VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

PROPERTY MONITORING SYSTEM

Technological details

20th Dec, 2018

Group 3
Nguyen Minh – e1500964
Pham Chinh – e1500962
Do Le – e1400476

CONTENTS

1 INTRODUCTION

1.1 Core Technology

MQTT protocols: MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers. *mqtt.org, MQTT's Official Website. [online] Available at: https://mqtt.org [Accessed 20th Dec. 2018]*

It's the perfect solution for Internet of Things applications. It works in a publish and subscribe manner, a device can publish a message on a topic, or it can be subscribed to a topic to receive messages.

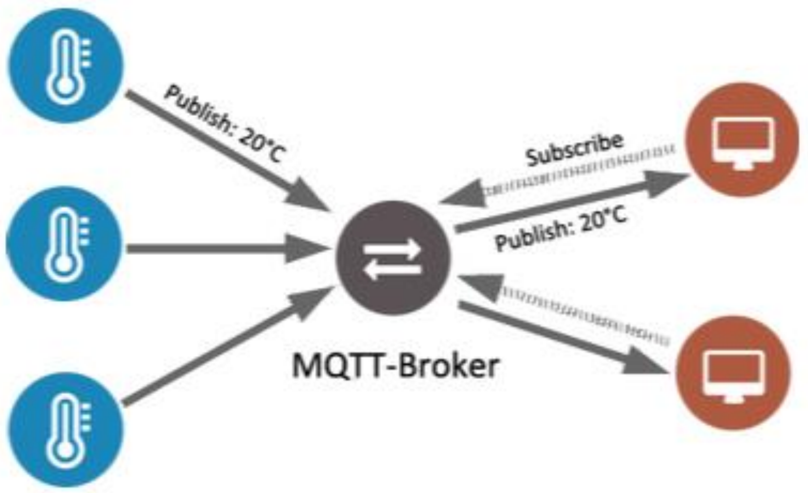

Figure 1: Demonstration of the MQTT protocol

1.2 Overall Application Diagram

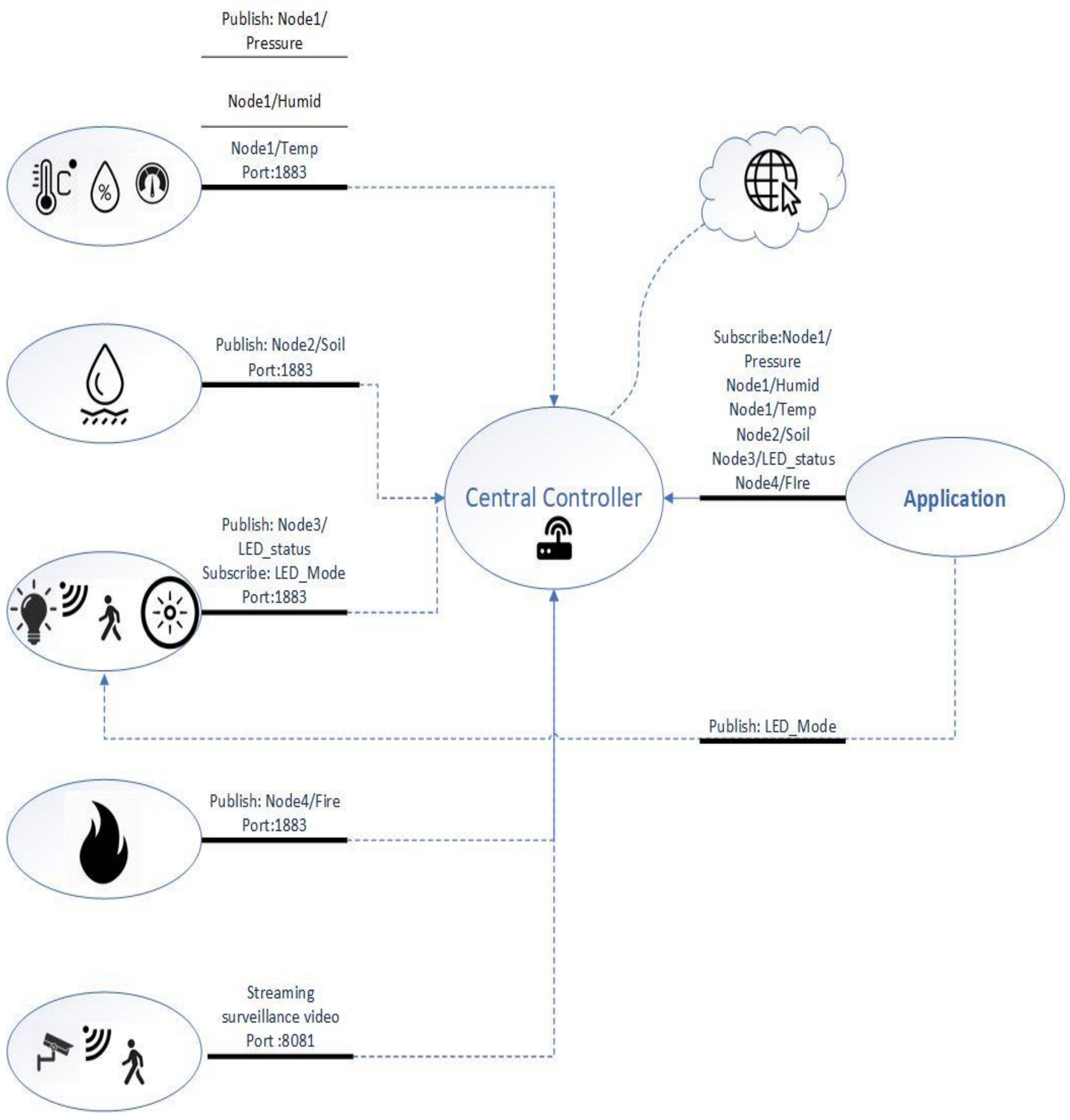

Figure 2: Overall system diagram

1.3 Primary Programming Languages

Python: for developing the application running on the Raspberry Pi.

Arduino – C: For programming the node devices.

1.4 Libraries and Frameworks

1.4.1 GUI:

PyQt5: the main framework to create and design the graphical user interface of the system

PyQtGraph: Library for drawing and visualizing data in graphs

Pandas: Mainly used for conveniently reading data from database

OpenCV: Used for capturing streaming video

1.4.2 Backend:

Paho-MQTT: MQTT Python library for data communications between the main controller and node devices.

Mosquitto-MQTT: MQTT broker for the Raspberry PI gateway.

Time: Python library used for time recording of data and performance testing of different algorithms.

Os: Python library used for file deleting

Glob: Python library used for file handling with paths.

Email + SMPTlib: Python libraries used for sending report email .

1.4.3 Node:

PubSubClient: MQTT library for NODEMCU.

Adafruit_BME280: library for the BME280 temperature, humidity, air pressure sensor.

ESP8266WiFi: library enable the ESP8266's internet connection.

Adafruit_INA219: library for the INA219 sensor.

1.5 Applications:

Motion: Linux application for controlling the Pi camera on Raspberry Pi.

Dnsmasq: Linux application providing the DHCP service for Raspberry Pi.

Hostapd: Linux application allowing Raspberry Pi be configured as an access point.

2 SOFTWARE

2.1 Function Details of Software in Hardware devices.

Setupwifi() : connect ESP8266 NODEMCU to the specified local network (WiPi)

Callback() : check messages and topics sent to and received from the main controller (Broker), in Light control system node, this function is used to check messages of the *mode* and *light state* sent from the Broker to control the LED bulb.

Reconnect() : loop until the client can connect to the MQTT server of the Broker and exit when connected.

Automatic(): in the light control system, this function is called when the Broker sends a signal that enable automatic light control using the node's PIR and photoresistor, how this function works was demonstrated in the above figure.

setupMQTT(): function enable the nodes to connect to the MQTT broker.

setupBME280(): start the BME280 temperature, humidity and air pressure sensor to work, if it is not working, error is output to the Serial Monitor in Arduino IDE.

GasDetect(): defines behaviour of the fire detect system. If the gas sensor 's input is higher than the threshold, the buzzer will start buzzing the alarm.

2.2 GUI

The GUI of the system consists of two different parts: UI files and Python source code files. UI files are created by designing with Qt designer and they can be converted to python source code file using command line. However, it is recommended to keep them as UI files for easier modification later on Qt designer. The python source code files load the UI files and define functions for the graphical user interfaces. Some of them also modify the visual look of the graphic loaded from UI files so that it suits better to certain functions of the interfaces.

Table 1 shows the general structure of the graphical user interface and the description each class. Except setup.py and main.py, all other Python files include a ClickLabel class that inherits QLabel class. ClickLabel class is basically a QLabel class but can emit "clicked" signal on "mouseReleaseEvent".

Table 1: general structure of the GUI and the description classes

Python files	**UI files**	**Class**	**Inherited**	**Description**
Mainwindow.py	mainwindow.ui	Mainwindow	QWidget	The main window of the interface. It is also the first window shown up when the application starts.
Reportwindow.py	basic_report_form.ui	Reportwindow	QWidget	The window for the Report function. It visualizes data collected in real-time to graphs.
Climatewindow.py	climate.ui	Climatewindow	QWidget	The window displays the current real time value of the climate such as temperature, humidity, air pressure and soil moisture.

StreamWindow.py	basic_report_form.ui	StreamWindow	QWidget	Open and display stream video from a http link
	None	Camera	None	Capture stream image using openCV
DeviceWindow.py	device.ui	DeviceWindow	QWidget	The window shows current available devices in the system and their status. It can directly send to control the light system.
setup.py	None	None	None	Includes constant variables that the graphical user interface uses to operate normally
main.py	None	None	None	The main file that will start the application. It also read command line parameter to decide whether it show open the application in window or full screen mode.

Figure 3 shows the relationships between each window and windows with the database and nodes. The main window directs the current window to one of the windows with specific functions, such as climate window or device window. To move from climate window to report window, the user need to go back to main window, then push the button "Report" to be directed to report window.

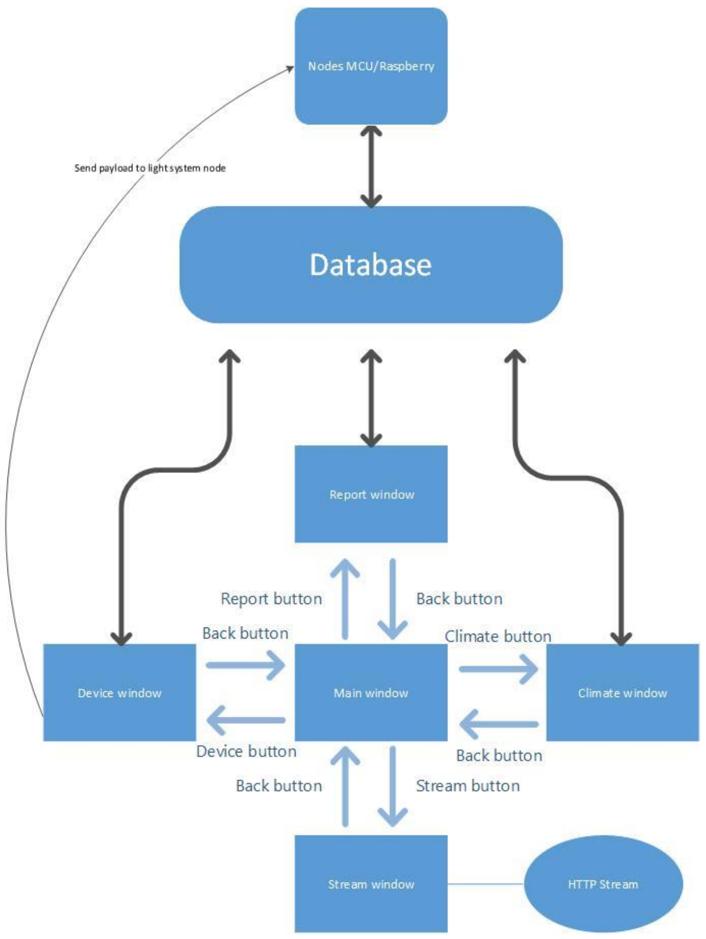

Figure 3: Relationships between windows with the database and nodes

2.2.1 Main window

1. _init_ function: Initialize variable, set up graphical element and connect buttons with its functions.
2. report_clicked(self): executed when report button is clicked. It creates an object to Reportwindow class and hide main window.

3. climate_clicked(self): executed when report button is clicked. It creates an object to Climatewindow class and hide main window.
4. security_clicked(self): executed when security button is clicked. It creates an object to StreamWindow class and Camera class from StreamWindow.py, then hide main window.
5. device_clicked(self): executed when device button is clicked. It creates an object to DeviceWindow class and hide main window.

2.2.2 Report window

1. _Init_: Initialize variable, set up graphical element and connect buttons with its functions. Create a timer that refresh the plot area every second using QTimer. _init_ takes the object to the previous window as parameter.
2. Change_data_type(self, data_type): change the self.data_type variable to one of the four reported climate element.
3. sendEmail(self): executed when email button is clicked. It runs send_mail() from send_mail.py and creates a QMessageBox to inform user that whether email is sent successfully or not.
4. back(self): executed when back button is clicked. It stops the timer, closes current window and enable previous window.
5. changeType(self): executed when type_change button is clicked. It changes the text of the plot type displayed on the header of the window to another following this succession: Minutely, Hourly, Daily, Monthly, Minutely.
6. Plot(self): executed every one second. It read the database based on the current data type and plot type using pandas. Then, it clears the plot area and draw new graph. Some attributes of the graph, such as Label, axis limit, are modified properly to each type of data.

2.2.3 Climate window

1. _init_: Initialize variable, set up graphical element and connect buttons with its functions. Create a timer that refresh the window every three seconds using QTimer. _init_ takes the object to the previous window as parameter.
2. Back(self): executed when back button is clicked. It stops the timer, closes current window and enable previous window.
3. Update(self): executed every three second. It reads the database by pandas and display values on the screen. It also saves the maximum and minimum values read so that the value range can be calculated.

2.2.4 Stream window

A. Streamwindow class
    1. _init_: Initialize variable, set up graphical element and connect buttons with its functions. Create a timer that refresh the plot area every 100 milisecond using QTimer. _init_ takes the object to the previous window and object to Camera class as parameters.
    2. Update_image(self): capture image frames, transpose the images and display them using QImageView.
    3. Back(self): executed when back button is clicked. It stops the timer, closes the camera and current window and enable previous window.

B. Camera class
    1. _init_: initialize camera number or http URL.
    2. Initialize(self): Open video stream using camera number or http URL.
    3. Get_frame(self): read and return the last frame of the video stream.
    4. Acquire_movie(self, num_frames): return an array of frame (a video / movie) capture from video stream.
    5. Set_brightness(self): set the brightness value.
    6. Get_brightness(self): return the brightness value.
    7. Close_camera(self): release the video capture.

2.2.5 Device window

1. _init_: Initialize variable, set up graphical element and connect buttons with its functions. Create a timer that refresh the window every three seconds using QTimer. Connection to node MCU is established using MQTT Client. _init_ takes the object to the previous window as parameter.
2. Back(self): executed when back button is clicked. It stops the timer, closes current window and enable previous window.
3. Update(self): executed every 3 second. It reads the database and update the device list. Then, it call display() to show the result on the screen.
4. Display(self): Clear the table and display device list.
5. doubleClicked(self, row, col): executed when a certain cell of the table is double-clicked. A sub-window for light adjustment appear at the bottom of the window.
6. lightControl(self): executed when there is a change brightness value. On manual mode, it publishes payload 10 when brightness value is less than 3, or else publishes payload 11. On auto mode, it does nothing.
7. modeChange(self): executed when mode button is clicked. When mode is changed from manual to auto, it publishes payload 00. On the reverse direction, it acts similar to lightControl().

2.3 BACKEND

2.3.1 Subscription (sub.py)

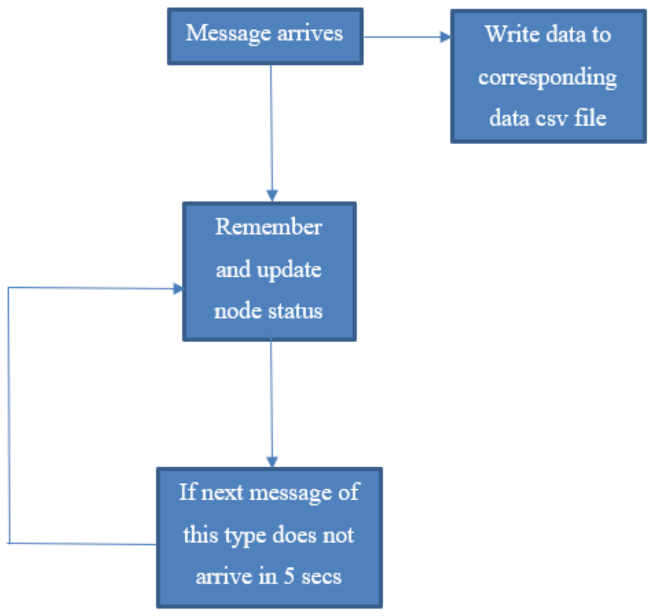
Message arrives
Write data to corresponding data csv file
Remember and update node status
If next message of this type does not arrive in 5 secs

Figure 4: Subscription mechanism summary

The application first establishes itself as a MQTT server and start to listen to messages according to a list of specified topics, i.e. 'esp8266/soil', 'esp8266/humid', etc.
On the arrival of a data message, it will perform string manipulation on its topic to extract the data kind and then construct a new csv data string and append it respectively to the right csv file, i.e. soil.csv, humid.csv, etc.
At the same time, the arrival of any particular message also tells the application the status of the node sending that message. The first time a node sends a message and is received by the controller, it will remember the node and if it does not send any message within 5 seconds, the application will assume that the node is off. The status of all remembered nodes is constantly updated to data file device_list.csv.

2.3.2 Publish (pub.py)

The application from the script is just a simulation for development of the light control system by sending data message to the node controlling the system. In contrast to the subscription application which runs independently, this implementation is directly integrated into the GUI application.

2.3.3 Data Management (data_manage.py)

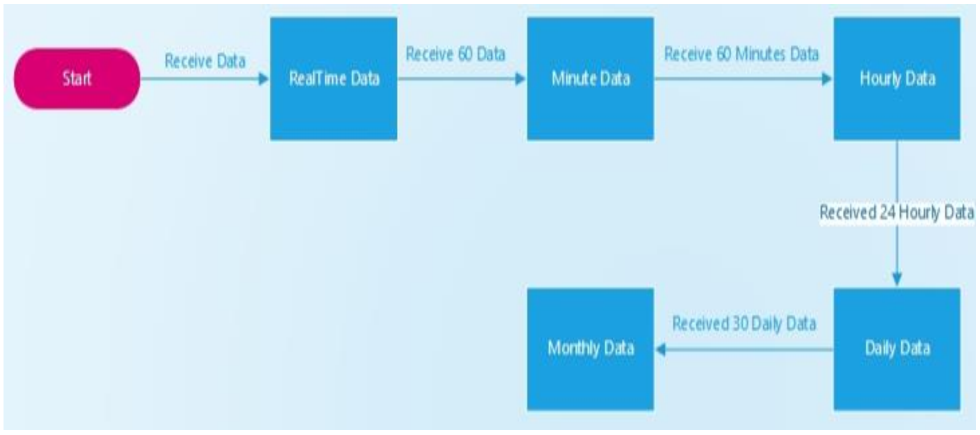

Figure 5: Data management behaviour summary

This independent application iterates over every single csv file and performs data average with respect to the kind and type of the data file. If the name of a file has type '_monthly' it will wait until the file acquired 12 entries and average, it to a new entry and append it to the data file with type 'yearly' of the corresponding kind. Similarly, for type 'daily', the application will average after 30 entries and append the new entry to file with type '_monthly'. for type 'hourly', the application will average after 24 entries and append the new entry to file with type '_daily'. for type 'minute', the application will average after 60 entries and append the new entry to file with type '_hourly'. Lastly, secondly file name does not have any type, the application will average after 60 entries and append the new entry to file with type '_minutely'.

2.3.4 Report Email (send_mail.py)

The application fetches the latest entries in every data file perform string manipulation on the file name for the data kind and type and construct an email then it through the Google SMPT server using its own Gmail account.

3 HARDWARE

3.1 ESP8266 NODEMCU

Wi-Fi Module – ESP-12E module is similar to ESP-12 module but with 6 extra GPIOs.
USB – micro USB port for 5V power supply, programming and debugging.
Headers – 2x 2.54mm 15-pin header with access to GPIOs, SPI, UART, ADC, and power pins.
Misc – Reset and Flash buttons.

3.2 NODES:

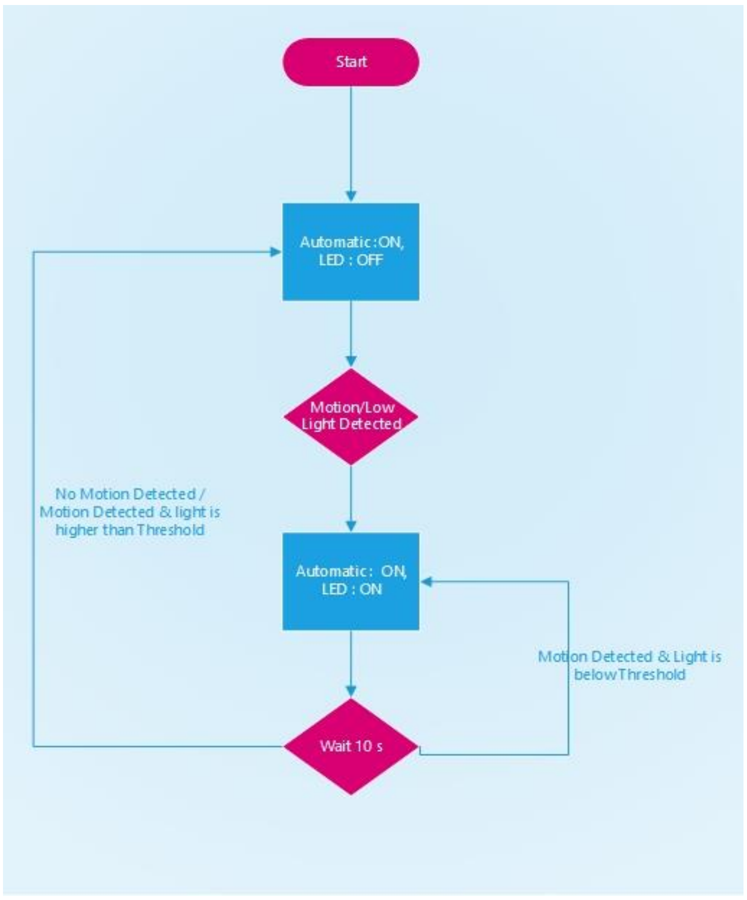
Start
Automatic:ON, LED : OFF
Motion/Low Light Detected
No Motion Detected / Motion Detected & light is higher than Threshold
Automatic: ON, LED : ON
Motion Detected & Light is below Threshold
Wait 10 s

Figure 6: the light system workflow diagram

Node1: using BME280 temperature, humidity and air pressure sensor.

Node2: using KEYESTUDIO soil moisture sensor

Node3: using KEYESTUDIO PIR sensor, photoresistor, SRD-05VDC-SL-C relay and LED light bulb.

Node4: using MQ135 gas sensor and a buzzer.

Node5: using Raspberry Pi 3 + Pi camera.

3.3 MAIN CONTROLLER:

Raspberry Pi 3 Model B + 7-inch touchscreen.

Functions:

- Receive data from nodes sensors and save them into database under csv format:
- Publish messages to the light system node allowing switching modes and control light state(Automatic - using sensors / manually - ON/OFF switches).

The system is currently powered by any microUSB connector external battery or adapter. The light bulb is powered by 12V power supply. These nodes were designed to consume as little power as possible.

Moreover, INA219 current sensor was used to calculate the power supply of the LED.

4 VIDEO SURVEILLANCE

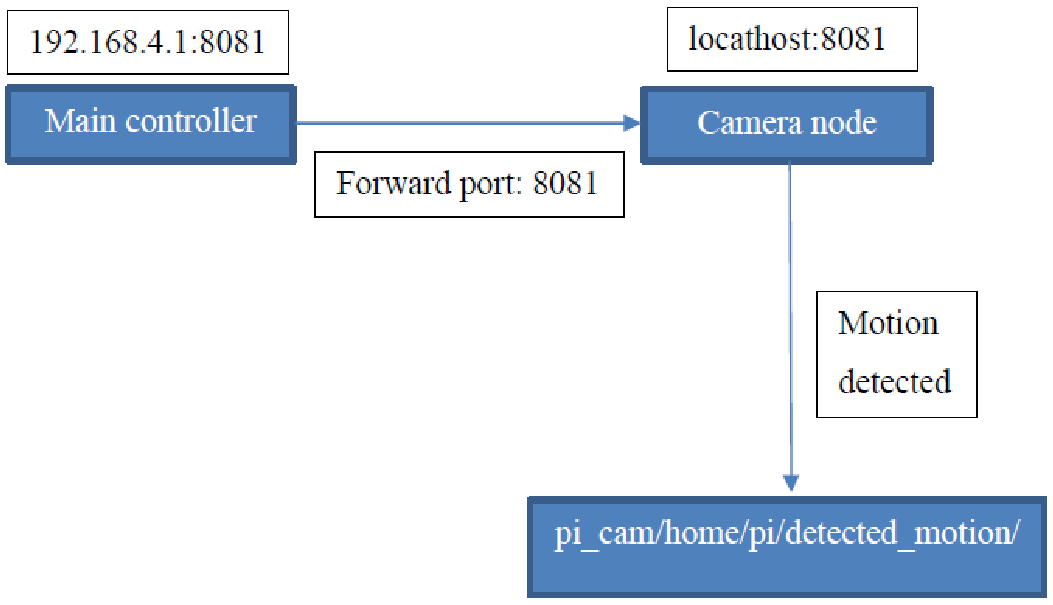

Figure 7: camera node workflow diagram

This feature is enabled by the motion Linux application installed on the Raspberry Pi. In motion.config file, the application is set to run as a daemon with image resolution of 640x480 with motion detection is set at default threshold of 1500 and noise level of 32 and any detection is saved in directory in pi_cam/home/pi/detected_motion/ with event gap of 60 seconds. Web streaming is on port 8081 with output is 100% the original quality.

For the current development progress, access authentication was disabled after configured due since the GUI application has not yet implemented the login action. However, the authentication feature can still be enabled by setting variable *stream_auth_method* to *2* and uncomment the line of variable *stream_autentication* then restart the motion daemon with command *sudo service motion restart*.

Apart from accessing the video output from the GUI application, the user can access the video stream on the web within the WiPi network with address 192.168.4.1:8081.

5 NETWORKING

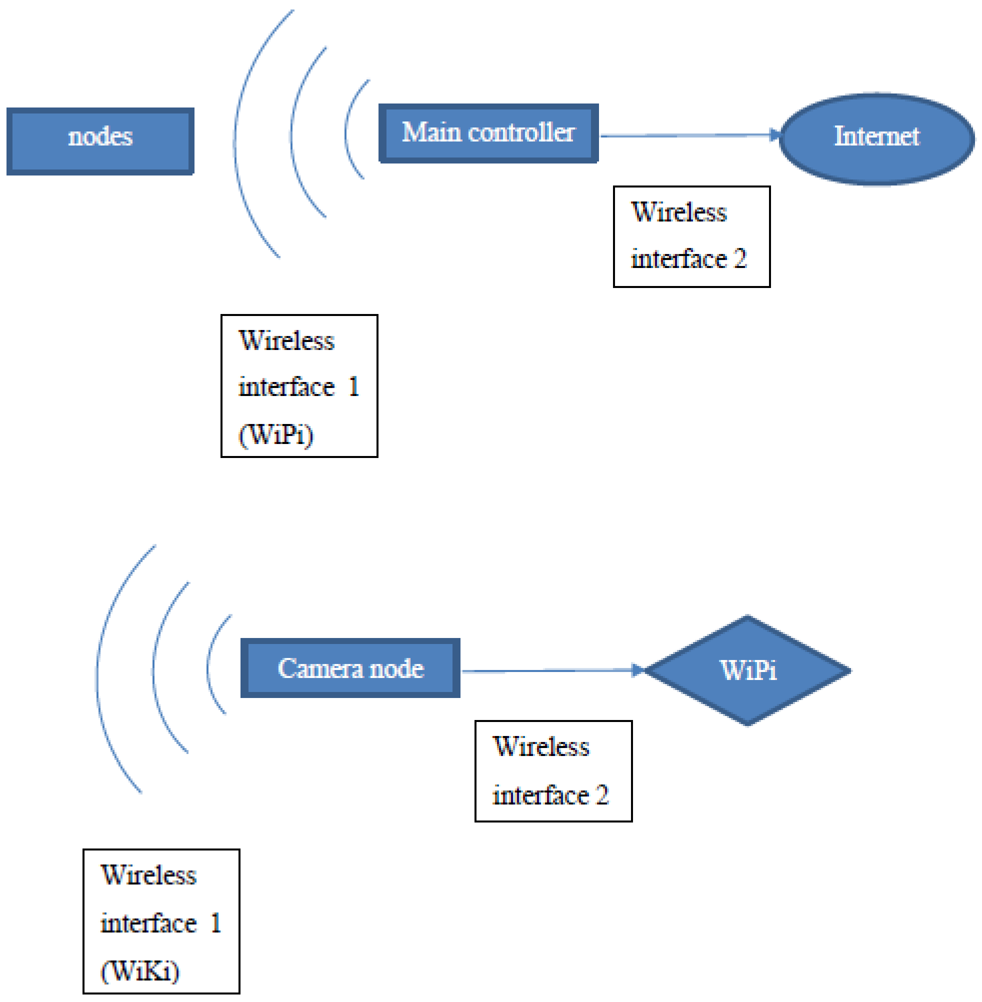

Figure 8: system networking workflow diagram

The main controller is set up to have 2 wireless interfaces, one integrated in the mainboard and the other is an 150Mbs Wi-fi dongle. The first Wi-fi interface is configured to serve as a wireless access point and broadcast network WiPi with addresses range from 192.168.4.20-50. The applications enabling this feature are dnsmasq and hostapd. Also, it's also needed to set static gateway ip address of 192.168.4.1 in dhcpd.conf of dhcp daemon. The secondary external wireless interface is configured to connect and forward data packets from WiPi to the internet by setting up NAT of iptables. However, it still works in similar manner if the dongle is replaced with the ethernet cable or if both are plugged in.

In addition, the camera node is also equipped with its own Wi-fi network WiKi in which it is convenient for debugging and testing. By plugging in either an ethernet cable or a Wi-fi dongle or both to connect to WiPi or any other network, it could also work as another repeating access point.