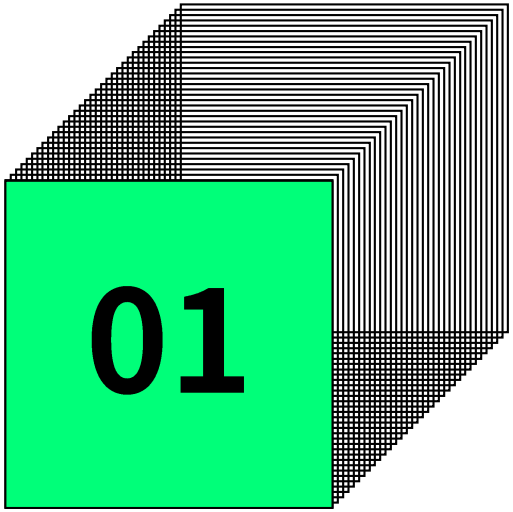
라즈베리 파이를 이용한 비트코인 상승장/시세 알리미

멀티미디어 임베디드 프로그래밍
– 이순용 교수님

정조안 (3조)
정경은 / 안선영 / 조예진

Index



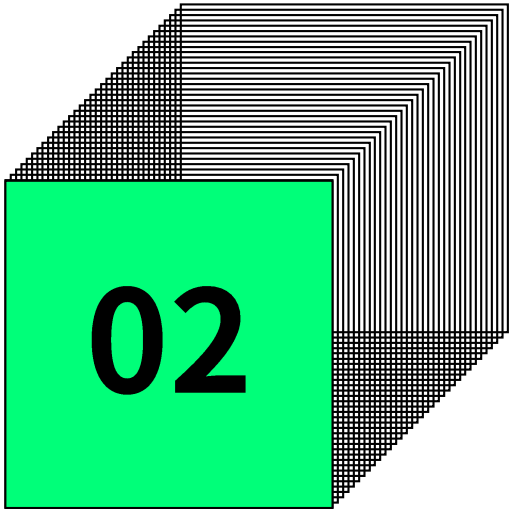
역할 분배

역할 분배

1. 정경은 : 팀장,
   시세/상승장/현재시각(16*32 도트매트릭스) 구현,
   웹 크롤링 구현, PPT, 발표

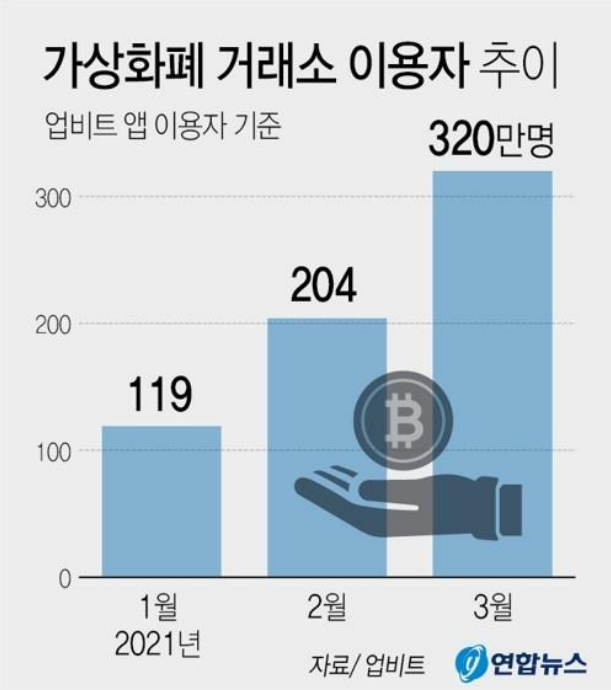
2. 안선영 : 정산,
   시세/상승장/현재시각(16*32 도트매트릭스) 구현,
   HTS(GUI 전체) 구현, PPT

3. 조예진 : 기록,
   인체감지센서 구현, 8*8 도트매트릭스 구현,
   시세 스레드 구성, PPT



배경/개요

프로젝트 배경



김영은 기자 / 20210407
트위터 @yonhap_graphics 페이스북 tuney.kr/LeYN1

높아지는 가상화폐 거래소의 관심도

+

라즈베리파이의 편리성

프로젝트 개요

시세는 24시간 바뀌는데,
매번 찾아보기가
귀찮다고 느껴질 때

시세와 현재 시각을
함께 보고 싶을 때
(비트코인은 시간이 생명이다.)

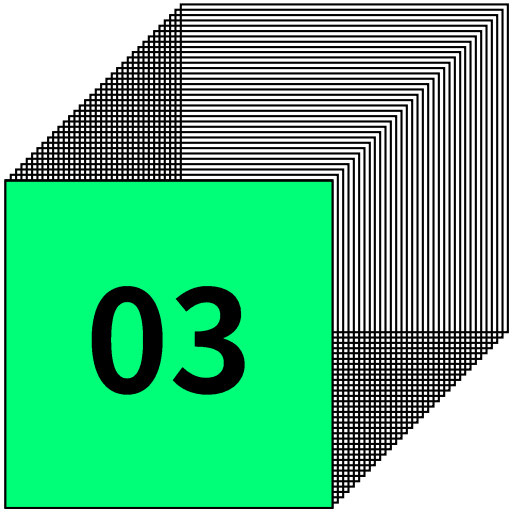
인체감지 센서로
손쉽게 기능을 On/off
하고 싶을때

프로젝트 개요

프로젝트 명 : 비트코인 시세 알리미

프로젝트 목적 : 트레이딩 사이트를 자주 들여다 볼 여유가 되지 않는 사람들을 위한 현재 시각 기능을 탑재한 비트 코인 시세 알림 서비스

프로젝트 일정 : 2021년 10월 7일 ~ 12월 9일



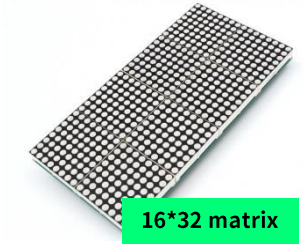
주 기능

사용하고자 한 디바이스

라즈베리 파이 2대

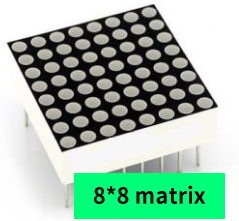
인체감지 센서

5V 어댑터



16*32 matrix

라즈베리파이와 연동하는 LCD



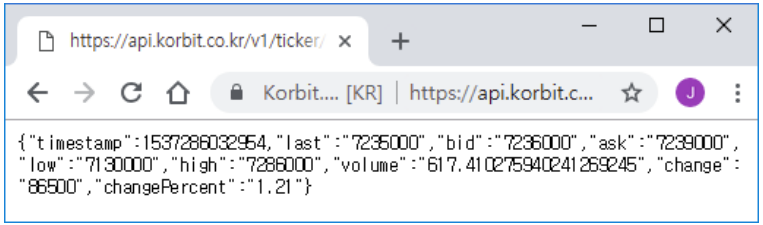
8*8 matrix

주 기능 - 호가 프로그램

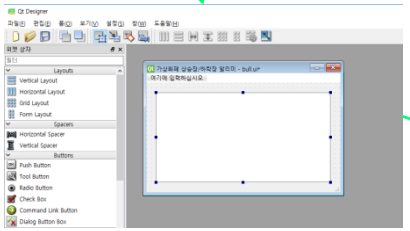
라즈베리 파이

Main 기능 : 호가창(HTS) 구현
- Python(3.7) , PyQT5
- Html, JSon

Html, Json -> 웹 크롤링



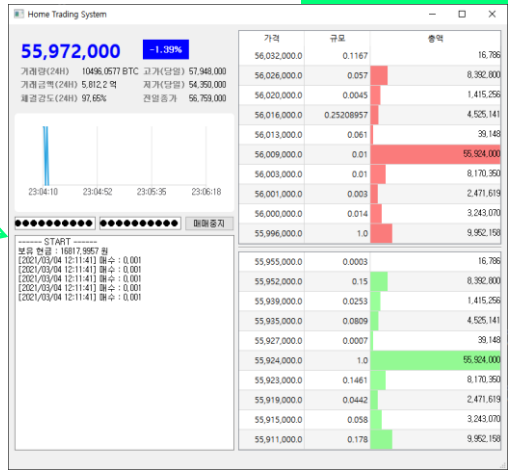
RestfulAPI 와 같은 API를 사용해
웹 주소를 통한 웹 크롤링 (현재가를 얻어오거나 시세 계산 (anti-aliasing))
: 빗썸, 업비트, 코빗 중 빗썸을 사용했다.
-> Pandas dataframe 을 사용해 값 저장 및 정렬



PyQT 사용 -> GUI 구현

QTimer, 사용자 정의 슬롯
=> UI 제작

최종 결과 화면



스레드를 통해 실시간 데이터를 출력

주 기능 - 라즈베리파이 간 통신

용도 : 호가창에서 필요한 pybithumb 데이터를 주고 받고, 이를 연결하기 위해

시리얼 통신

라즈베리 파이 – (1)

라즈베리 파이 – (2)

UART Rx/Tx GPIO와
USB to Serial 어뎁터의 Rx/Tx 핀을 서로 연결

시리얼 포트 설정에서 포트는 자동으로 테라텀에서 활성화 된 시리얼 포트를 탐색.
PC에 연결된 USB to Serial 어뎁터가 2개 이상이라면, 현재 사용하는 어뎁터의 포트를 찾아서 포트를 선택.

주 기능 - 인체 감지 센서

센서 모델 : 적외선 PIR 센서 인체감지 모션센서 HC-SR501 HCSR501

센서 용도 : PIR은 Passive Infrared의 줄임말로, 수동 적외선 센서.

자신의 시야 내에 있는 일정한 적외선을 띈 움직이는 물체 감지.
일정한 양의 적외선을 띈 물체의 움직임 -> trigger
움직임이 없다면 출력 방지

Ex) 현관문, 건물의 복도 … etc

구현 목적 : 뭐든 귀찮은 사람들을 위해 손 한번만 흔들어도 호가창이 켜지게끔 하는 용도이다.

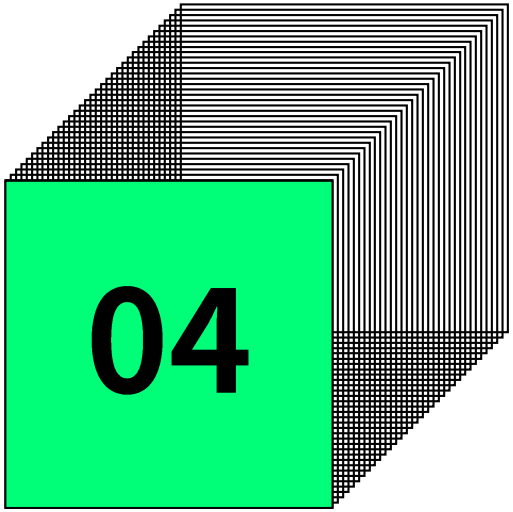
주 기능 - 시세 표시 및 시간 표시

센서 모델 : 16x32 RGB LED 매트릭스 패널[420] / 8*8 RGB LED 매트릭스

센서 용도 : 전면의 16x32 그리드에 512개의 밝은 RGB LED가 배열되어 있음.
뒷면에는 2개의 IDC 커넥터(1개의 입력, 1개의 출력: 이론상 함께 연결할 수 있음)와
1:8 스캔 속도로 디스플레이를 구동할 수 있는 12개의 16비트 래치가 있는 PCB가 있다.

구현 목적 : 호가창으로 부터 받아온 데이터와 현재 시각을 표현하고자 한다 (16*32).

인체감지 센서로부터 받아온 출력값을 통해 프로젝트 명이 스크롤 방식으로 출력된다. (8*8).



구성도

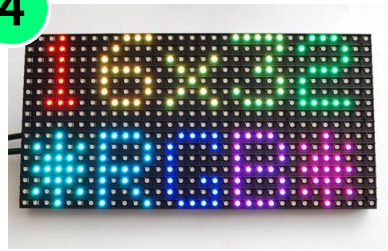
구성도

2

HDMI를 사용한
SSH 원격제어

노트북 LCD

4



16X32 RGB MATRIX

3

Serial 통신
코인 시세창 data를 가져온다.

1

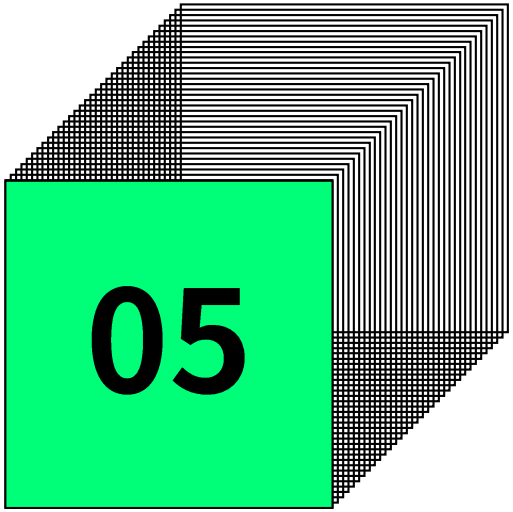
인체감지센서
(PIR)

라즈베리파이 3 B+

2

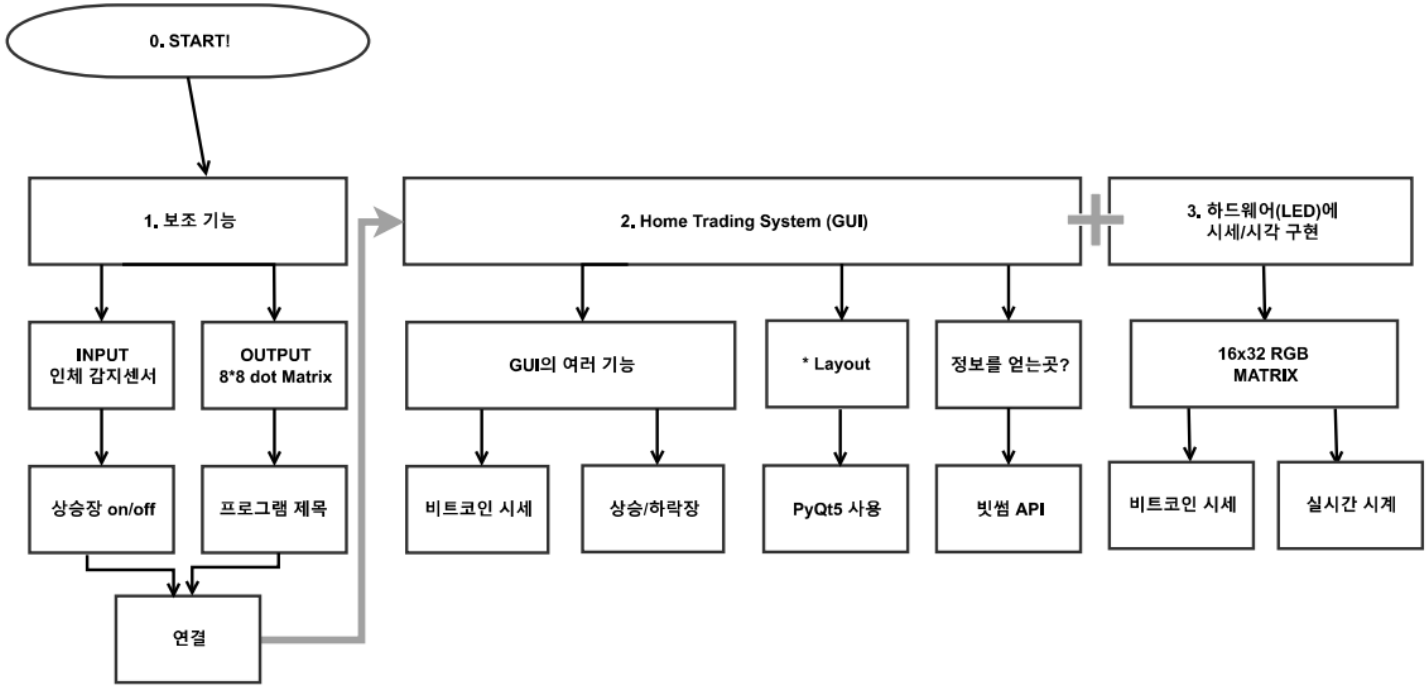
8X8 MATRIX

라즈베리파이 3 B+



세부 흐름도

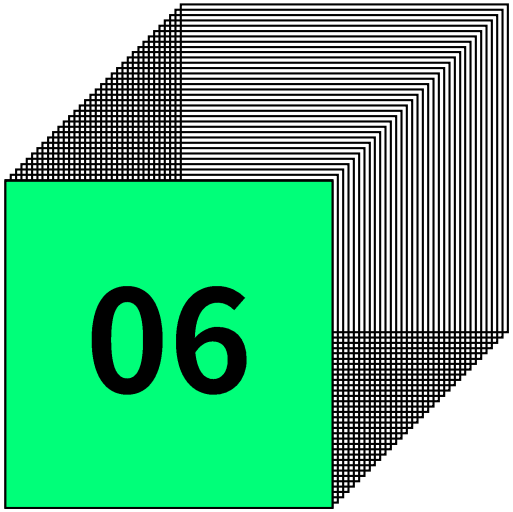
흐름도



설명:

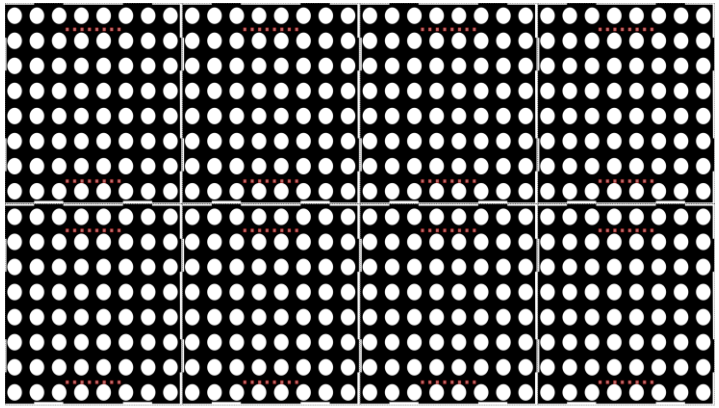
(프로그램은 1 -> 3 순서대로 진행된다)

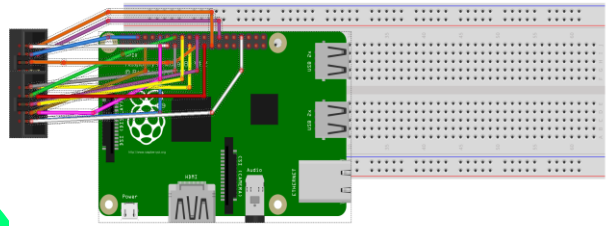
1. 보조 기능 – 인체감지센서로 HTS 창을 띄운다.
2. HTS – GUI에 비트코인 시세/상승&하락장을 표현한다.
3. LED – 16*32 Dot matrix에 실시간 시세/시각을 표현한다.



하드웨어 설계도

16*32 - Matrix 하드웨어 설계도





설명:

- 16*32 도트 매트릭스: 전체 핀들 다 연결되어야 정상 구현 가능.

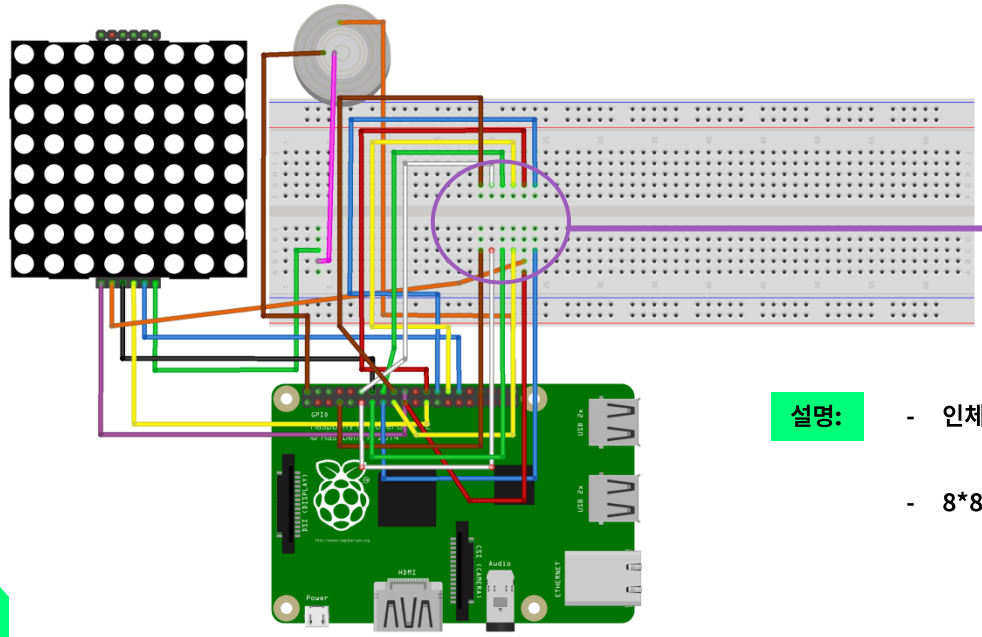
  6번 – GND, 7번 STROBE, 10번 E, 11번 CLK

  12번 – OE-, 13번 - G1, 15번 - A, 16번 - B

  18번 - C, 19번 - B2, 21번 - G2, 22번 - D,

  23번 – R1, 24번 – R2, 26번 – B1

인체 감지 센서 (PIR) 하드웨어 구조



* 해당 부분 핀들은 사용X

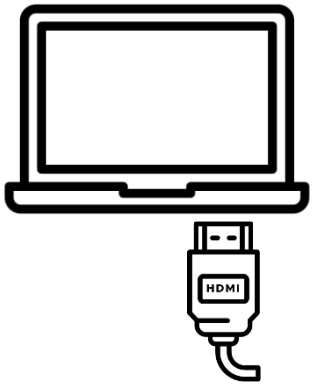
설명:

- 인체 감지 센서: 21번 pin을 통해 사용자 인식
  2번 – 전원, 6번 - GND
- 8*8 도트 매트릭스: 전체 핀들 다 연결되어야 정상 구현 가능.
  2번 – 전원, 6번 – GND,
  19번 – DIN, 23번 – CLK, 24번 – CS

* 인체 감지 센서/도트 매트릭스 동시 사용으로 일부 겹치는 핀들은 브레드보드에 연결하였음.

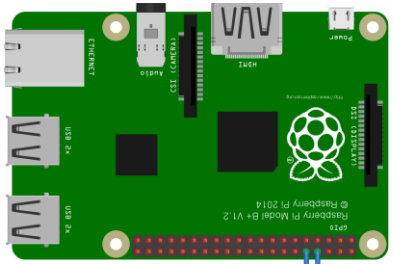
라즈베리파이 통신 하드웨어

LCD

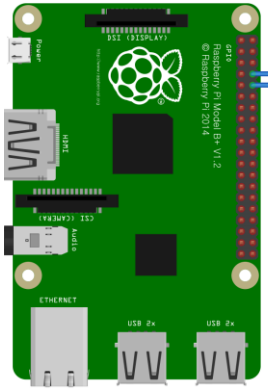


Pi1
(HDMI로 노트북과 연결)

Pi2



Pi1



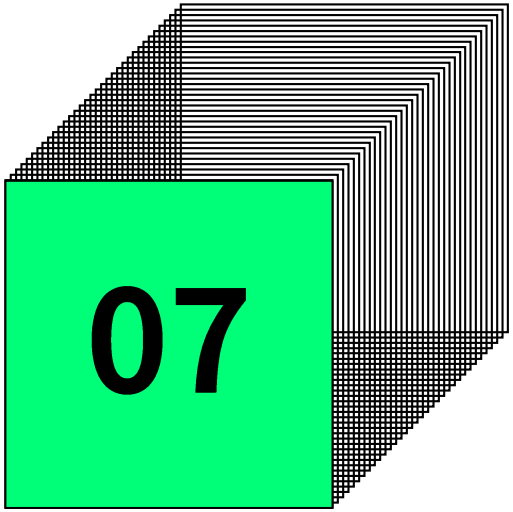
Pi1 <-> Pi2 시리얼 통신

설명:

1. Pi1은 HDMI와 SSH 원격제어 통신으로 LCD와 연결됨
2. 노트북에서는 HTS 호가창 열림
3. Pi1 – Pi2 사이의 시리얼 통신으로 pi2에 시세/시각 값을 전달

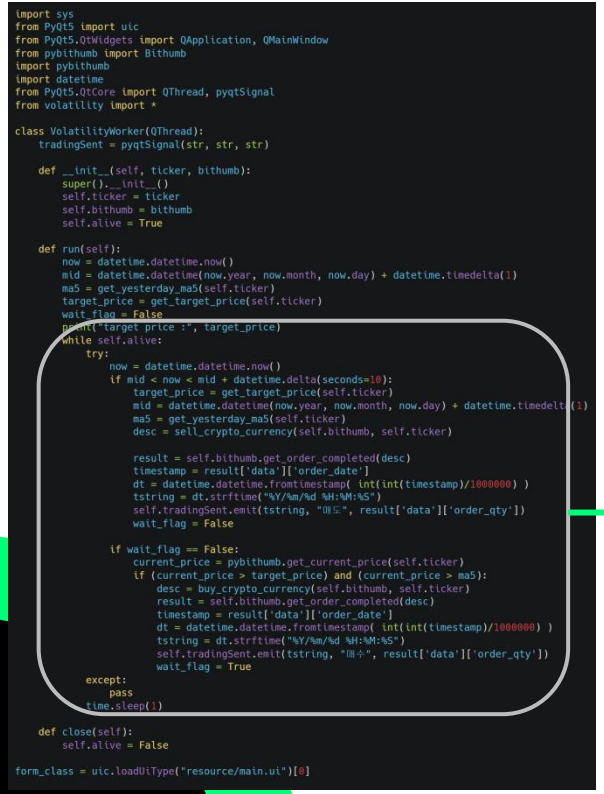
➔ 왜 시리얼 통신을 사용했는가? 하나의 라즈베리파이에 구현하기엔 충분한 사양이 필요했고, 두개의 라즈베리파이를 통신하자는 결론

➔ 결론적으로 모두 연결 가능한 상태 완성



코드 설명

Python, pyqt, pybithumb 라이브러리를 활용한 비트코인 시세 호가창



```python
import sys
from PyQt5 import uic
from PyQt5.QtWidgets import QApplication, QMainWindow
from pybithumb import Bithumb
import pybithumb
import datetime
from PyQt5.QtCore import QThread, pyqtSignal
from volatility import *

class VolatilityWorker(QThread):
    tradingSent = pyqtSignal(str, str, str)

    def __init__(self, ticker, bithumb):
        super().__init__()
        self.ticker = ticker
        self.bithumb = bithumb
        self.alive = True

    def run(self):
        now = datetime.datetime.now()
        mid = datetime.datetime(now.year, now.month, now.day) + datetime.timedelta(1)
        ma5 = get_yesterday_ma5(self.ticker)
        target_price = get_target_price(self.ticker)
        wait_flag = False
        print("target price :", target_price)
        while self.alive:
            try:
                now = datetime.datetime.now()
                if mid < now < mid + datetime.delta(seconds=10):
                    target_price = get_target_price(self.ticker)
                    mid = datetime.datetime(now.year, now.month, now.day) + datetime.timedelta(1)
                    ma5 = get_yesterday_ma5(self.ticker)
                    desc = sell_crypto_currency(self.bithumb, self.ticker)

                    result = self.bithumb.get_order_completed(desc)
                    timestamp = result['data']['order_date']
                    dt = datetime.datetime.fromtimestamp( int(int(timestamp)/1000000) )
                    tstring = dt.strftime("%Y/%m/%d %H:%M:%S")
                    self.tradingSent.emit(tstring, "매도", result['data']['order_qty'])
                    wait_flag = False

                if wait_flag == False:
                    current_price = pybithumb.get_current_price(self.ticker)
                    if (current_price > target_price) and (current_price > ma5):
                        desc = buy_crypto_currency(self.bithumb, self.ticker)
                        result = self.bithumb.get_order_completed(desc)
                        timestamp = result['data']['order_date']
                        dt = datetime.datetime.fromtimestamp( int(int(timestamp)/1000000) )
                        tstring = dt.strftime("%Y/%m/%d %H:%M:%S")
                        self.tradingSent.emit(tstring, "매수", result['data']['order_qty'])
                        wait_flag = True
            except:
                pass
            time.sleep(1)

    def close(self):
        self.alive = False

form_class = uic.loadUiType("resource/main.ui")[0]
```

```python
class MainWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.ticker = "BTC"
        self.button.clicked.connect(self.clickBtn)
        self.setWindowTitle("Home Trading System")

        with open("bithumb.txt") as f:
            lines = f.readlines()
            apikey = lines[0].strip()
            seckey = lines[1].strip()
            self.apiKey.setText(apikey)
            self.secKey.setText(seckey)

    def clickBtn(self):
        if self.button.text() == "매매시작":
            apiKey = self.apiKey.text()
            secKey = self.secKey.text()
            if len(apiKey) != 32 or len(secKey) != 32:
                self.textEdit.append("KEY가 올바르지 않습니다.")
                return
            else:
                self.bithumb = Bithumb(apiKey, secKey)
                self.balance = self.bithumb.get_balance(self.ticker)
                if self.balance == None:
                    self.textEdit.append("KEY가 올바르지 않습니다.")
                    return

            self.button.setText("매매중지")
            self.textEdit.append("------ START ------")
            self.textEdit.append(f"보유 현금 : {self.balance[2]} 원")

            self.vw = VolatilityWorker(self.ticker, self.bithumb)
            self.vw.tradingSent.connect(self.receiveTradingSignal)
            self.vw.start()
        else:
            self.vw.close()
            self.textEdit.append("------- END -------")
            self.button.setText("매매시작")

    def receiveTradingSignal(self, time, type, amount):
        self.textEdit.append(f"[{time}] {type} : {amount}")

    def closeEvent(self, event):
        self.vw.close()
        self.widget.closeEvent(event)
        self.widget_2.closeEvent(event)
        self.widget_3.closeEvent(event)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mw = MainWindow()
    mw.show()
    exit(app.exec_())
```

빗썸 API 호출하기 위한 key , value 값 저장한 bithumb txt 파일 열기

PyQT widget 관련 부분 button 구현 및 전반적인 GUI 객체에 기능 구현

- Process spawning을 사용하여 부모 프로세스가 자식 프로세스를 새롭게 생성함을 요청한다.
- 업비트 거래소에서 구독 방식을 통해 전달받은 실시간 데이터를 PyQt를 사용한 GUI 프로그램으로 출력

인체감지센서로 프로젝트 on + 8*8 도트매트릭스로 프로젝트 제목 띄우기

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017-18 Richard Hull and contributors
# See LICENSE.rst for details.

import RPi.GPIO as GPIO

import re
import time
import argparse

from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT, SINCLAIR_FONT, LCD_FONT

GPIO.setmode(GPIO.BCM)

pirPin = 21
GPIO.setup(pirPin, GPIO.IN, GPIO.PUD_UP)

while GPIO.input(pirPin):
    if GPIO.input(pirPin) == GPIO.LOW:
        print("Motion Detected")
    else:
        print("No Motion")
    time.sleep(0.2)

def demo(n, block_orientation, rotate, inreverse):

    serial = spi(port=0, device=0, gpio=noop())
    device = max7219(serial, cascaded=n or 1, block_orientation=block_orientation,
                     rotate=rotate or 0, blocks_arranged_in_reverse_order=inreverse)
    print("Motion Detected")

    msg = "Bitcoin Market Price & Time Information System"
    print(msg)
    show_message(device, msg, fill="white", font=proportional(LCD_FONT), scroll_delay=0.08)
    time.sleep(1)

    msg = "Welcome!!"
    print(msg)
    show_message(device, msg, fill="white", scroll_delay=0.05)

    time.sleep(1)

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='matrix_demo arguments',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('--cascaded', '-n', type=int, default=1, help='Number of cascaded MAX7219 LED
matrices')
    parser.add_argument('--block-orientation', type=int, default=0, choices=[0, 90, -90],
help='Corrects block orientation when wired vertically')
    parser.add_argument('--rotate', type=int, default=0, choices=[0, 1, 2, 3], help='Rotate display
0=0°, 1=90°, 2=180°, 3=270°')
    parser.add_argument('--reverse-order', type=bool, default=False, help='Set to true if blocks are in
reverse order')

    args = parser.parse_args()

    try:
        demo(args.cascaded, args.block_orientation, args.rotate, args.reverse_order)
    except KeyboardInterrupt:
        pass
```

코드 목적 : 인체감지 센서의 일정 값을 받아와, 프로젝트 제목을 출력하며 호가창의 시작을 알린다.

- GPIO 핀 21번 사용
- while GPIO.input(pirPin) 통하여 입력값이 계속 들어오게 하고
- 센서가 별다른 반응 없으면 "No motion" 메시지 호출
- pir 센서가 위로 움직이는 물체에 반응하면
  "Motion detected!" 메시지 호출되고
  while문 종료되면서 도트매트릭스 켜짐과 동시에 모니터 on

- max7219 라이브러리 (dot matrix lib) 사용
- msg 변수에 프로그램 제목 저장 후 도트매트릭스에 호출
  (LCD_PONT 사용, scroll_delay = 0.08)
- msg 변수에 위와 동일하게 'Welcome' 메시지 저장 후 도트매트릭스에 호출
  (기본 폰트 사용, scroll_delay = 0.05)

16*32 도트매트릭스에 시세 + 현재시각 띄우기

```python
#!/usr/bin/env python
# Display a runtext with double-buffering.

from samplebase import SampleBase
from rgbmatrix import graphics
import time
import random
import pybithumb as pyBI
from datetime import datetime, timedelta


class RunText(SampleBase):
    def __init__(self, *args, **kwargs):
        super(RunText, self).__init__(*args, **kwargs)
        self.parser.add_argument("-t", "--text", help="The text to scroll on the RGB LED panel"
default="Hello world!")

    def run(self):
        offscreen_canvas = self.matrix.CreateFrameCanvas()
        now = datetime.now()
        delta = timedelta(seconds = 15)
        trigger_time = now + delta

        font = graphics.Font()
        font.LoadFont("../../../fonts/7x13.bdf")
        textColor = graphics.Color(255, 255, 0)
        textColor2 = graphics.Color(255, 0, 0)
        pos = offscreen_canvas.width
        pos2 = offscreen_canvas.height
        my_text = self.args.text
        bit_Data = self.args.text

        while pyBI > 0 :
            offscreen_canvas.Clear()

            now = datetime.now()
            #bit_Data = pyBI.get_ohlcv("BTC")
            bit_Data = pyBI.get_current_price(self.ticker)
            time.sleep(1)
            self.dataSent.emit(bit_Data)
            if int(current_time) > 11:
                current_time = now.strftime('%H:%M:%S')
                my_text=current_time
                print(current_time)
            else:
                print('error')
                current_time = now.strftime('%H:%M:%S')
            font = graphics.Font()
            pos = 0
            graphics.DrawText(offscreen_canvas, font, pos, 7, textColor, my_text)
            graphics.DrawText(offscreen_canvas, font, pos, 15, textColor2, bit_Data)

            time.sleep(0.05)
            offscreen_canvas = self.matrix.SwapOnVSync(offscreen_canvas)


# Main function
if __name__ == "__main__":
    run_text = RunText()
    if (not run_text.process()):
        run_text.print_help()
```

코드 목적 : 라즈베리 파이 1번에 있는 호가창의 비트 코인 시세를 받아와서, 그 시세를 배열로 저장, 그리고 시간마다 변동하는 그 시세를 16*32 dot matrix에 출력하고자 한다.

Init을 하는 노드 -> error 발생 시 어떠한 msg가 출력이 되어야 하는지 명시

- Offscreen_canvas 와 같은 해당 라이브러리에서 제공하는 rgbmatrix를 받아와, led 창에 띄워지게 함
- time 모듈을 사용하여 현재 시각 변수 등록

- time 모듈을 사용하여 현재 시각을 받아온다. Now.strftime('%H : %M :%S') 를 사용해서 데이터를 받는다.
- Bit_Data 라는 변수로, 라즈베리파이 1번에 있는 비트코인 시세를 받는다.
- Graphics.DrawText하는 함수를 사용해 bit_Data를 출력한다.

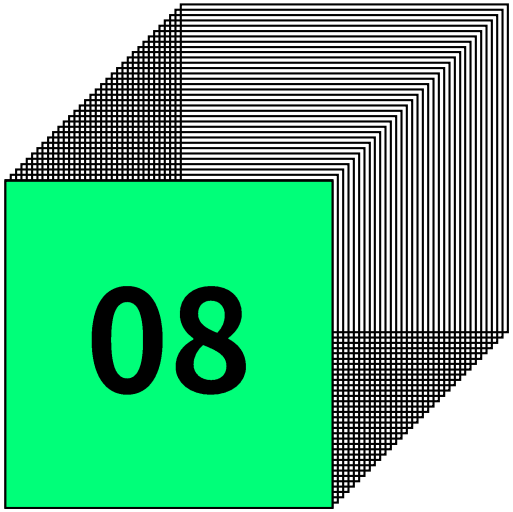
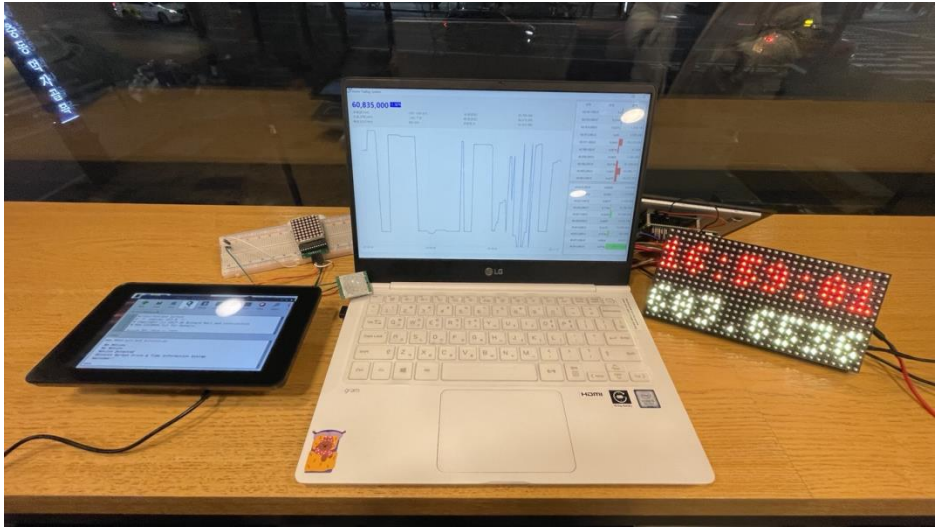
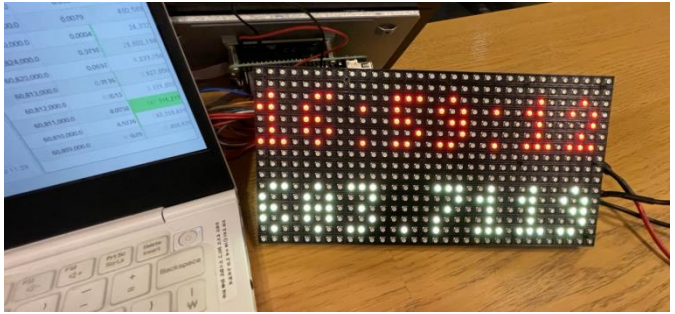
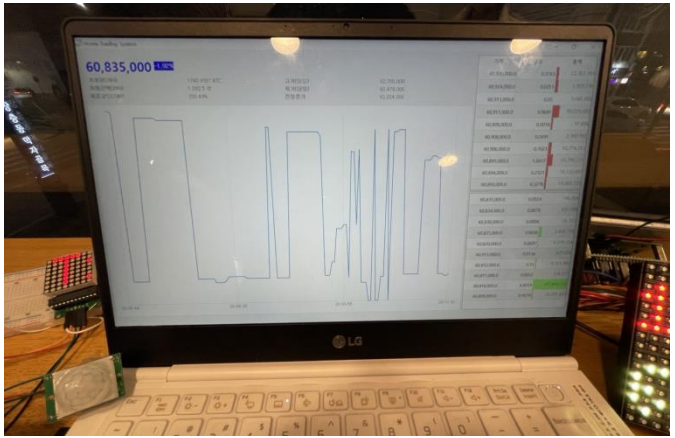


사진 / 동작 동영상

사진

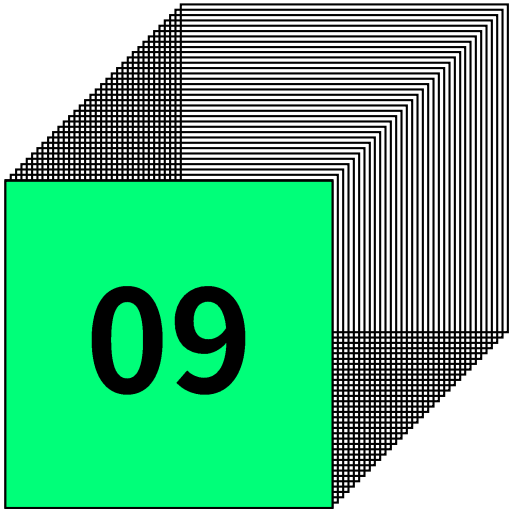






동영상

https://youtu.be/rjcahFfKHWU



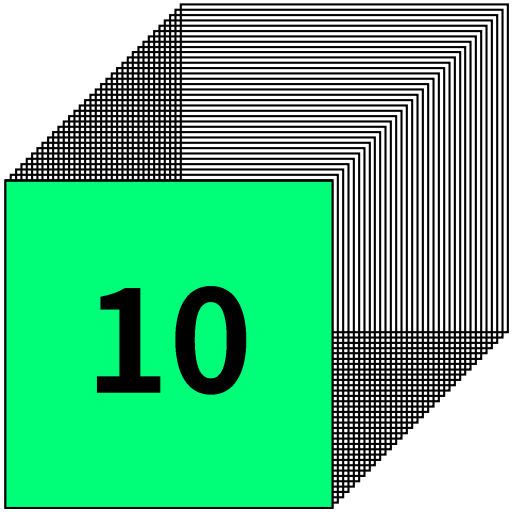
추진 일정

프로젝트 추진 일정

- **프로젝트 기획**
  - 프로젝트 주제 기획
  - GPIO 기능 선정
- **프로젝트 발표**
  - 프로젝트 발표 자료 작성 (정경은,안선영, 조예진)
  - 발표 (정경은)
- **시스템/하드웨어 기능 구현**
  - Home Trading System (GUI) 구현 (안선영)
  - 시세 데이터로 부터 16*32 도트매트릭스 시세 + 현재시각 구현 (정경은)
  - 인체감지센서 + 8*8 도트매트릭스 구현 (조예진)

프로젝트 추진 일정

	10월	11월	12월	비고
주제 선정(기획)				
GUI 개발				
GPIO 기능 구현				
최종 정리 + 발표				



결론

결론, 제한점, 향후 연구

결론

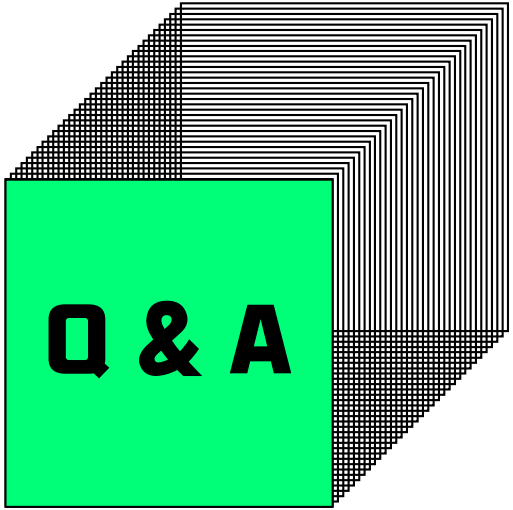
- HTS 구현을 통한 비트 코인 상승장 알리미 가시성 향상
- LED matrix에 시세/시각 구현을 통한 시각적 자극 효과
- 시간이 생명인 비트코인을 시시각각 확인할 수 있는 편리성
- 다양한 input값을 활용한 센서 기능 구현

제한점

- 트레이딩 시스템의 한계점. 실제로 트레이딩을 하기 위해선 웹소켓 네트워킹이 필요했으나, 모듈 설치의 에러와 같은 장애 요인이 있음

향후 연구

- 실제로 트레이딩 시스템을 구현하기엔, 최소 xavier 과 같은 싱글보드 컴퓨팅 파워가 필요하다.
- 추후에 더 다양한 센서들을 활용해 호가창을 통한 트레이딩 구현을 향후에 연구해볼 수 있겠다.


Q & A

이상 발표 마치겠습니다.
감사합니다.

정조안 (3조)
정경은, 조예진, 안선영