to FLED. Click on it again to resume to the previous mode and end manual mode.

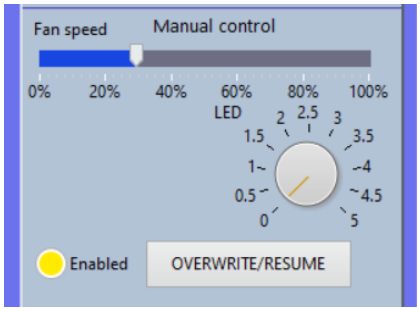


Figure 7. Manual control interface

2.1.1.2. Live feed tab

Live feed tab show latest 100 datapoint in real time:

- PIR voltage is the raw data from DAQ
- PIR Sensor shows processed sensor data, 1 means passed the threshold, 0 mean below.
- Fan speed and LED graph show Output data to FLED.

[1] Figure 6
[2] For more information, see Appendix B

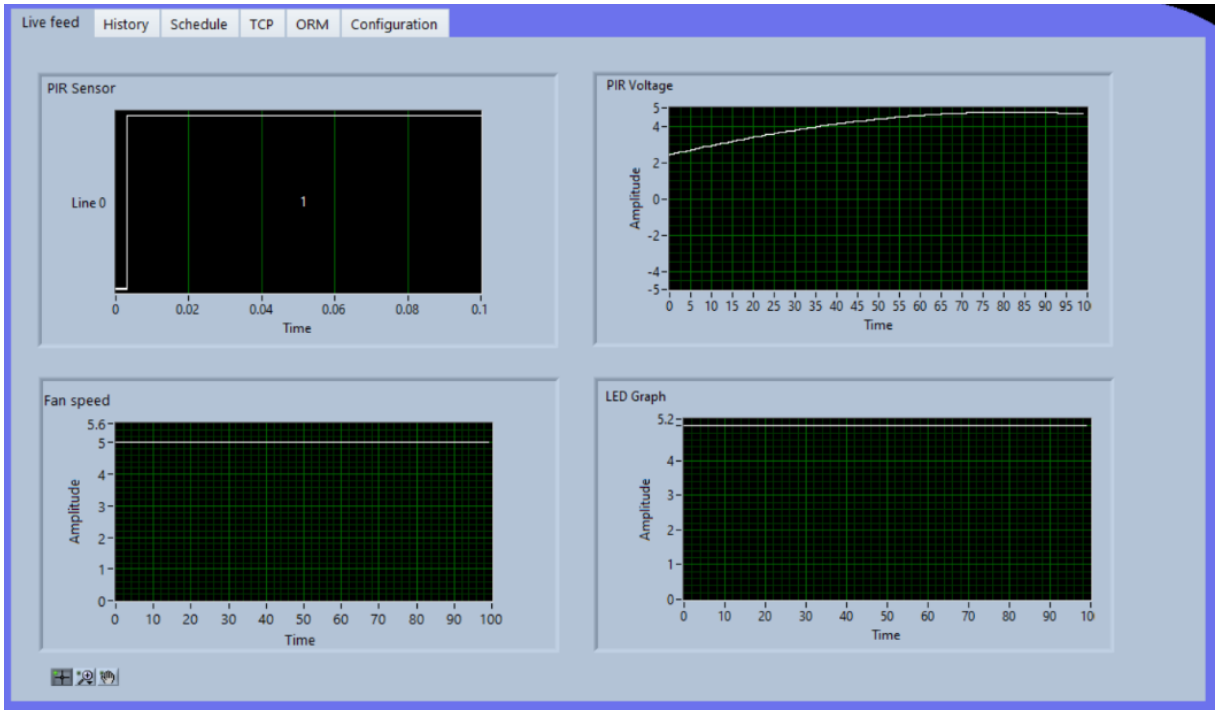


Figure 8. Livefeed tab

2.1.1.3. History tab

History tab show all data from current session, it can also save and load data.

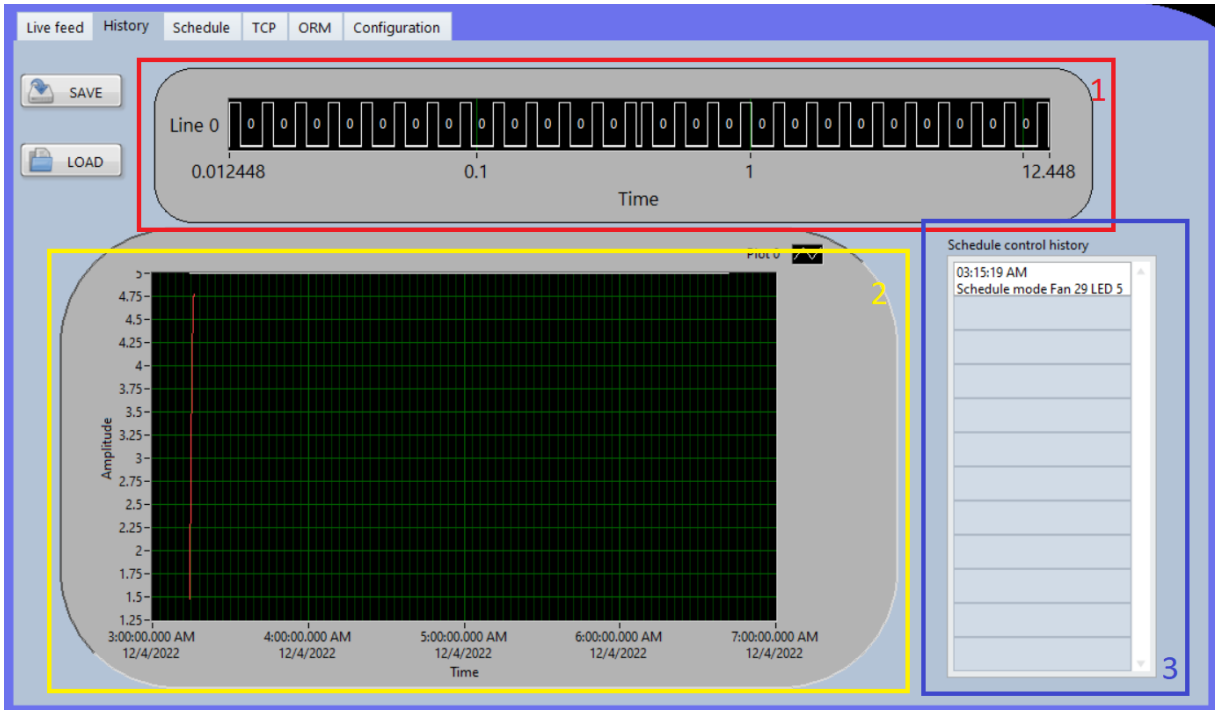


Figure 9. History tab

Graph (1) is PIR sensor data processed, (2) contain 2 lines: Red line is PIR sensor voltage level, white is FLED output data. (3) data table past schedule that executed.

2.1.1.4. Schedule tab

On schedule tab, user can create, modify, delete schedule. Available action are: Light on, Light off, Fan on, Fan off, All on, All off. After created, the schedule will appear on the pending list below in execute time order. Switch on Enable button will change program to schedule mode and each pending schedule will display their time till activation[1]. After activated, the schedule will be deleted.

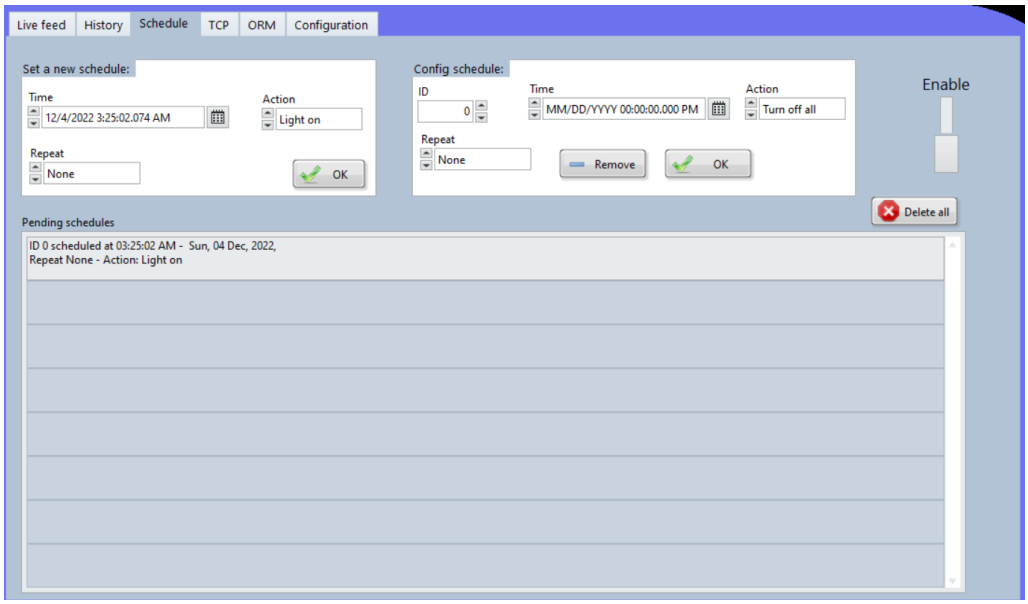


Figure 10. Schedule tab - Disabled

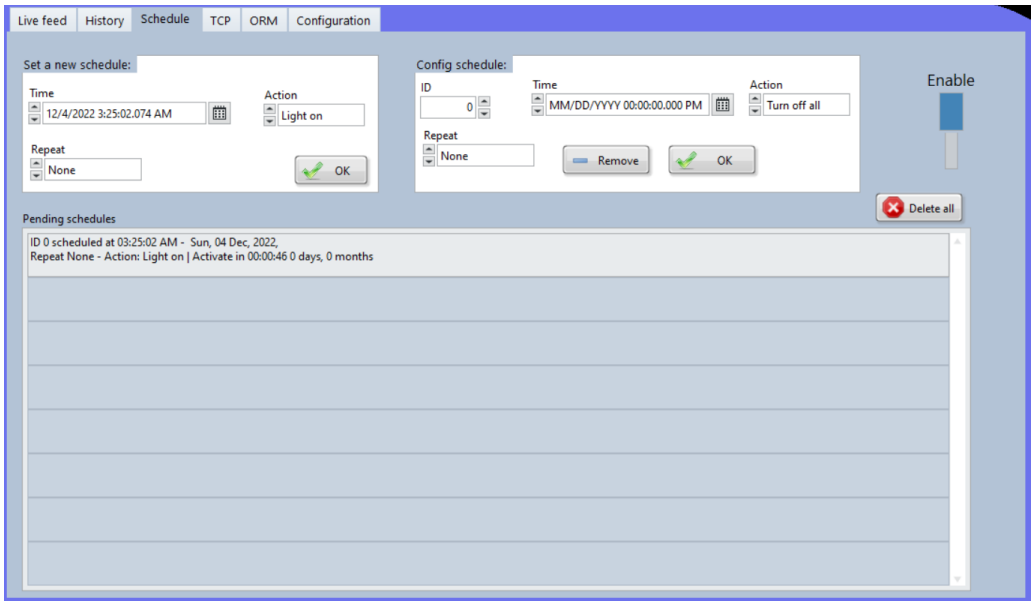


Figure 11. Schedule tab – Activated

[1] Figure 11

2.1.1.5. TCP tab

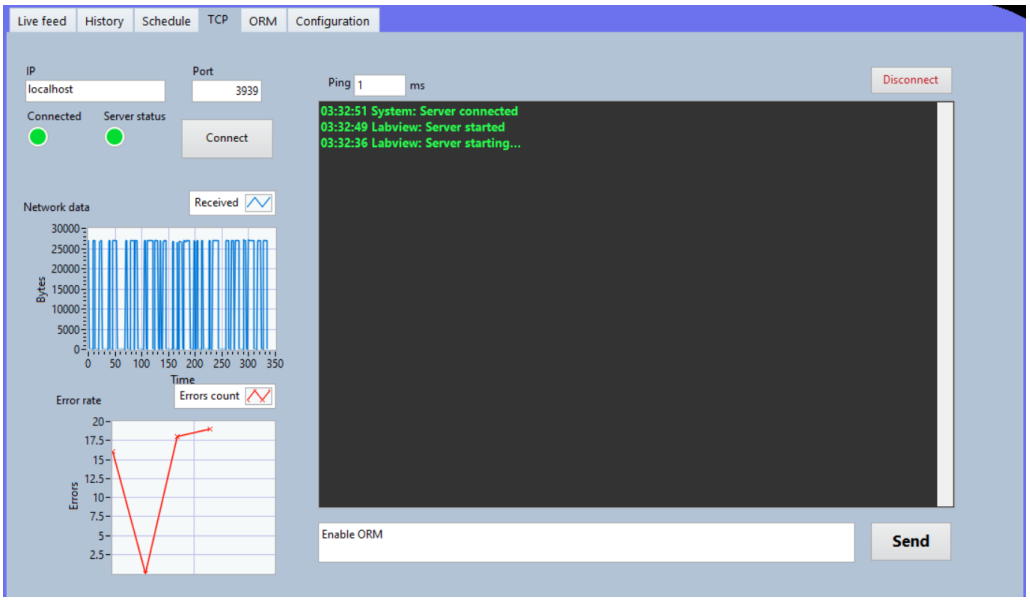


Figure 12. TCP in action

TCP tab shows TCP connection status between Python server and Labview.

2.1.1.6. ORM tab

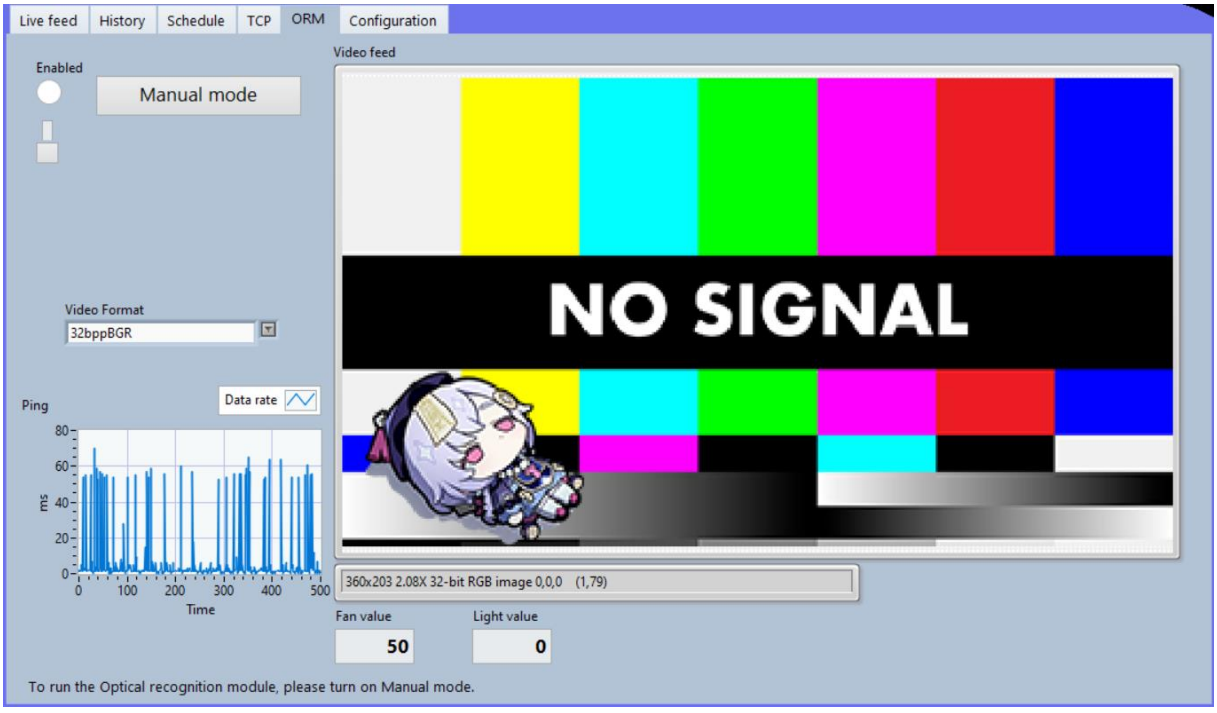


Figure 13. ORM no signal

ORM tab show MJPEG video stream from python server through ’OMF’ command. Fan value and Light value is from server through ’OMV’ command.

2.1.1.7. Configuration tab

Configuration tabcontain all setting available:

- DAQ device and PIR input terminal select
- PIR threshold, time before Fan turn off. etc.

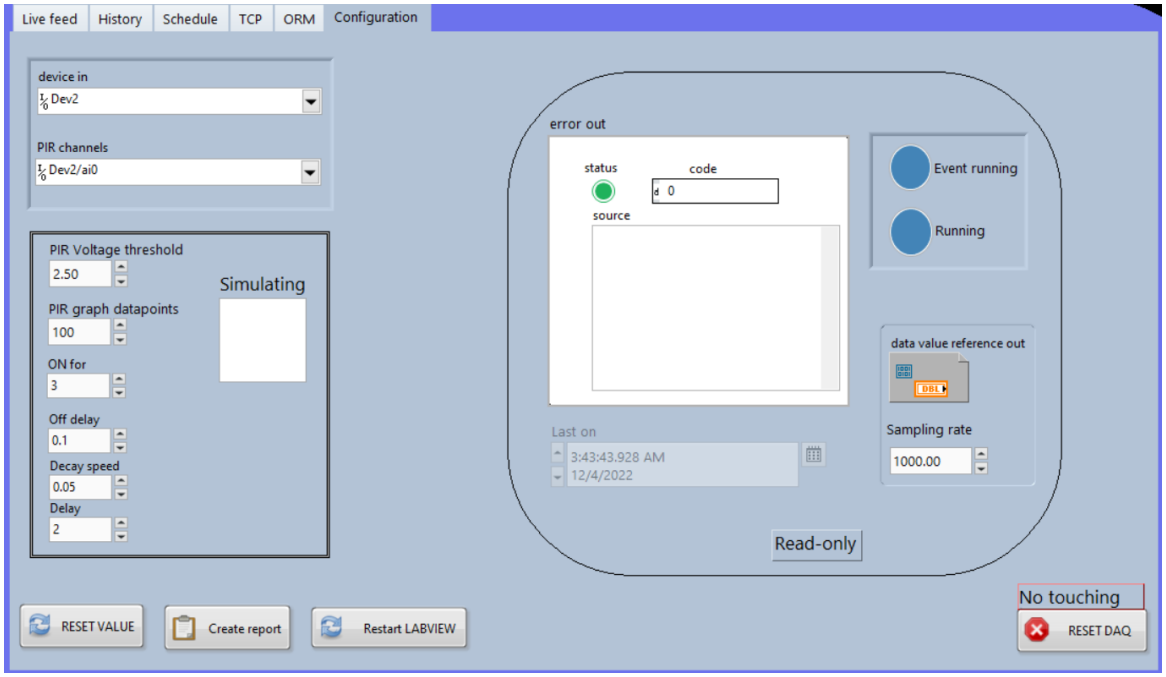


Figure 14. Configuration tab – default value

2.1.2. Labview block diagram

2.1.2.1. Program structure

This project contains 1 main VI and 21 subVIs, along with 7 typedef files and 1 global variable file.

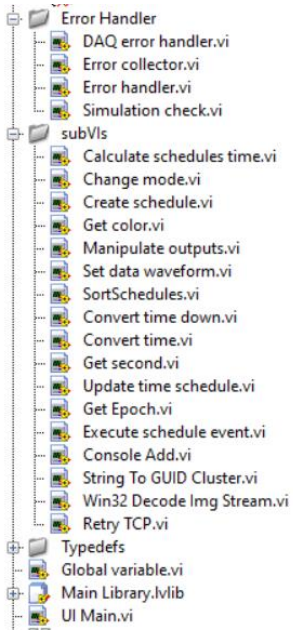


Figure 16.Main VI, subVIs list

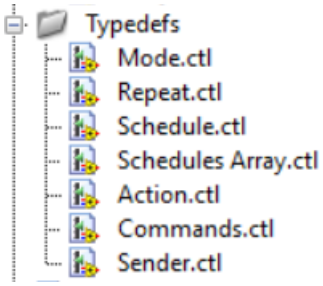


Figure 15. Typedef list

2.1.2.2. Main VI block diagram

We can break the block diagram into 2 parts for better explanation: Main block (2) and an event handler (1).

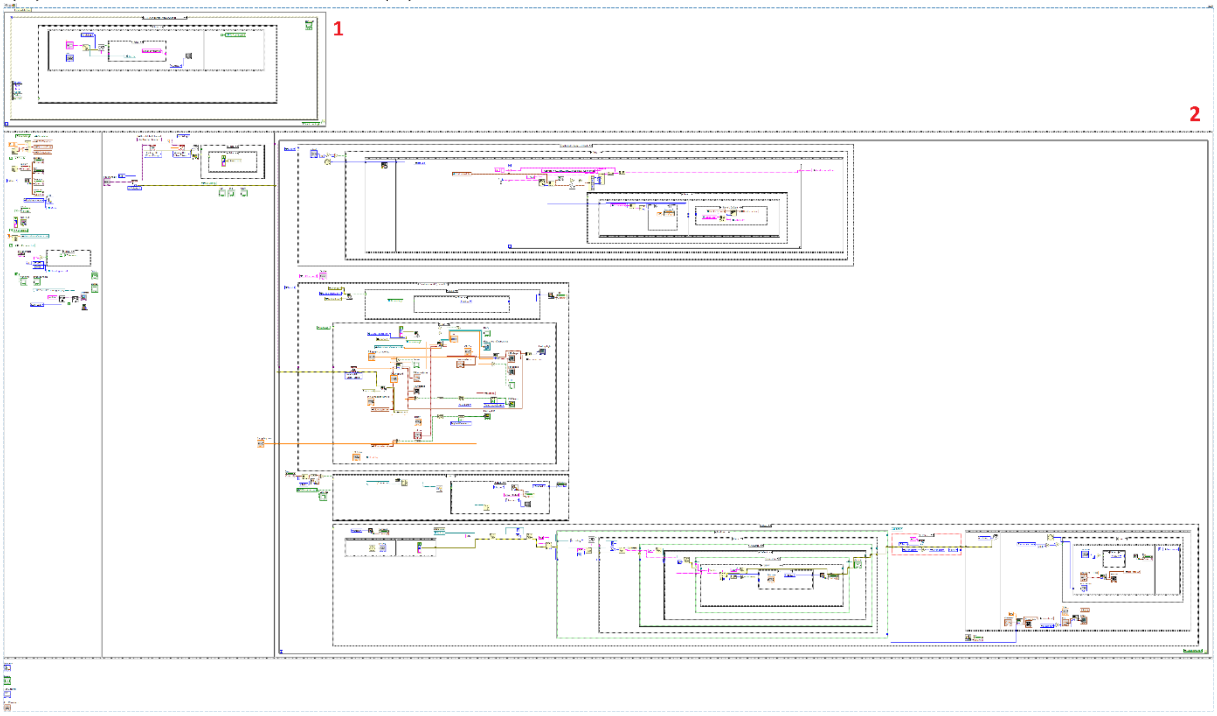


Figure 17. Main UI block diagram

2.1.2.2.1. Main block

The main program is Flat sequence structure. It contains 3 frames which execute in order from left to right.

2.1.2.2.1.1. Setup frame

The first frame to set up all required variables for each run. This frame will run each time the user presses the Run button.

Variable include Button, Indicator color, Global ignore error list code, Data value ref, Initial video feed image.

This also set the default mode to Realtime mode.

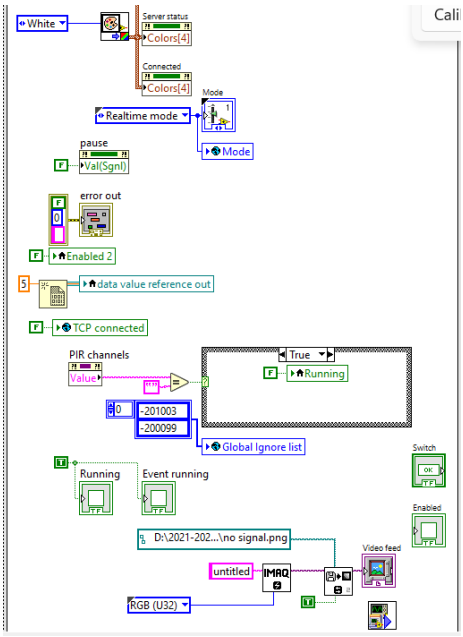


Figure 18. A part of setup frame

2.1.2.2.1.2. Simulation mode handling

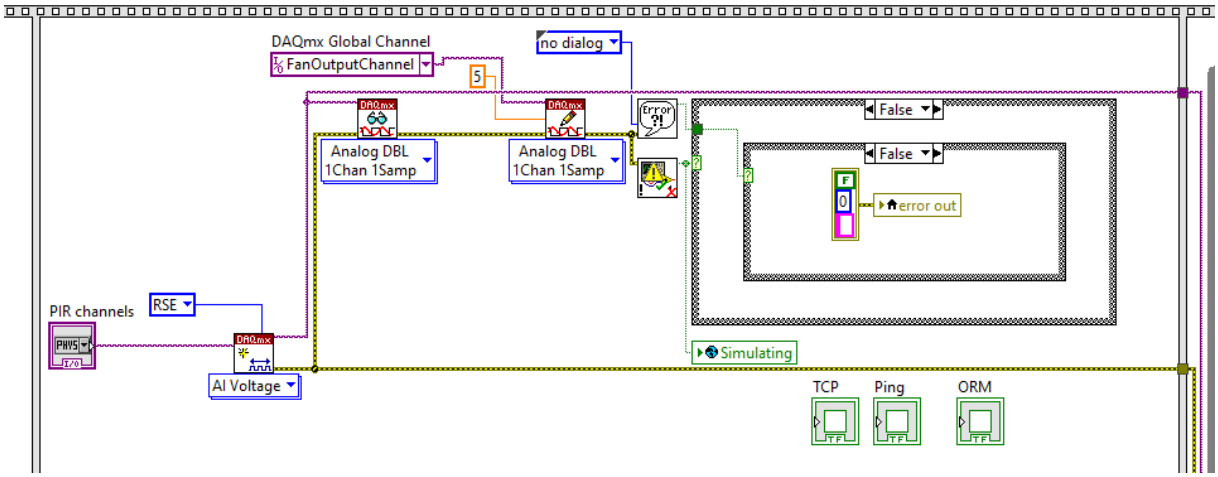


Figure 19. Frame 2

The second frame is to check DAQ connection and determine whether to run in simulation mode or not: It tries to send a signal to turn on FLED to the DAQ, the error out from DAQ write wired to simulation check to handle the error.[1]

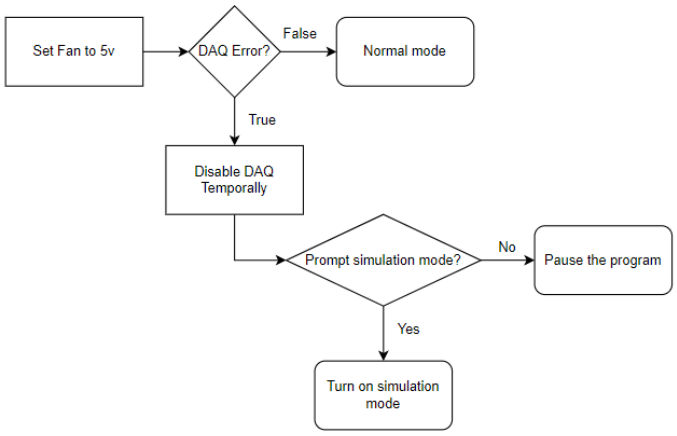


Diagram 1. Simulation mode

[1] For more info see section 2.1.3.3

2.1.2.2.1.3. Program main loop

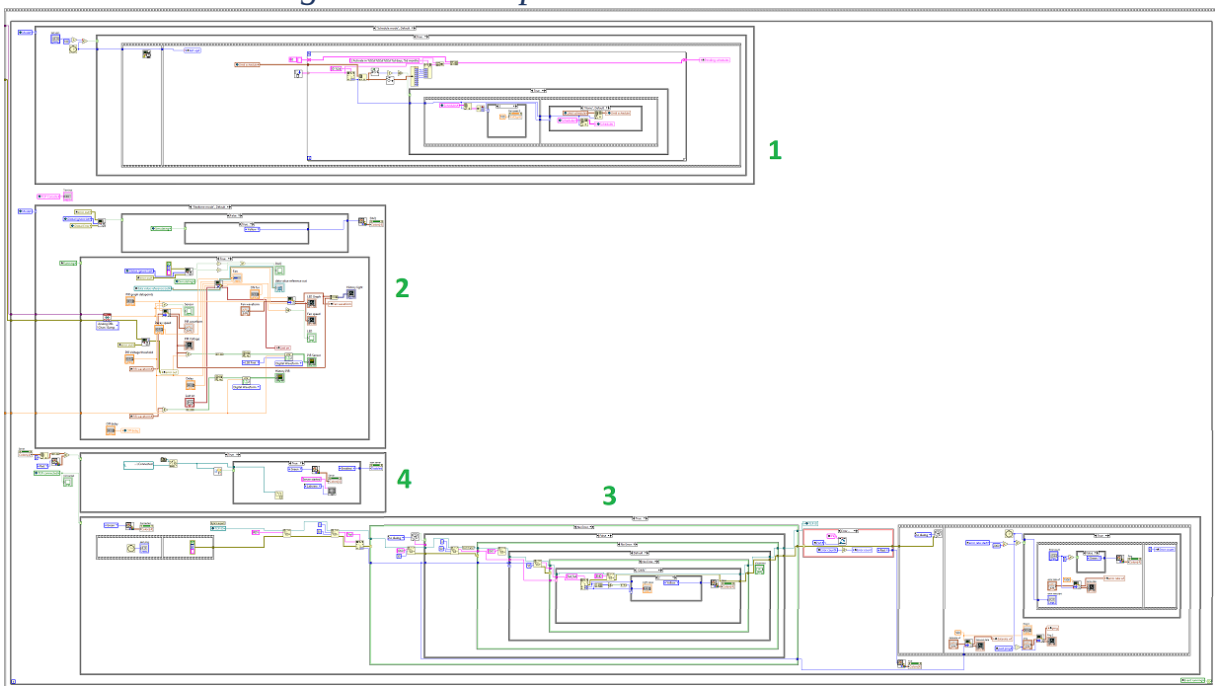


Figure 20. Main while loop

Inside the last frame is 4 case structure, when the user run the program, the first 3 case will execute in parallel with each other:

Case (1) is for computing and process schedule.

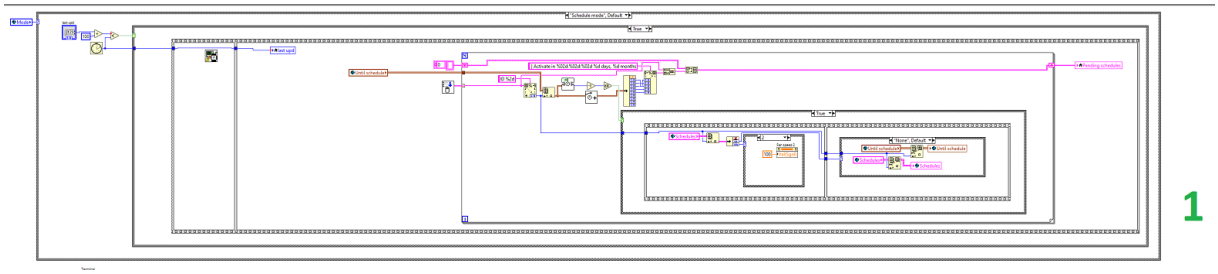


Figure 21. Case (1) - Schedule

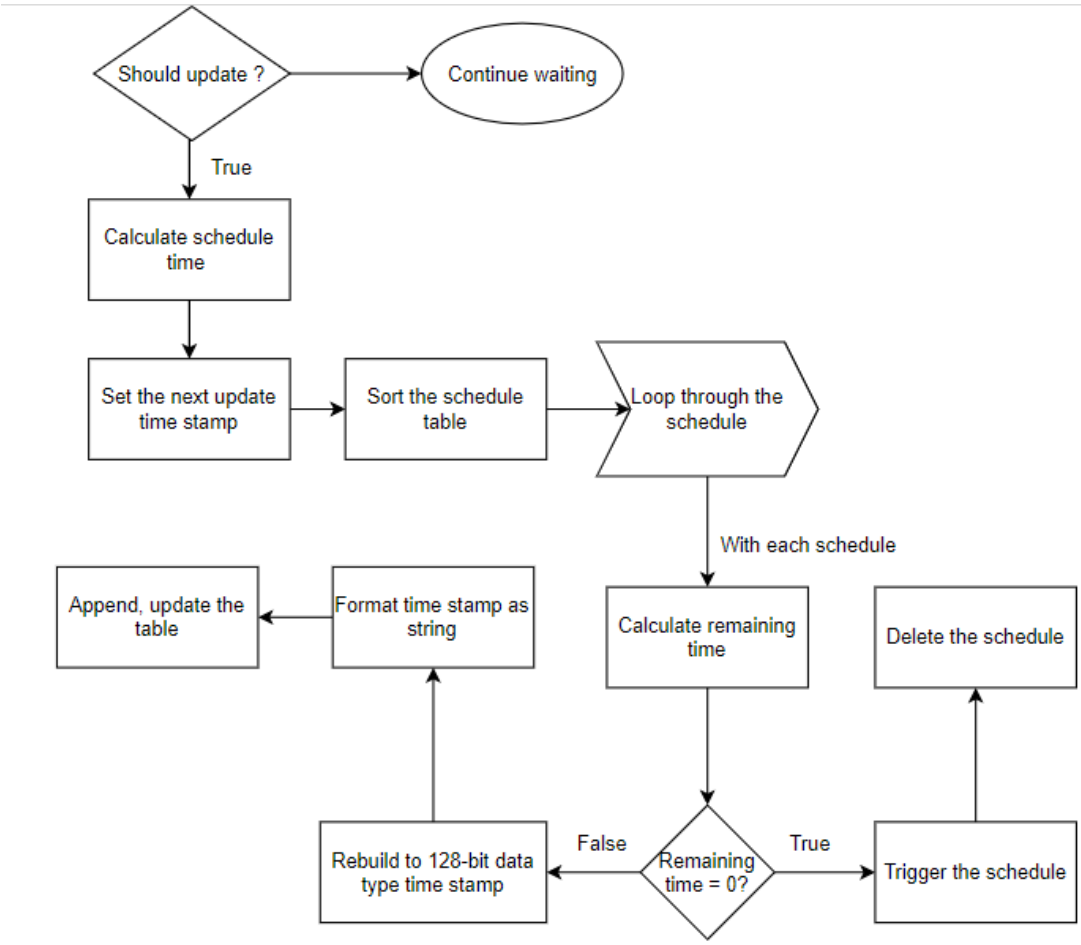


Diagram 2. Update schedule

Case (2) read the sensor data continuously:

- The data from DAQ was then sent to 3 independent sections: Manipulate Output sub VI[1], show waveform on Live feed tab, save data on history tab.
- Manipulate sub VI handle everything needed to control light and fan.
- Set waveform sub VI create and combine data to show as waveform on Live feed tab.

[1] See section 2.1.3.11 for more details

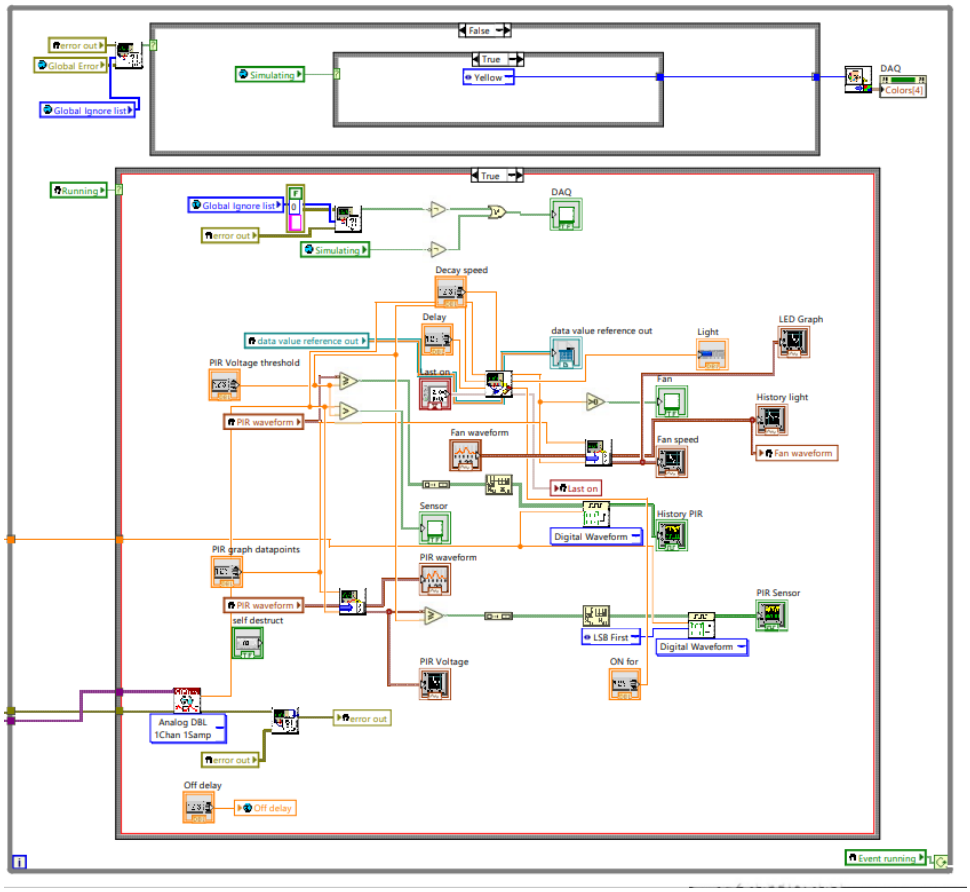


Figure 22. Realtime mode

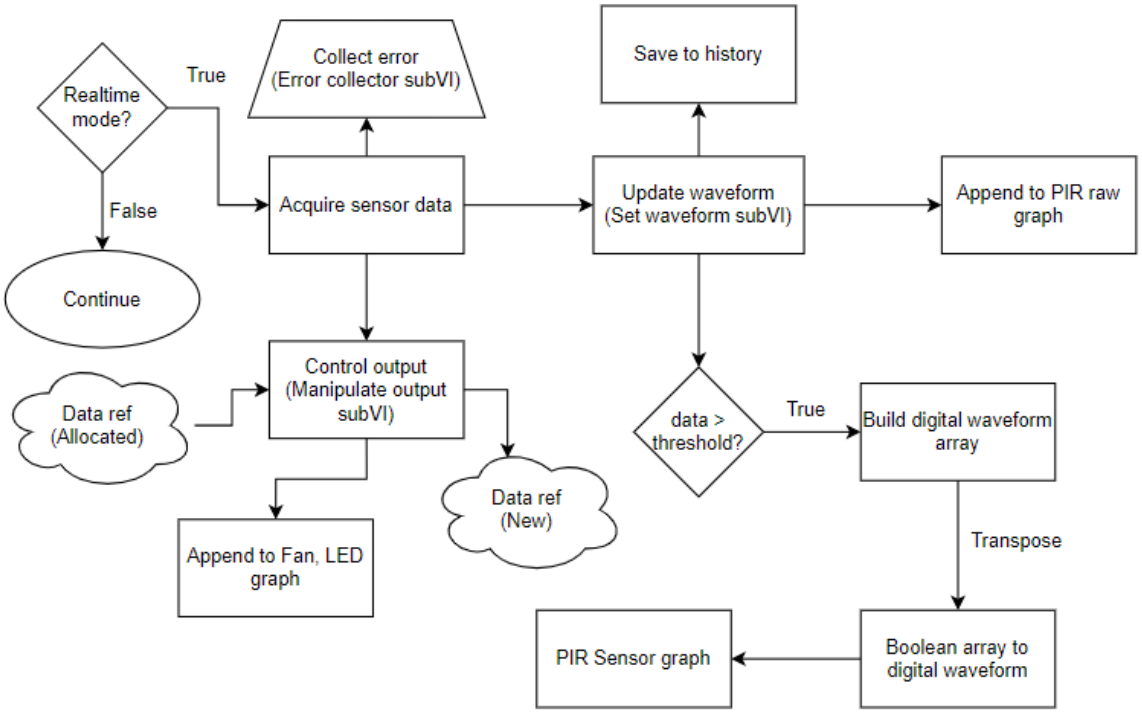


Diagram 3. Realtime data acquisition

Case (3) is TCP client data handling block. This case controls all TCP data in and out from the server to Labview client.

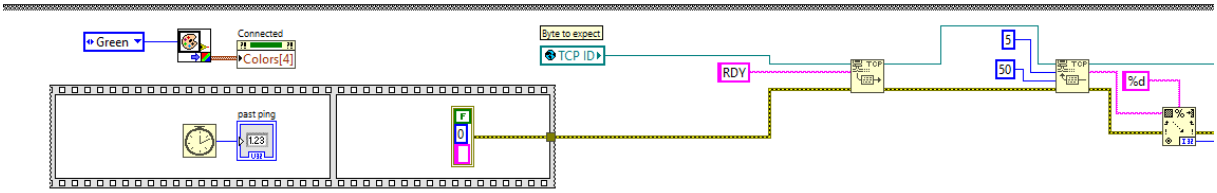


Figure 23. TCP ready state

Each time we receive data from the server, we need to check for error in connection and data integrity. After verifying the data, OK code is sent back to the server.[1]

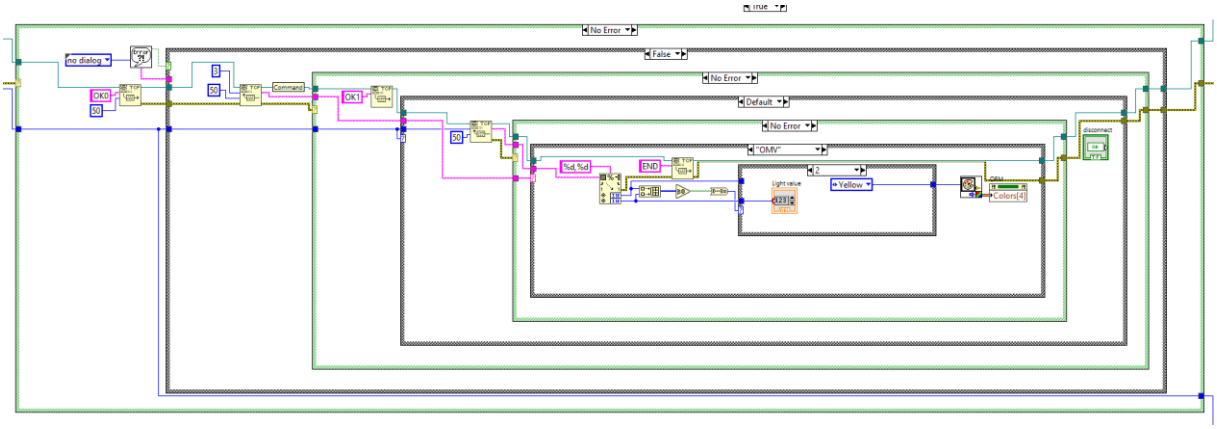


Figure 24. TCP data filter, classify

If we discover any error with the connection or data doesn't match, TCP connection is restarted.

[1] See command structure – section 2.2.5.1

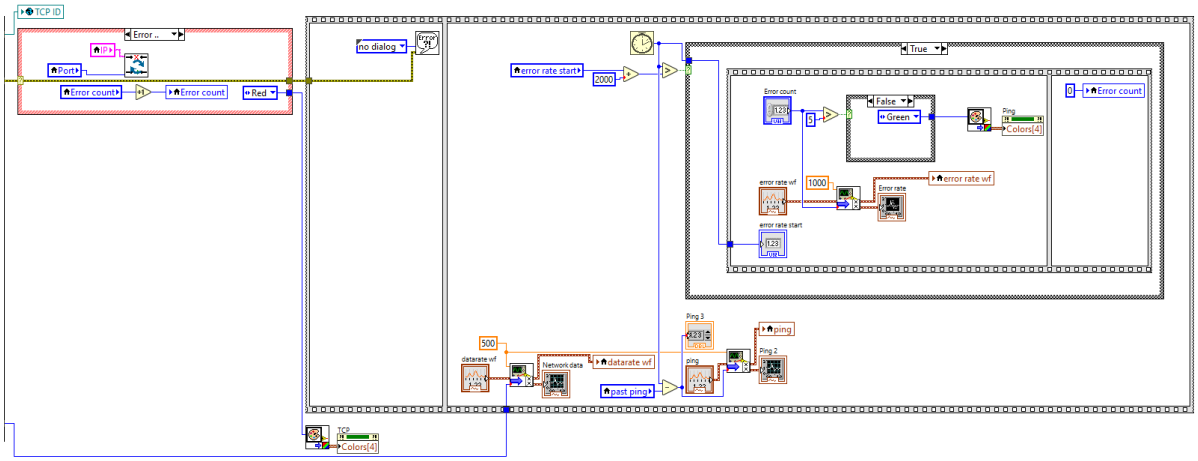


Figure 25. TCP error handing, Network stats acquire

Ping graph is calculated by subtracting the time when command ends with the command start time. Error rate is the time connection restart per 2000ms.

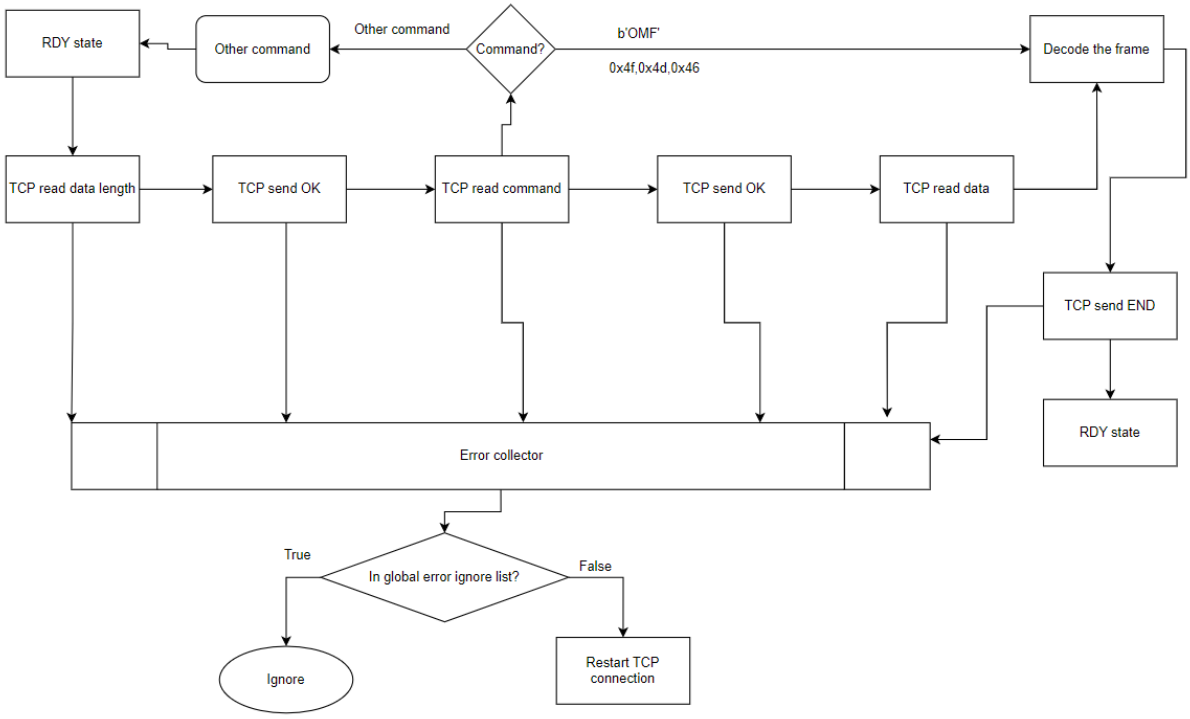


Figure 26. TCP client - simplified

TCP restarting procedure:

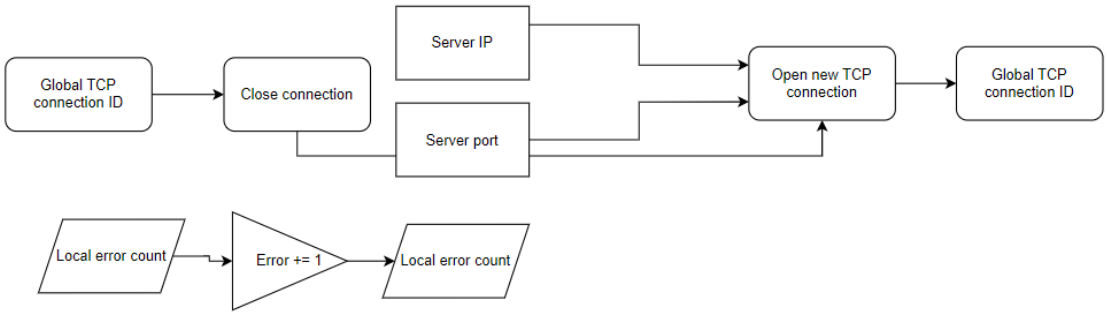


Diagram 4. TCP restart

Case (4) is for checking server started state.

2.1.2.2.2. Event block

The event block contains 20 Event:

1. Save to file:
   Write data from history tab to a file
2. Load file:
   Read data from file and display to history tab
3. Stop:
   Stop the program
4. Reset value:
   Using VI server Invoke node to call Default Vals.Reinit All
5. Restart VI:
   Call App.restart Invoke node
6. Simulating:
   Set the local simulating state to global
7. Start:
   Recover from Pause state by set Running to True
8. Pause:
   Temporary pause the program by set Running to false
9. Manual mode resume:
   Change current mode to Manual mode and enable manual control
10. Connect:
    Get the current control IP, Port and attempt to connect to TCP server
11. Start server:
    Call python PythonApplication.py in cmd using System Exec.vi
12. Manual mode:

Change the manual in ORM tab

13. Fan speed, LED value change (joint event):

    Manually change the FLED output.

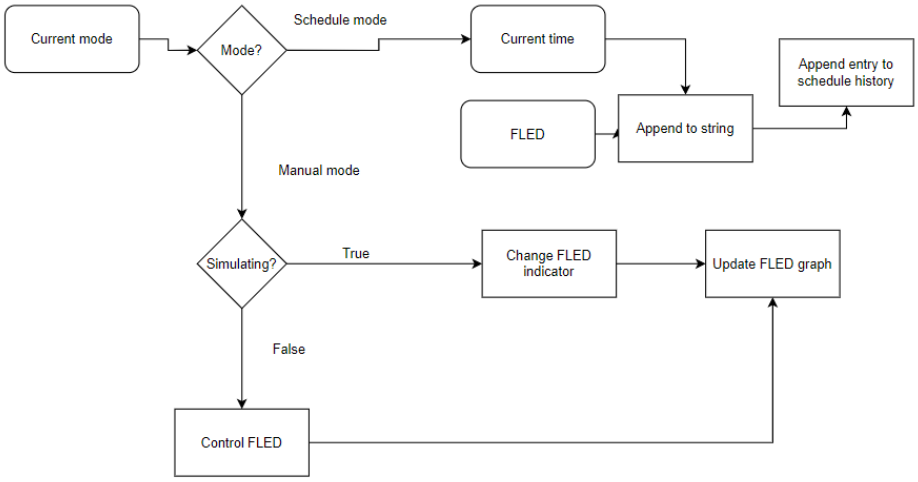


Diagram 5. FLED value change

14. Clear schedule:

    Clear the pending schedule

15. Set schedule:

    Create a new schedule

16. Enable schedule:

    Change program to schedule mode

17. Schedule LED:

    Change the Schedule indicator on control panel

18. Reset device:

    Reset the DAQ programmatically

19. Remove schedule:

    Remove the schedule with ID

20. ID value change:

    Update the Change schedule value when change ID control.

2.1.3. SubVIs

In our project, SubVIs play an **essential role** which make program up and running perfectly, 60% of Labview our code is in SubVIs.

2.1.3.1. Error collector

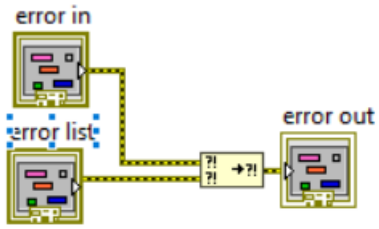


Figure 27. Error collector

Error collector subVI is used to collect error for Error handler subVI.

2.1.3.2. Error handler

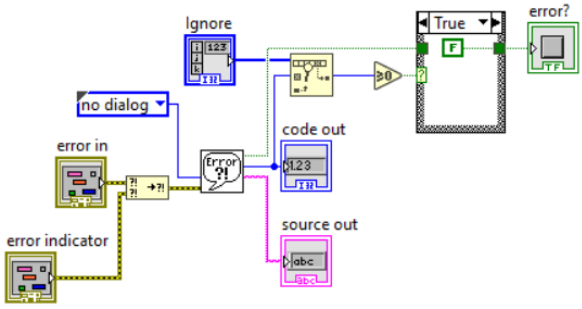


Figure 28. Error handler

Error handler print out error and ignore error in Ignore error global list

2.1.3.3. Simulation check

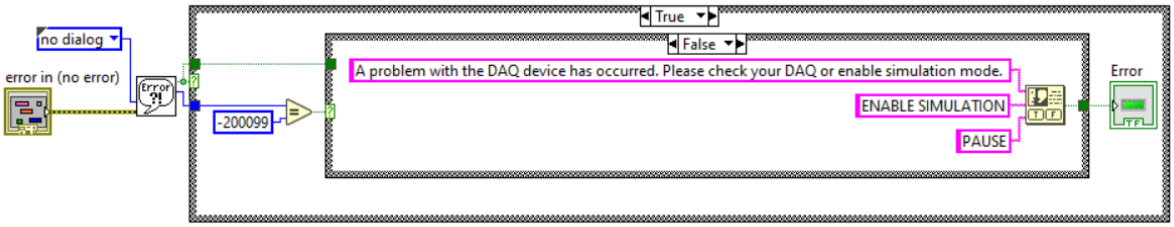


Figure 29. Simulation check

This subVI check and prompts the user to turn on simulation mode or not.

Simulation mode is used when DAQ device is not available, this requires simulated USB-6255 device in NI MAX.

2.1.3.4. Retry TCP

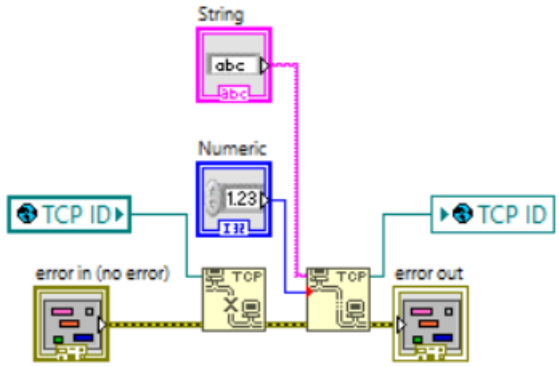
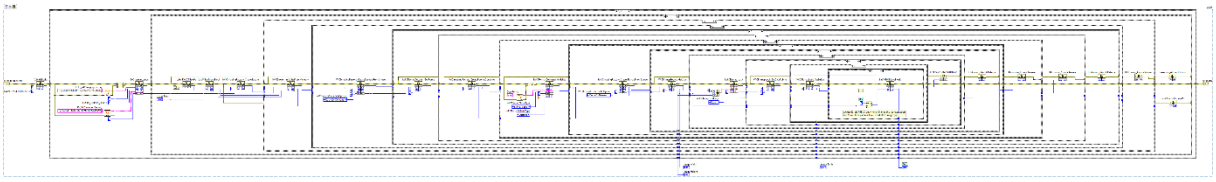


Figure 30. Retry TCP

This subVI restarts the TCP connection.

2.1.3.5. Win32 decode IMG stream



Decode JPEG string from TCP stream using CLFN blocks (ole32, windowscodecs and shlwapi calls)

2.1.3.6. String to GUID Cluster

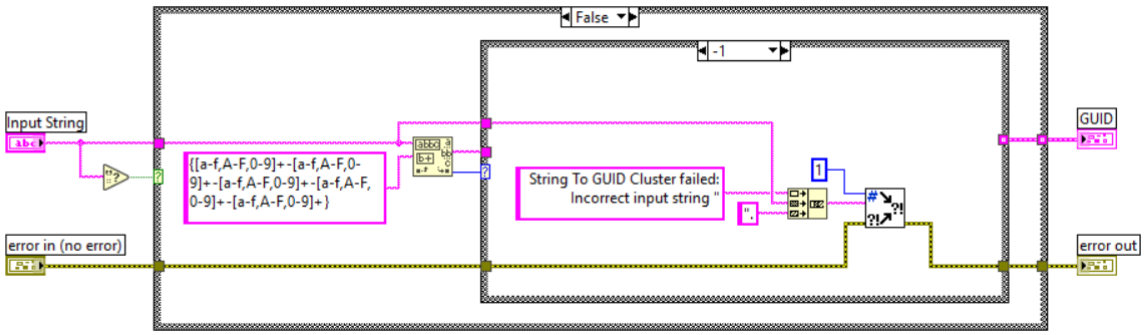


Figure 31. GUID

Part of Win32 decodes IMG stream, convert string to GUID cluster.

2.1.3.7. Console Add

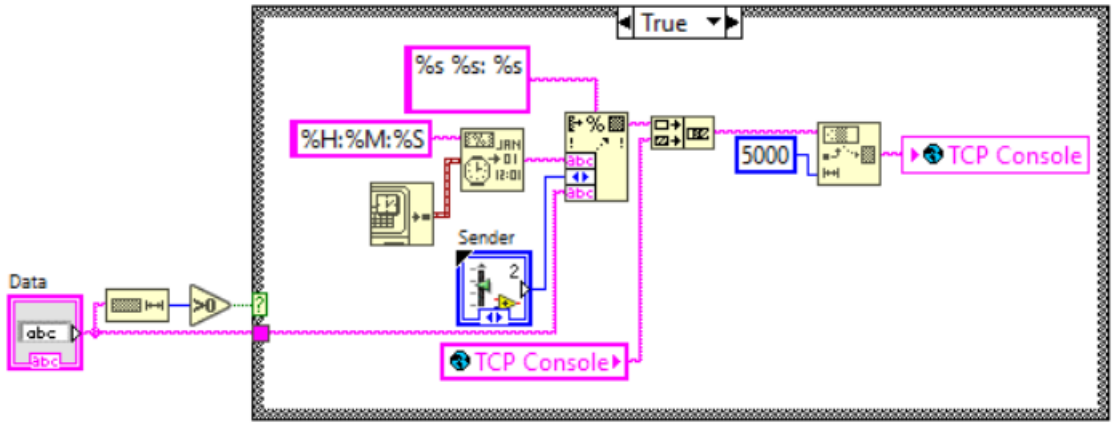


Figure 32. Console

Format the console input and display it in TCP tab, similar to print() function.

2.1.3.8. Get color

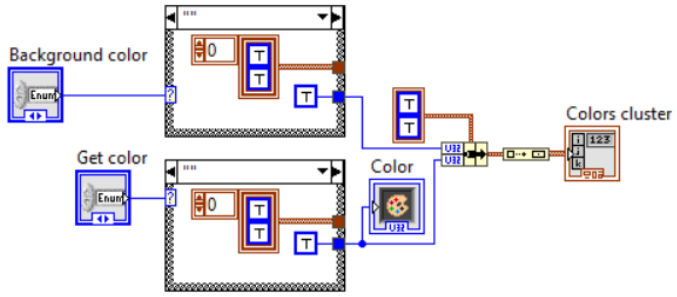


Figure 33. Get color

Get color cluster from defined color Enum, build into array for easy Property node color change.

2.1.3.9. Change mode

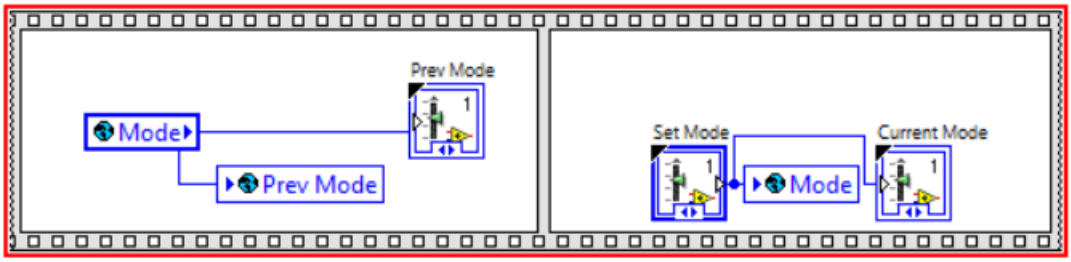


Figure 34. Change mode

Change the current mode to Set mode and Previous mode to current mode.

2.1.3.10. Set data waveform

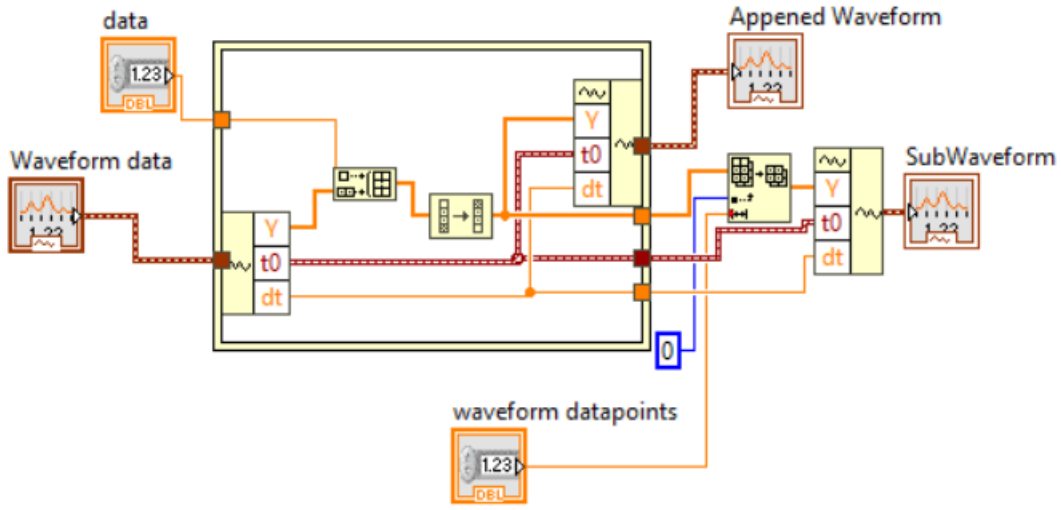
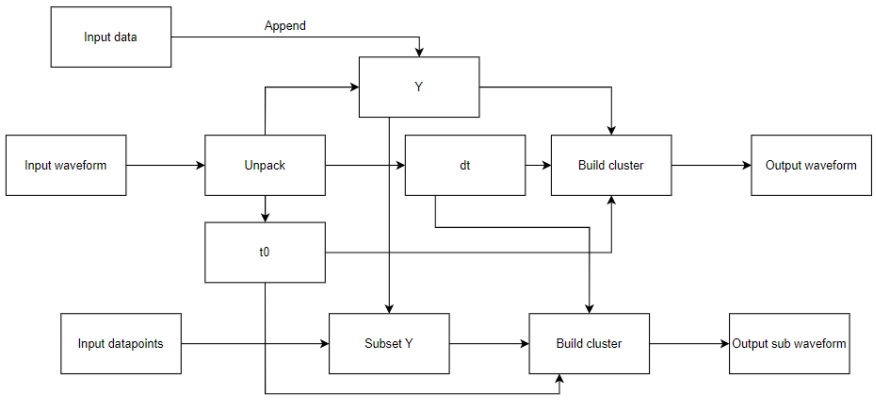


Figure 35. Set data waveform

This subVI gets the input waveform, append it with input data and return appended waveform and a sub waveform that contains the first [waveform datapoints] value.



2.1.3.11. Manipulate output subVI

This subVI control all write operations to the DAQ, it processes data from PIR sensor and calculates time to turn FLED on/off, speed, how long it stays on, voltage to write.

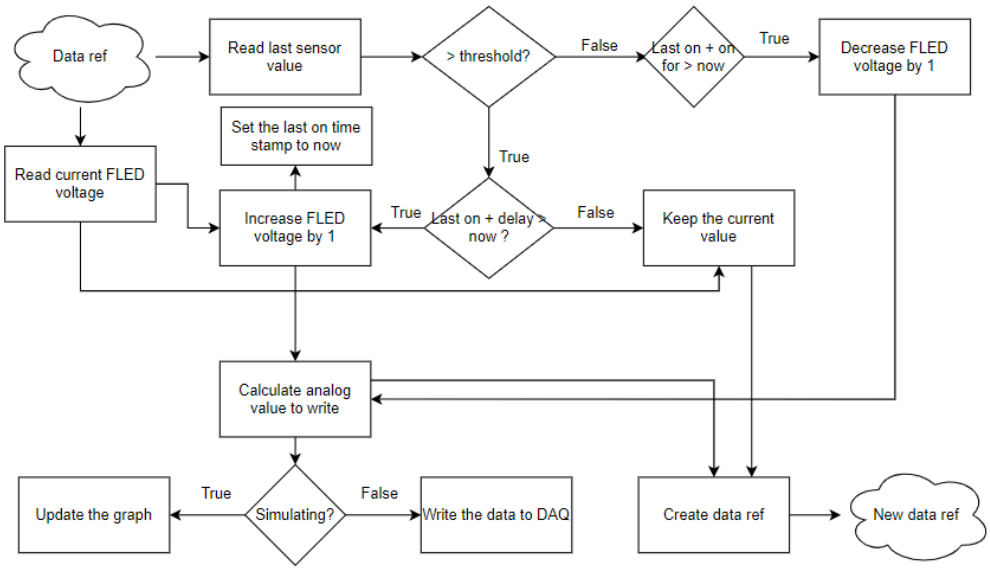


Diagram 6. Manipulate output subVI - simplified

2.1.3.12. Create schedule

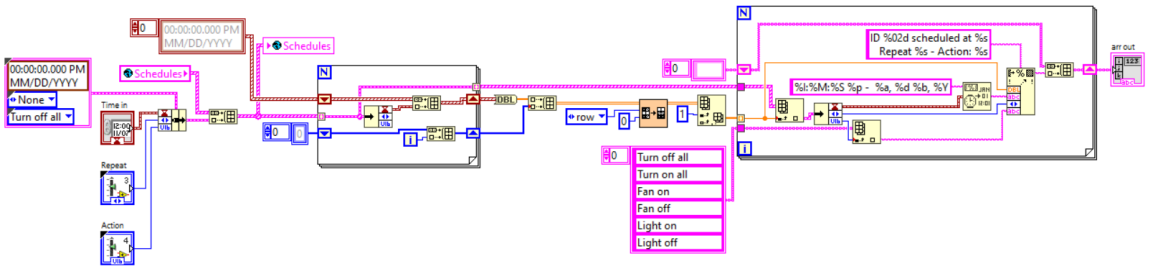


Figure 36. Create schedule

Create a new schedule, add to global schedules list, and sort it.

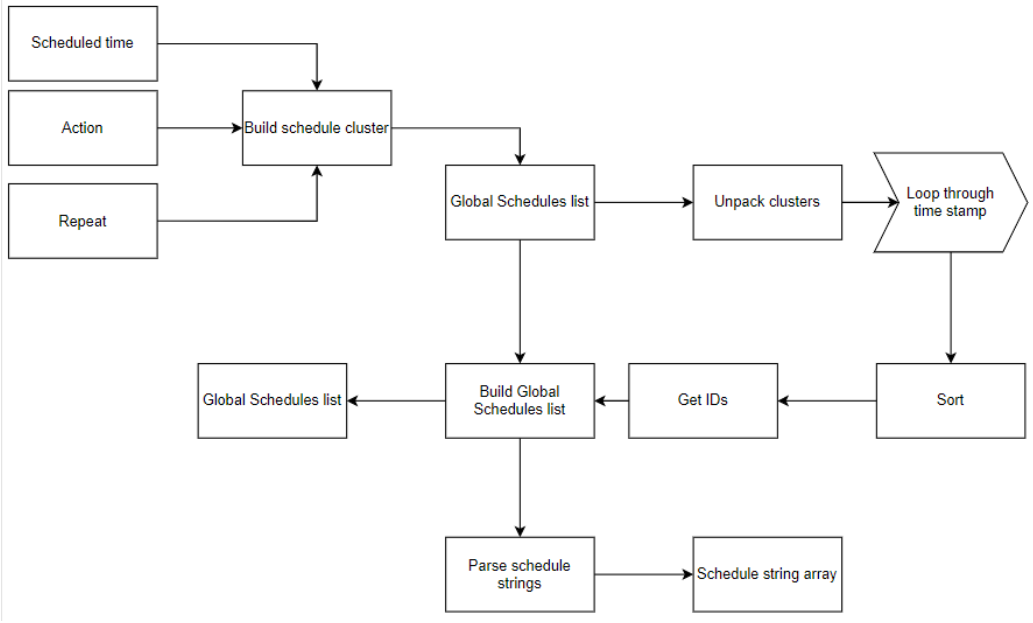


Diagram 7

2.1.3.13. Sort schedule

Similar to create schedule subVI, but only loop through the global schedule list and sort it.

2.1.3.14. Get time Epoch

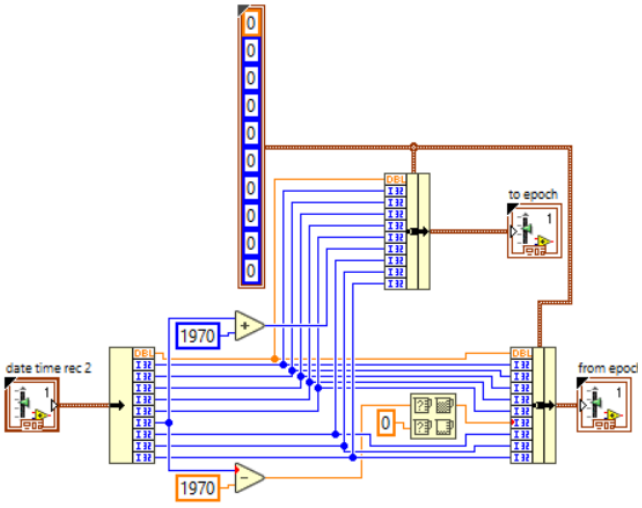


Figure 37. Epoch

When converting Labview 128-bit time stamp data to LVDataTimeRec type, because LabVIEW timestamp significant 64 bits interpreted as a 64-bit signed two's complement integer represents the number of whole seconds after the Epoch, while LVDataTimeRec is a cluster contain time from year 0. So, to get the correct time stamp we need to convert to Epoch. This subVI return 2 LVDataTimeRec: To Epoch and From Epoch.[1]

[1] https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YKVgCAO&l=en-VN

2.1.3.15. Convert time & convert time down

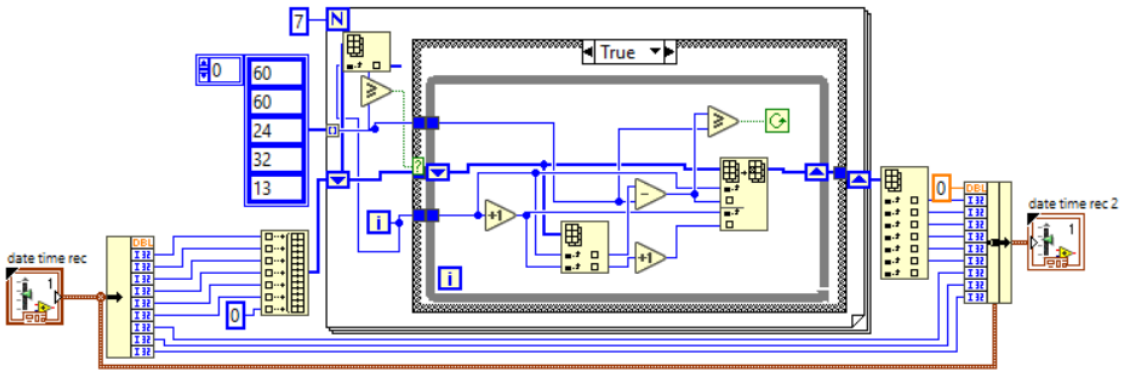


Figure 38. Convert time

This subVI check and converts the LVDataTimeRec value. For example, 62 seconds will be converted to 1 minute 2 seconds.

LVDataTimeRec will be unpacked and converted into an array of 7 elements, with the last element is 0. Then we loop through all element and with each element: compare the value with a pre-defined constant array containing maximum value allowed on each element (in order: second, minute, hour, day of month, month). And if the element exceeds this value, a while loop will subtract the element and increase the next element value by 1. This continues till the end of list.[1]

2.2. Python Application

2.2.1. Purpose

In this project, we used Python as a server to run the optical recognition module and send MJPEG video stream to LabVIEW.

2.2.2. Environment

We use Python 3.9 (64-bit) on windows 11 machine with 1280x720 30 fps integrated camera.

CPU: Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz, 2496 MHz, 4 Core(s), 8 Logical Processor(s)

Total Physical Memory: 11.9 GB

TensorFlow is configurated to run on: NVDIA GeForce GTX1650ti Integrated card.

2.2.3. Script structure

The python script can be described as follows:

[1] The last 6 least important subVI will not be included in this report. For more information, see the source code.

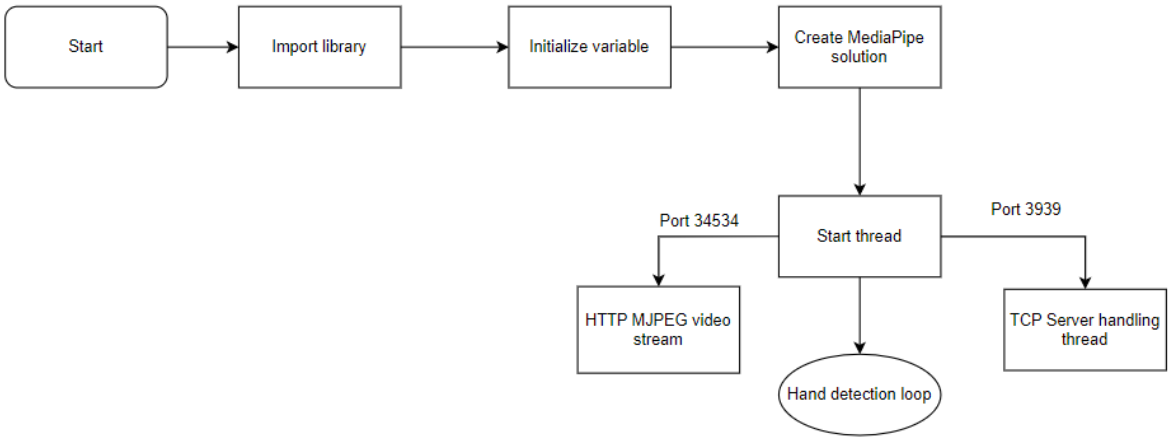


Diagram 8. Script diagram

Using threading built-in module, we created 2 threads running in parallel with the hand detection loop to handle the network connection.

2.2.4. The Optical recognition module (ORM)

The optical recognition module refers to detect, track and FLED value converting function in the server and client. On the server, OpenCV is used to capture the video stream from the camera. Then we apply Media Pipe solutions to acquire the hand tracking data and determine if it is in control range or not. If a hand is detected, then we draw its position on to the frame and compute the FLED value. After that, send it to the client video feed.

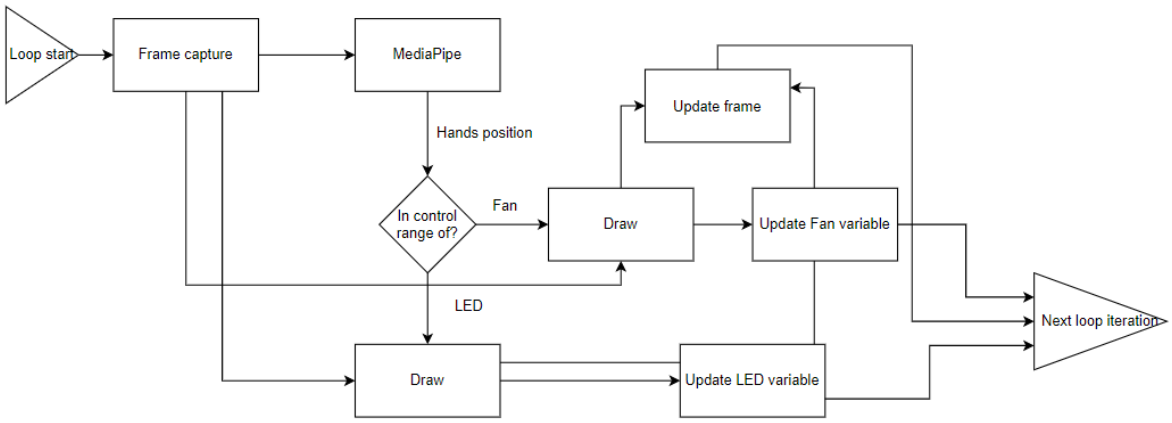


Diagram 9. ORM module

2.2.5. TCP & Video streaming

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol. Therefore, the entire suite is commonly referred to as TCP/IP.

Because LabVIEW doesn't officially support video streaming from some external source such as MJPEG, we have to build our own command system based on TCP.

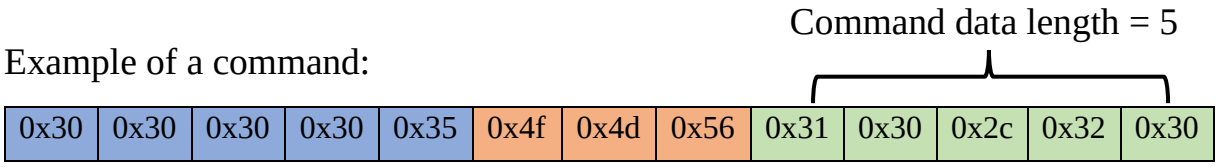
2.2.5.1. Command structure

To send the command between Python server and LabVIEW client back and forth, we implemented a 3 stages command:

Command data length	Command	Command data

Each command sent from server will have structure like above. Command data length is 5 bytes of ASCII characters representing the length of Command data in bytes; Command is a 3 bytes of ASCII characters representing the command for the client; Command data is the data that comes with the Command.

Example of a command:



We can break the data down:

Representation	Command data length	Command	Command data
Hex	0x30,0x30,0x30,0x30,0x35	0x4f, 0x4d, 0x56	0x31, 0x30, 0x2c, 0x32, 0x30
ASCII string	00005	OMV	10,20

This command control FLED value of client, set Fan value to 10 and LED value to 20.[1]

The timed-out event on LabVIEW is set to 50ms, so it's important to make sure that the command is sent within this time.

Between each stage, make sure to wait for LabVIEW to respond before sending the next instructions.

[1] See appendix A for command table and respond table

2.2.5.2. Video encoding

In order to increase performance, the original frame 1280x720 is downscaled to 360x203 pixels. Then the frame is encoded into JPEG format and stored in server memory.

Diagram 10. Video encoding

2.2.5.3. TCP server handling

The TCP server runs in parallel with the ORM. Default, it's continuously sent 2 commands repeatedly to the client: OMF command and OMV command to update the ORM in client side when connected.

The server also implemented a watchdog timer to retry connection to client when timed out.

Section 4: RESULT

1. Running the program

To run the project, first open the *Project S.S.I.C.F.L.lvproj* Project file and navigate to UI Main.vi file. Please make sure to run the project on LabVIEW 20.0 32-bit version[1] with DAQmx 20.0, NI-IMAQdx 20.0 and Vision Development Module 2020 installed.

Default, the user interface will be similar to user interface on Figure 6.

1.1. Realtime mode

To run the program in real-time mode, make sure USB-6255 device is connected and configurated in NI MAX. Next, navigate to Configuration tab and select *device in* and *PIR channel* (See Section 2, part 2.1.1.7) then click on Run (Ctrl + R), the program will automatically start in real-time mode.

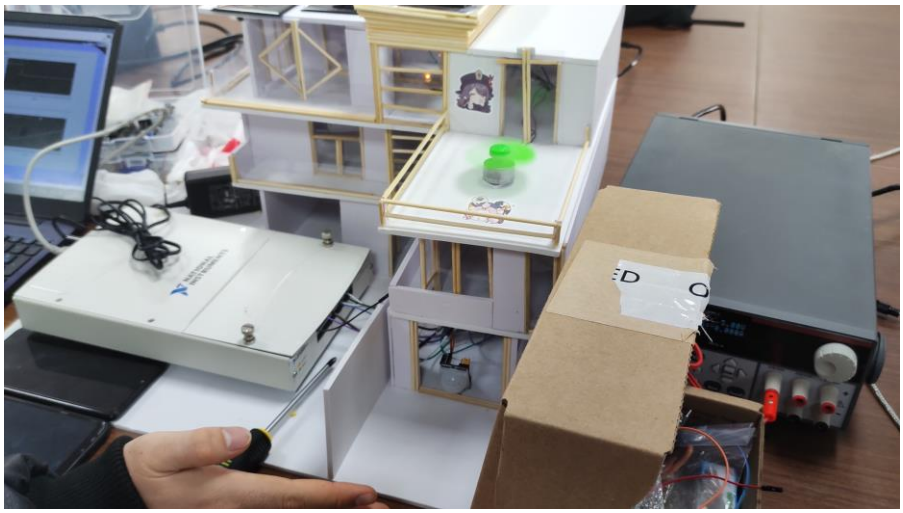


Figure 39. Realtime mode

1.2. Manual mode

While on real-time mode, you can switch to manual mode by clicking on *OVERWRITE/RESUME* button. When manual mode is enabled, Enable light on the left of *OVERWRITE/RESUME* button will change to green and Status text in control panel will change to *Manual mode*. To turn off manual mode and return to the previous mode, click on the button again.

[1] 18.0 Version also available in branch 18.0 on GitHub repository https://github.com/dtungpka/I.C.F.L/tree/18.0

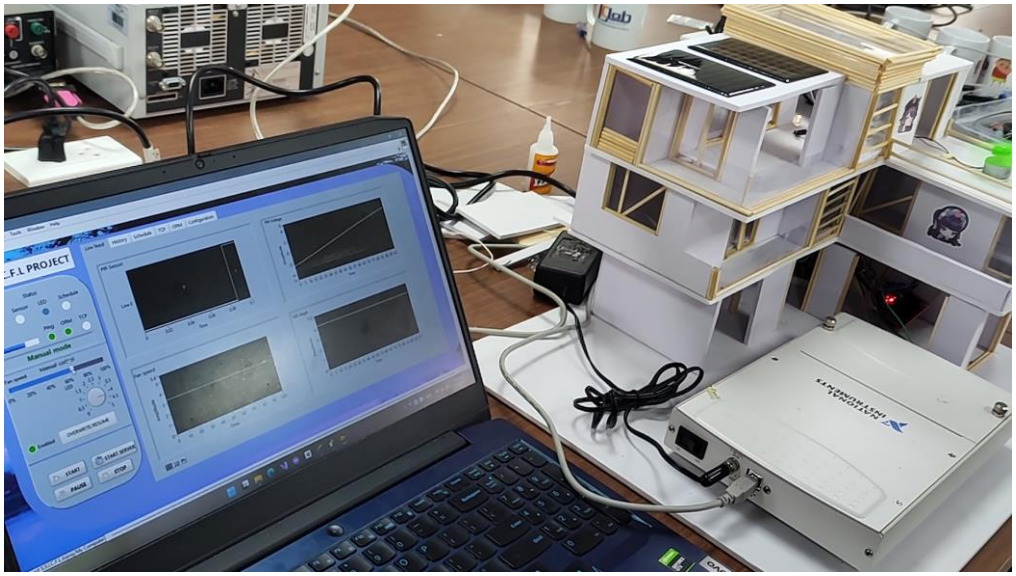


Figure 40. Manual mode in action

1.3. Schedule mode

1.3.1. Create a new schedule

To create a new schedule, navigate to schedule tab, on *Set a new schedule* section, change the time to the time you want to run the action, then choose when it should repeat, what action to run. After that click on the OK button. If you set the repeat to None, the action will be deleted after it runs.

1.3.2. Edit a schedule

Editing a schedule is easy, after you create a schedule, it will appear in the *Pending section*. Each schedule will be given a unique ID, Enter the ID in *Config schedule* section and press Enter. After the *time*, *action* and *restart* inputs are updated correctly you can change it and click on the right OK button to save changes or click on delete button to delete the schedule.

1.3.3. Enable schedule

To enable schedule mode, click on the Enable button on schedule tab. The mode indicator on the control panel will be updated.

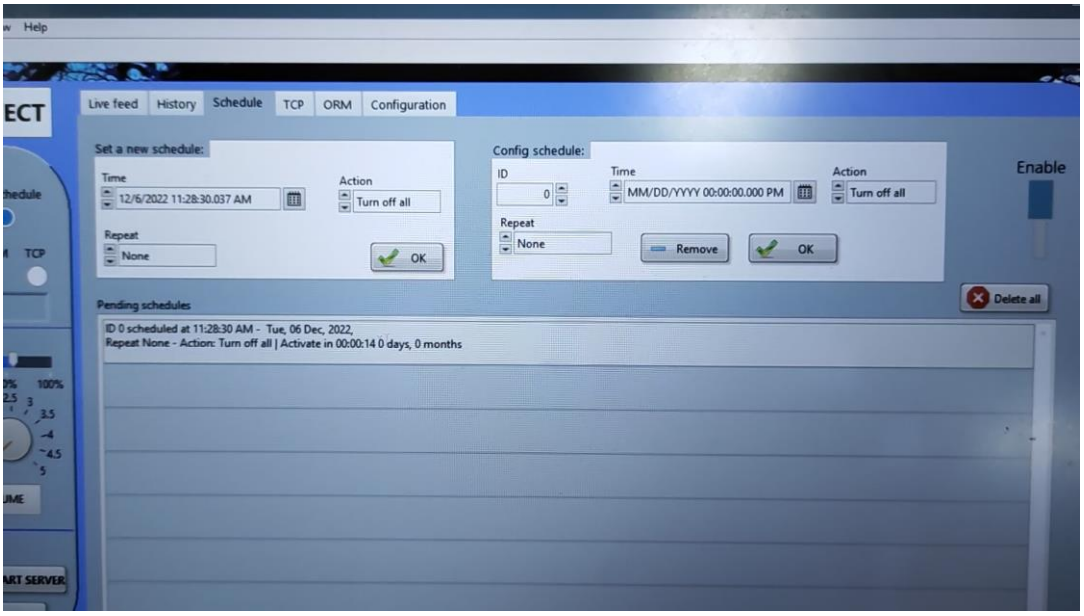


Figure 41. Schedule mode enabled

1.4. Running the ORM

To run the optical recognition module, first you need to run the python server. Make sure you have Python 3.9.x 64-bit installed and the library in requirements.txt file.

After verifying that a correct version of python is installed, click on *START SERVER* to start the python script, then navigate to TCP tab, when Server light indicator changes to green, press on Connect button to connect python with LabVIEW. Then you can navigate to ORM tab to see video feed and value received from the server.

To make data from ORM to have effect on DAQ, press on Manual mode button.

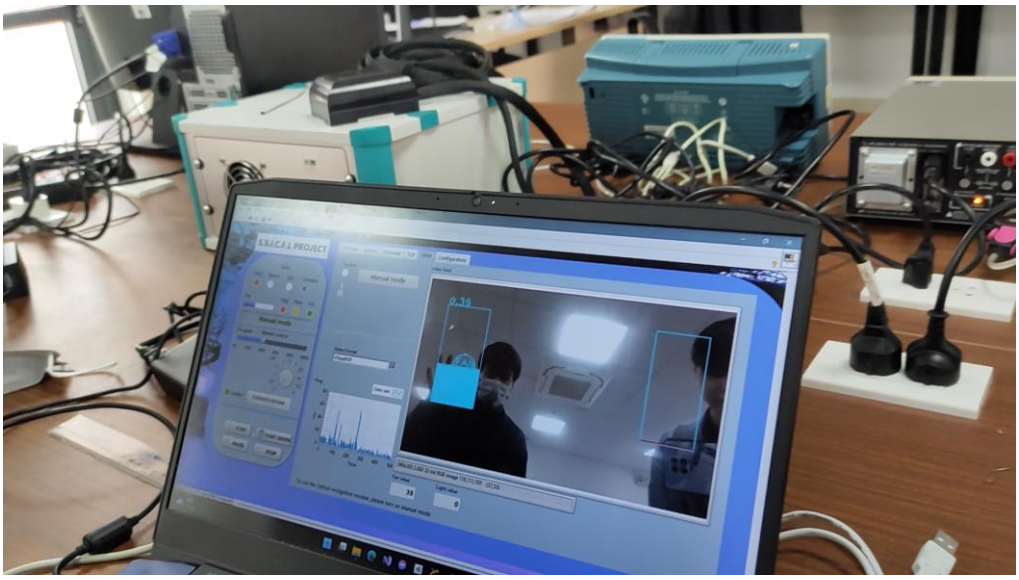


Figure 42. Controlling fan using ORM

2. Performance Analysis

Because of the scale of our project, performance is a very important factor of our program. On LabVIEW, we implemented non-blocking wait method for all function that require timing such as schedule time calculation, Writing-Reading DAQ data, real-time FLED control, etc.

Implementing subVI also an important factor:

For example, on a 17 second session, the main program called Set data waveform.vi 15416 times, which only took 0.2158 second to run. If we wrote this subVI in our main program, this would not only reduce the readability of the code, but also the performance.

On Python server, threading is used to run ORM, server handing in parallel.

Because we are short on time, we couldn't optimize the Python Server much, so the start-up time is somewhat slow (about 11.2 seconds). We could try to optimize this in the future.

Section 5: CONCLUSION

1. Conclusion

After a lot of discussion, planning and programming, we successfully created the S.S.I.C.F.L. The system is up and running perfectly on our machine and controlled LED and Fan smoothly, although the ORM still has some trouble connection and some optimization needs to be done, we still considered this project is finished.

2. Further development

In our original idea, we also thought about a new module, the Google Assistant Controlled module (GAC). This module created a device called FLED (Fan and LED) as another way to control DAQ from smartphone device. We successfully created and connected the module to google cloud and set up another server port for handling request from google. But due to the deadline, we cannot finish it in time.

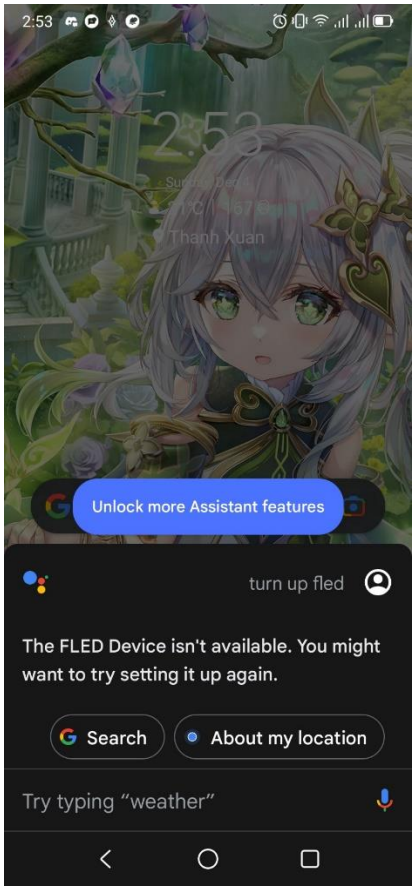


Figure 43. Turning on Led and Light with GAC module

Section 6: REFERENCES

(n.d.). Retrieved from NI: https://www.ni.com/en-vn/support/documentation/supplemental/08/labview-timestamp-overview.html

HC-SR501 PIR Motion detector datasheet. (n.d.). Retrieved from https://www.mpja.com/download/31227sc.pdf

Koon, J. (2019, Feb 13). Retrieved from engineering.com: https://www.engineering.com/story/how-iot-will-change-our-lives

Motor Driver Module-L298N. (n.d.). Retrieved from http://wiki.sunfounder.cc/index.php?title=Motor_Driver_Module-L298N

NI. (n.d.). Retrieved from Get Components of Time Returned by Get Date/Time in Seconds: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YKVgCAO&l=en-VN

NI. (n.d.). *Error 363507 - LabVIEW could not verify the authenticity of the server* . Retrieved from https://forums.ni.com/t5/LabVIEW/Error-363507-LabVIEW-could-not-verify-the-authenticity-of-the/td-p/2881254

NI. (n.d.). *NI 6255 Specifications*. Retrieved from https://www.ni.com/docs/en-US/bundle/pci-pxi-usb-6255-specs/page/specs.html

NI. (n.d.). *Programmatically Reset VI Example*. Retrieved from https://forums.ni.com/t5/Example-Code/Programmatically-Reset-VI-Example/ta-p/3532614

NI. (n.d.). *USB-6255*. Retrieved from https://www.ni.com/en-vn/support/model.usb-6255.html.

NI. (n.d.). *USB-6255*. Retrieved from https://www.ni.com/en-vn/support/model.usb-6255.html

Appendix A. TCP command table

Command to send from server:

Commands	Hex	Description	Command data
LED	0x4c 0x45 0x44	Set the LED value	%d
FAN	0x46 0x41 0x4e	Set the fan value	%d
OMF	0x4f 0x4d 0x46	Send JPEG image stream in memory converted to bytes	%s
SCH	0x53 0x43 0x48	Create a new schedule (HH,MM,SS,dd,mm,yy)	%d,%d,%d,%d,%d,%d
OMV	0x4f 0x4d 0x56	Set the FLED value recognized by optical module (Optical Module Value)	%d,%d

Client responds:

Respond	Hex	Description
RDY	0x52 0x44 0x59	Client ready to receive command
OK0	0x4f 0x4b 0x30	Command data length received, waiting for command
OK1	0x4f 0x4b 0x31	Command received, waiting for command data
END	0x45 0x4e 0x44	Command executed successfully, return to RDY state
ERO	0x45 0x52 0x4f	Data mismatch – Error occurred

Appendix B. LabVIEW status indicator

Status name	Color	Meaning
DAQ	Red	DAQ error
	Yellow	Simulation mode
	Green	DAQ running normally
Sensor	White	-[1]
	Blue	PIR Sensor voltage passed the threshold
LED	White	-
	Blue	LED turned on
Schedule	White	-
	Yellow	Schedule pending but not enabled
	Blue	Schedule enabled
Fan	Blue bar	Fan value indicator
Ping	White	-
	Red	High ping (>60ms)
	Green	Low ping
ORM	White	-
	Yellow	ORM is controlling FLED
	Green	ORM is enabled
TCP	White	-
	Red	TCP error
	Green	TCP is running normally
Enable	White	-
	Yellow	Manual control section value changed, but no effects are applied
	Green	Manual mode enabled

[1] Default, not enabled