Definitions – Phase 1

Raspberry Pi Side

/etc/lirc/lircd.conf – Remote IR mapping (currently at 38kHz)

/home/pi/modules/mainSys.c – main system for mqtt and lirc – compiled to mainSystem

Compile: gcc –o [outputfile] mainSys.c –l mosquitto

float getTemperature()

get temperature data from sensor = identical definition with the one in si7021-2.c

float getCurrent()

get current data from sensor = identical definition with the one in ads1115.c

void messageArrived(struct mosquitto \*m, void \*obj, const struct mosquitto\_message \*message)

classify message received by mosquitto subscriber

format: \*ABC where

A = {d, u, 0, 1, r}. d = decrease temperature, u = increase temperature, 0 = power off, 1 = power on, r = read sensor

B (if only A is r) = power status of the AC {0, 1}. 0 = off, 1 = on

C = {16 … 30} (temperature)

For each type of command,

1. Build LIRC response

execute system call of LIRC

format: irsend SEND\_ONCE MY\_REMOTE COMMAND

COMMAND: a proper and suitable keypress defined in lircd.conf

1. (If command is r) read sensor data
2. Build MQTT response

format: P;T;S;C;% where P = power (ON/OFF), T = temperature (float), S = set/target temperature (int), C = current (float)

int main(int argc, char \*\*argv)

initialize I2C bus for temperature reading, initialize mosquitto, connect to server, and subscribe to the specified topic. Contains a main message-waiting loop

/home/pi/modules/i2c/ads1115.c – current sensor (for testing)

int main()

initialize i2c bus and device (location 0x49)

commanding the device by sending configurations:

0x01 – select config register

0x80 - AIN0 and AINN = AIN1, +/- 6.144V

0xE3 - Continuous conversion mode, 860 SPS

read 2 bytes (msb, lsb) reading

conversion from binary to voltage, and then voltage to current

(for debugging) printing the current level

/home/pi/modules/i2c/si7021-2.c – temperature/humidity sensor [b hum+temp; h hum; t temp] (for testing)

float getHumidity(int device)

sends command 0xF5 (humidity)

returns humidity level

float getTemperature(int device)

sends command 0xF3 (temperature)

returns temperature in ºC

int main(int argc, char \*\*argv)

initialize i2c bus and device (address 0x40)

(for debugging) printing humidity level and/or temperature

(for debugging) saving log data to a file

/home/pi/modules/serverComm.c – sending sensor data to the server – compiled to serverComm

float getTemperature(int device)

get temperature data from sensor = identical definition with the one in si7021-2.c

float getCurrent()

get current data from sensor = identical definition with the one in ads1115.c

int main()

Initialization for I2C device, CURL initialization, obtaining timestamp, then sending PHP GET request to the server.

Server Side (tbniot.000webhostapp.com – ctrl.html works if and only if the MainSys.c is running)

tbniot server public\_html/add\_data.php – php GET parser for building MySQL add data query

tbniot server public\_html/connect.php – php credential for MySQL

tbniot server public\_html/index.php – php index containing MySQL queries and reformat for showing sensor data to the html body

tbniot server public\_html/ctrl.html – controller html

Showing AC status and providing AC control

Javascript main:

initializations, display infos, bind events

publish\_msg() (for debugging/tester)

composing a MQTT message and publishing it to the corresponding MQTT topic

publish(txt)

composing a MQTT message and publishing it to the corresponding MQTT topic

resetting the timer of connection checker

temp\_up()

sending command to increase temperature

temp\_down()

sending command to decrease temperature

read\_sensor()

sending command to get/refresh sensor reading

toggle\_power()

sending command to turn the AC’s power on/off and refresh the temperature reading

onConnectionLost(responseObject)

connection loss event

onMessageArrived(message)

message arrival event. Parsing the received message and updating the status fields

format: P;T;S;C;% where P = power (ON/OFF), T = temperature (float), S = set/target temperature (int), C = current (float)

send\_console(text)

add text to the debugging console

check\_response()

check whether all messages have been responded to determine if the systems (Pi and controller) are connected or not

/home/pi/modules/ServerSend.sh

content: /home/pi/modules/serverComm - Automatic sensor sending every 5 minutes

/etc/cron.d/ServerSend

Location of cron job

content: \*/5 \* \* \* \* pi /home/pi/modules/ServerSend.sh

How to Start the System

1. Sensor data is automatically uploaded to the server every 5 minutes since reboot
2. Run mainSys [can be automated]
3. Open ctrl.html any time when mainSys is loaded

Hardware Calibrations

Celsius = (output)(175.72)/65536 – 46.85

Volt = (output)(5.14)/27468

Ampere = (volt)(625/3000) + 0.1092