POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI
INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ
ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PROJEKT

Serwer IoT

(TEMAT PROJEKTU)

GRZEGORZ AWTUCH, KACPER JABŁOŃSKI, JĘDRZEJ GRZEBISZ, DAWID SKIBIŃSKI

(AUTOR/AUTORZY)

APLIKACJE MOBILNE I WBUDOWANE DLA INTERNETU
PRZEDMIOTU

(NAZWA PRZEDMIOTU)

LABORATORIUM

(FORMA ZAJĘĆ {PROJEKT/LABORATORIUM/ĆWICZENIA})

PROWADZĄCY:

mgr inż. Adrian Wójcik
adrian.wojcik@put.poznan.pl

POZNAŃ 16.09.2020

DANE SZCZEGÓŁOWE

Rok studiów: INŻ. III

Rok akademicki: 2019/2020

Termin zajęć: środa g. 13:30

Data wykonania: 2020/09/16

Temat zajęć: Serwer IoT.

Prowadzący: mgr inż. Adrian Wójcik

Skład grupy (nazwisko, imię, nr indeksu):

1. Jabłoński Kacper (135828)
2. Grzegorz Awtuch (135794)
3. Jędrzej Grzebisz (135825)
4. Dawid Skibiński (135889)

OCENA (*WYPEŁNIA PROWADZĄCY*)

Spełnienie wymogów redakcyjnych: ……

……

Wykonanie, udokumentowanie oraz opis wykonanych zadań: ……

……

Zastosowanie prawidłowego warsztatu programistycznego: ……

……

Spis treści

1 SPECYFIKACJA PROJEKTU

1.1 SPEŁNIONE WYMOGI PODSTAWOWE

- Serwer WWW umożliwia pobieranie danych pomiarowych ze wszystkich dostępnych w układzie wbudowanych czujników oraz wysyłanie danych sterujących do wszystkich elementów wykonawczych
- Serwer WWW umożliwia przesyłanie danych do wszystkich trzech aplikacji klienckich oraz odbiera dane sterujące od wszystkich trzech aplikacji klienckich
- Wszystkie trzy aplikacje klienckie umożliwiają podgląd danych pomiarowych za pomocą dynamicznie generowanego interfejsu użytkownika
- Wszystkie trzy aplikacje klienckie umożliwiają podgląd danych pomiarowych za pomocą wykresu przebiegu czasowego
- Wszystkie trzy aplikacje klienckie umożliwiają próbkowanie danych pomiarowych z okresem nie większym niż 1000 ms
- GUI wszystkich trzech aplikacji klienckich zawiera informacje o jednostkach wielkości pomiarowych
- Wszystkie trzy aplikacje klienckie umożliwiają sterowanie pojedynczymi elementami wykonawczymi w pełnym dostępnym zakresie kontroli
- Wszystkie trzy aplikacje klienckie umożliwiają konfiguracje komunikacji sieciowej oraz akwizycji danych

1.2 PODJĘTE WYMOGI DODATKOWE

- Implementacja systemu wykorzystuje architekturę REST
- Całość kodu źródłowego komentowana jest wg wspólnego standardu doxygen
- Wszystkie trzy aplikacje klienckie umożliwiają próbkowanie danych pomiarowych z okresem nie większym niż 100 ms
- Przy realizacji zadania wykorzystano system kontroli wersji Git
- Aplikacja desktopowa wykorzystuje wzorzec projektowy zapewniający separację interfejsu użytkownika od logiki aplikacji

1.3 PLATFORMY DOCELOWE

Docelowymi platformami są: urządzenia mobilne z systemem Android, przeglądarki internetowe, oraz komputery osobiste z systemem Windows.

2 IMPLEMENTACJA SYSTEMU

2.1 APLIKACJE SERWERA

Wszelkie dane przechowywane są na serwerze w postaci plików w formacie JSON. Przykładowy plik możemy zobaczyć na Listingu 1, ukazuje on składnie użytego formatu zapisu.

```
{"Temperature": 35.046875, "Humidity": 44.8203125, "Pressure": 613.02880859375}
```

Listing 1 Przykład zapisu danych w formacie JSON

Zapisywanymi danymi są informacje na temat ostatniego pomiaru temperatury, ciśnienia, wilgotności oraz kątów roll, pitch i yaw. Na serwerze znajdują się także dane dotyczące ustawień użytkownika, do których należą adres IP oraz port serwera, a także czas próbkowania i ilość wyświetlanych próbek na wykresie. Za zapis wcześniej opisanych danych odpowiadają skrypty w języku Python. Na Listingu 2

widać skrypt odpowiedzialny za odczyt danych z nakładki SenseHat tj. temperatura, ciśnienie i wilgotność. Listing 3 oraz 4 pokazują implementację sterowania diodami LED, zapalanie pojedynczej diody i wyświetlanie tekstu, również przy użyciu języka Python.

```
1.  #!/usr/bin/python3
2.
3.  import sys
4.  import getopt
5.  import os
6.  import json
7.  import time
8.
9.  from sense_emu import SenseHat
10. licznik = 0
11. while licznik<5:
12.     sense = SenseHat()
13.     pogoda = {"temperatura" : 0, "cisnienie" : 0, "wilgotnosc" : 0}
14.     pogoda["temperatura"] = sense.temperature
15.     pogoda["cisnienie"] = sense.pressure
16.     pogoda["wilgotnosc"] = sense.humidity
17.
18.     print(json.dumps("temperatura : " + str(pogoda["temperatura"]) + " C"))
19.     print(json.dumps("cisnienie : " + str(pogoda["cisnienie"]) + " hPa"))
20.     print(json.dumps("wilgotnosc : " + str(pogoda["wilgotnosc"]) + " %"))
21.
22.     file = open("pogoda.json", "w")
23.     file.write(json.dumps(pogoda))
24.     file.close()
25.     time.sleep(0.5)
```

Listing 2 Skrypt pythonowy do pomiaru i zapisu pogody

```
1.  #!/usr/bin/python3
2.  import json
3.  from sense_emu import SenseHat
4.  sense = SenseHat()
5.  red = (255, 0, 0)
6.  green = (0, 255, 0)
7.  blue = (0, 0, 255)
8.  filename = "led_single.json";
9.  if filename:
10.     with open(filename, 'r') as f:
11.         ledDisplayArray = json.load(f)
12. if ledDisplayArray["kolor"] == "red" or ledDisplayArray["kolor"] == "Red":
13.     sense.set_pixel(ledDisplayArray["poziom"], ledDisplayArray["pion"], red)
14. elif ledDisplayArray["kolor"] == "blue" or ledDisplayArray["kolor"] == "Blue":
15.     sense.set_pixel(ledDisplayArray["poziom"], ledDisplayArray["pion"], blue)
16. elif ledDisplayArray["kolor"] == "green" or ledDisplayArray["kolor"] == "Green":
17.     sense.set_pixel(ledDisplayArray["poziom"], ledDisplayArray["pion"], green)
```

Listing 3 Skrypt pythonowy odpowiedzialny za zapalanie diody

```
1.  #!/usr/bin/python3
2.  import json
3.  import time
4.  from sense_emu import SenseHat
5.
6.  sense = SenseHat()
7.
8.  time.sleep(1)
9.  red = (255, 0, 0)
10. green = (0, 255, 0)
11. blue = (0, 0, 255)
12. filename = "led_text.json";
13.
14.
```

```
if filename:
    with open(filename, 'r') as f:
        ledDisplayArray = json.load(f)

if ledDisplayArray["kolor"] == "red" or ledDisplayArray["kolor"] == "Red":
    sense.show_message(ledDisplayArray["text"], text_colour=red)
elif ledDisplayArray["kolor"] == "blue" or ledDisplayArray["kolor"] == "Blue":
    sense.show_message(ledDisplayArray["text"], text_colour=blue)
elif ledDisplayArray["kolor"] == "green" or ledDisplayArray["kolor"] == "Green":
    sense.show_message(ledDisplayArray["text"], text_colour=green)
```

Listing 4 Skrypt pythonowy do wyświetlania napisów na matrycy LED

Polecenia wydawane przez użytkownika przetwarzane są przez skrypty PHP, które uruchamiają odpowiednie skrypty Pythona. Umożliwiają zapis danych i ustawień do pliku, oraz wykonanie odpowiednich akcji na serwerze. Listing 5 pokazuje skrypt PHP odpowiedzialny za zapis wybranych ustawień do pliku na serwerze. Na Listingu 6 możemy zauważyć, że skrypt PHP uruchamia skrypt pythona odpowiedzialny za zapalenie diod na wyświetlaczu nakładki SenseHat oraz zapisuje dane do pliku.

```
<?php
$tablica = array();
$fp = fopen('zapisane_dane.json', w);

$tablica['czas_probkowania'] = $_POST['czaspro'];
$tablica['liczba_probek'] = $_POST['liczbapro'];

$tablicaJson = json_encode($tablica);
fwrite($fp, $tablicaJson);
fclose($fp);
?>
```

Listing 5 Skrypt PHP do zapisu ustawień

```
<?php
$ledDisplay = array();
$ledDisplayFile = fopen('led_single.json', w);

$ledDisplay['poziom'] = (int)json_decode($_POST['poziom']);
$ledDisplay['pion'] = (int)json_decode($_POST['pion']);
$ledDisplay['kolor'] = $_POST['kolor'];

$ledDisplayJson = json_encode($ledDisplay);
fwrite($ledDisplayFile, $ledDisplayJson);
fclose($ledDisplayFile);

exec("sudo ./led_display.py");
?>
```

Listing 6 Skrypt PHP do zapalenia diody LED

2.2 Aplikacja Webowa

Ułożenie strony napisaliśmy w języku HTML, aby uzyskać przyjazny oku wygląd strony wykorzystaliśmy CSS oraz Bootstrap. Funkcjonalność strony została napisana w JavaScripcie przy użyciu biblioteki jQuery. Za pomocą Ajax zaimplementowano także pobór danych z pliku. Na Listingu 7 pokazany został kod w języku HTML odpowiedzialny za ułożenie strony do wyświetlania wykresów pogodowych.

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <title>Pogoda: cisnienie, temperatura i wilgotnosc</title>
        <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/jquery@3.2.1/dist/jquery.min.js"></script>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
        <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
        <script type="text/javascript" src="pogoda.js"></script>
        <link rel="stylesheet" type="text/css" href="styles/pogoda.css">
    </head>
    <body>
        <h1 class="text-center">Pogoda: temperatura, ciśnienie i wilgotność</h1>
        <div id="ramka" class="container border border-primary rounded datagrabber">
            <ul class="nav">
                <li><button id="start" type="button" class="btn btn-primary menuitem">Start</button></li>
                <li><button id="stop" type="button" class="btn btn-primary menuitem">Stop</button></li>
                <li><div class="border border-primary rounded menuitem"><span>Czas próbkowania: <input type = "number" value = "100" step="10" id="sampletime"></input> ms</span></div></li>
                <li><div class="border border-primary rounded menuitem"><span>Maksymalna liczba próbek: <input type = "number" value = "100" step="10" id="samplenumber"></input></span></div></li>
            </ul>
            <div class="chartcontainer">
                <canvas id="chart"></canvas>
            </div><br/>
            <div class="chartcontainer1">
                <canvas id="chart1"></canvas>
            </div>
            <div class="chartcontainer2">
                <canvas id="chart2"></canvas>
            </div>
            </div>
        </div>
    </body>
</html>
```

Listing 7 Kod HTML formularza wykresu pogody

Do realizacji funkcjonalności użyty został język JavaScript. W wielu fragmentach kod jest bardzo podobny, jak na przykład funkcja do odczytu danych z pliku widoczna na Listingu 8. Użyto tu metody Ajax z typem GET by z podanego w funkcji adresu serwera pobrać dane dotyczące IP oraz portu. Użyliśmy także Ajaxa z typem POST służącego do wysłania informacji na serwer, co pokazuje Listing 9. Jest to fragment na którym przekazujemy serwerowi informacje na temat tego, którą diodę ma zapalić oraz na jaki kolor. Zmienna dane przechowuje informacje o wybranej przez użytkownika diodzie i przekazuje te dane do skryptu php widocznego na Listingu 6, który zapala odpowiednią diodę na nakładce SenseHat. Wyświetlanie tekstu jest niemal analogiczne, zamiast podawania współrzędnych diody podajemy tekst, a w skrypcie python jest to różnica polegająca na użyciu innej metody, show_message zamiast set_pixel.

```
function odczyt_adresu() {
    $.ajax({
        type: 'GET',
        url: "http://"+a+"/ustawienia.json",
        success: function(response) {
            zmienna = response.adres_serwera;
            port = parseFloat(response.port_serwera);
```

```
        }
    });
}
```

Listing 8 Funkcja Ajax do wczytania danych z pliku

```
function dioda(){
    Dane = {
    poziom: $('#poziom').val(),
    pion: $('#pion').val(),
    kolor: $('#kolor').val()
    }
    $.ajax({
        type: 'POST',
        url: 'http://'+zmienna+'/led_one.php',
        data: Dane,
        success: function(respons) {
            console.log(Dane);
        }
    });
}
```

Listing 9 Funkcja Ajax do uruchomienia skryptu PHP zapalającego diodę

Funkcja powtarzającą się w każdym skrypcie javascript jest funkcja pokazana na Listingu 10. Odpowiada ona za wykonanie funkcji podczas uruchamiania okna w przeglądarce.

```
$(document).ready(() => {
    odczyt_adresu();
    $("#btn").click(dioda);
    $("#wyswietl").click(napis);
});
```

Listing 10 Funkcja wykonywana w momencie włączenia strony

W przypadku stron z generowanymi wykresami, mamy do czynienia z bardziej skomplikowaną implementacja. Możemy wyodrębnić fragment widoczny na Listingu 11 do pobierania nowych danych z pliku. Funkcję do dodawania nowych próbek do wykresu widocznych na Listingu 12, oraz do usuwania najstarszych danych w celu uniknięcia zbytniego wykorzystania pamięci, co możemy zobaczyć na Listingu 13. Wymagane jest także odpowiednie ukształtowanie wykresów, ich parametrów, kolorów, podpisów osi a następnie ich inicjalizacja, kod realizujący to możemy zobaczyć na Listingu 14.

```
function ajaxJSON() {
    $.ajax('http://'+zmienna+'/dane_pomiarowe.json', {
        type: 'GET', dataType: 'json',
        success: function(responseJSON, status, xhr) {
            addData1(+responseJSON.Temperature);
            addData2(+responseJSON.Pressure);
            addData3(+responseJSON.Humidity);
        }
    });
}
```

Listing 11 Funkcja Ajax pobierająca dane o pogodzie

```
function addData1(y){
    if(ydata1.length > maxSamplesNumber)
    {
        removeOldData();
        lastTimeStamp += sampleTimeSec;
        xdata.push(lastTimeStamp.toFixed(4));
    }
```

```
    ydata1.push(y);
    chart.update();
}
```

Listing 12 Funkcja do dodawania nowych próbek do wykresu

```
function removeOldData(){
    xdata.splice(0,1);
    ydata1.splice(0,1);
    ydata2.splice(0,1);
    ydata3.splice(0,1);
}
```

Listing 13 Funkcja usuwająca najstarszą próbkę

```
function chartInit()
{
    xdata = [...Array(maxSamplesNumber).keys()];
    xdata.forEach(function(p, i) {this[i] = (this[i]*sampleTimeSec).toFixed(4);}, x
data);
    lastTimeStamp = +xdata[xdata.length-1];
    ydata1 = [];
    ydata2 = [];
    ydata3 = [];
    chartContext = $("#chart")[0].getContext('2d');
    chartContext1 = $("#chart1")[0].getContext('2d');
    chartContext2 = $("#chart2")[0].getContext('2d');
    chart = new Chart(chartContext, {
        type: 'line',
        data: {
            labels: xdata,
            datasets: [{
                fill: false,
                label: 'temperatura',
                backgroundColor: 'rgb(255, 0, 0)',
                borderColor: 'rgb(255, 0, 0)',
                data: ydata1,
                lineTension: 0
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            animation: false,
            scales: {
                yAxes: [{
                    ticks: {
                    suggestedMin: 0,
                    suggestedMax: 1
                },
                    scaleLabel: {
                        display: true,
                        labelString: 'temperatura [oC]'
                    }
                }],
                xAxes: [{
                    scaleLabel: {
                        display: true,
                        labelString: 'Time [s]'
                    }
                }]
            }
        }
});
```

Listing 14 Fragment kodu odpowiadający za utworzenie wykresu

2.3 Aplikacja Mobilna

Aplikacja mobilna została stworzona w programie Android Studio z wykorzystaniem języka programowania Java. Graficzny interfejs zrealizowany został z wykorzystaniem „xml", w znacznej większości metodą drag and drop, dlatego też w sprawozdaniu pominięte zostały kody odpowiadające za wygląd i parametryzację poszczególnych widoków – podejrzeć można je w dołączonym projekcie, natomiast same layouty zaprezentowane zostały w dołączonym filmie. Pokazywane poniżej fragmenty kodu nie są rzecz jasna wszystkimi potrzebnymi do działania aplikacji. Początkowa deklaracja zmiennych, inicjalizacja elementów interfejsu graficznego, prosta parametryzacja(np. wykresów), funkcje które swoją nazwą jasno informują jakie wykonują zadanie w większości zostały pominięty w omówieniu. W razie chęci ich przejrzenia odsyłamy do pełnego projektu, który też został odpowiednio skomentowany, w celu zwiększenia jego czytelności(komentarze ze standardu doxygen został usunięte w listingach by skrócić ich długość).

2.3.1 Prezentacja wykresów

W aplikacji mamy dostępne dwa widoki z wykresami czasowymi – wykresy pogodowe oraz wykresy położenia, w obu przypadkach możliwy jest wybór jednostki, ustawienie czasu próbkowania, a także adresu IP. Sam kod odpowiedzialny za działanie wykresów oraz zmianę wcześniej wymienionych parametrów jest analogiczny dla obu widoków dlatego w sprawozdaniu ograniczymy prezentację do jednego z nich(kąty RPY).

```
1.  public void btnsRpy_onClick(View v) {
2.          switch (v.getId()) {
3.              case R.id.goToRpyOptionsBtn: {
4.                  if (requestTimer != null)
5.                      dialogAlertShow();
6.                  else
7.                      openRpyOptions();
8.                  break;
9.              }
10.             case R.id.startRpyChartsBtn: {
11.                 startRequestTimer();
12.                 break;
13.             }
14.             case R.id.stopRpyChartsBtn: {
15.                 stopRequestTimerTask();
16.                 break;
17.             }
18.             default: {
19.                 // do nothing
20.             }
21.         }
22.     }
```

Listing 15 Funkcja reagująca na wciśnięcia przycisków w widoku

Powyższa funkcja ukazuje logikę działania aplikacji przy wykrycie wciśnięcia konkretnego przycisku w interfejsie wykresów. Poniżej kolejne listingi 16, 17, 18 pokazują działanie wywoływanych wyżej funkcji.

W momencie gdy chcemy włączyć opcje, a wykres działa, to ukazuje się okno informujące o przerwaniu pobierania danych, poniżej widoczna jest parametryzacja tego okna oraz funkcje wywołujące się po naciśnięciu konkretnego przycisku okna.

```
1.  public void dialogAlertShow() {
2.          configAlertDialog = new AlertDialog.Builder(RpyActivity.this);
3.          configAlertDialog.setTitle("Pobieranie danych zostanie zatrzymane");
```

```
        configAlertDialog.setIcon(android.R.drawable.ic_dialog_alert);
        configAlertDialog.setPositiveButton("OK", new DialogInterface.OnClickListen
er() {
            public void onClick(DialogInterface dialog, int which) {
                stopRequestTimerTask();
                openRpyOptions();
            }
        });
        configAlertDialog.setNegativeButton("Anuluj", new DialogInterface.OnClickLi
stener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
            }
        });
        configAlertDialog.setCancelable(false);
        configAlertDialog.show();
    }
```

Listing 16 Funkcja tworząca okno z komunikatem

Włączenie opcji poprzedzone jest poprzez stworzenie paczki z danymi o czasie próbkowania, jednostce oraz IP w celu umożliwienia zmiany tych danych w widoku opcji, paczka ta jest dalej odbierana i odczytywana w activity obsługującym opcje wykresów RPY zaprezentowanym w dalszej części sprawozdania.

```
private void openRpyOptions() {
        //Utworzenie nowej intencji oraz paczki danych
        Intent openConfigIntent = new Intent(RpyActivity.this, RpyOptionsActivity.c
lass);
        Bundle configBundle = new Bundle();

        //Umieszczenie w paczce informacji o IP, TP oraz jednostce
        configBundle.putString(CommonData.CONFIG_IP_ADDRESS, ipAddress);
        configBundle.putInt(CommonData.CONFIG_SAMPLE_TIME, sampleTime);
        configBundle.putString(CommonData.CONFIG_RPY_UNIT, rpyUnit);

        //Umieszczenie paczki w intencji oraz uruchomienie activity, jako ForResult

        openConfigIntent.putExtras(configBundle);
        startActivityForResult(openConfigIntent, CommonData.REQUEST_CODE_CONFIG);
    }
```

Listing 17 Funkcja uruchamiająca activity z opcjami

```
private void startRequestTimer() {
        if (requestTimer == null) {
            //Utworzenie nowego timera
            requestTimer = new Timer();
            //Inicjalizacja zadania TimerTask
            initializeRequestTimerTask();
            requestTimer.schedule(requestTimerTask, 0, sampleTime);
        }
    }
private void stopRequestTimerTask() {
        //Zatrzymanie timera, jeśli istnieje
        if (requestTimer != null) {
            requestTimer.cancel();
            requestTimer = null;
            requestTimerFirstRequestAfterStop = true;
        }
    }
```

Listing 18 Funkcje uruchamiające oraz zatrzymujące timer

Kolejną istotną funkcją jest funkcja wysyłająca zapytanie GET na serwer, została ona stworzona zgodnie z biblioteką Volley(standardowa implementacja), istotniejsze są natomiast wywoływane funkcje w przypadku uzyskania odpowiedzi lub jej braku, które to zostały pokazane na fragmencie kodu 19. Na początku widzimy funkcję wyświetlającą prosty komunikat typu toast w przypadku wystąpienia błędu, natomiast dalej mamy funkcję która wywołuje odczytywanie danych o kątach(zaprezentowane na listingu 20), zapisuje informacje do odpowiedniej zmiennej, a następnie aktualizuje wykres o kolejną próbkę.

```
1.  private void errorHandling(int errorCode) {
2.          Toast errorToast = Toast.makeText(this, "ERROR: "+errorCode, Toast.LENGTH_S
    HORT);
3.          errorToast.show();
4.      }
5.
6.
7.  private void responseHandling(String response) {
8.          if (requestTimer != null) {
9.              //Pobranie informacji o aktualnym czasie
10.             long requestTimerCurrentTime = SystemClock.uptimeMillis();
11.             requestTimerTimeStamp += getValidTimeStampIncrease(requestTimerCurrentT
    ime);
12.
13.             //Pobranie danych z pliku JSON
14.             if (getRawDataFromResponse(response).size() != 0) {
15.                 rollRawData = getRawDataFromResponse(response).get(0);
16.                 pitchRawData = getRawDataFromResponse(response).get(1);
17.                 yawRawData = getRawDataFromResponse(response).get(2);
18.             }
19.             //Aktualizacja wykresu, gdy próbki są liczbami
20.             if (isNaN(rollRawData) || isNaN(pitchRawData) || isNaN(yawRawData)) {
21.                 errorHandling(CommonData.ERROR_NAN_DATA);
22.             }
23.             else {
24.
25.                 //Aktualizacja serii
26.                 double timeStamp = requestTimerTimeStamp / 1000.0; // [sec]
27.                 boolean scrollGraph = (timeStamp > dataGraphMaxX); //Skrollowanie
    gdy czas jest większy niż na osi x
28.                 rollDataSeries.appendData(new DataPoint(timeStamp, rollRawData), sc
    rollGraph, dataGraphMaxDataPointsNumber);
29.                 pitchDataSeries.appendData(new DataPoint(timeStamp, pitchRawData),
    scrollGraph, dataGraphMaxDataPointsNumber);
30.                 yawDataSeries.appendData(new DataPoint(timeStamp, yawRawData), scro
    llGraph, dataGraphMaxDataPointsNumber);
31.                 //Odświeżanie widoku
32.                 rpyDataGraph.onDataChanged(true, true);
33.             }
34.
35.             //Zapamiętanie ostatniej próbki
36.             requestTimerPreviousTime = requestTimerCurrentTime;
37.         }
38.     }
```

Listing 19 Funkcje obsługujące odpowiedź serwera

```
1.  private List<Double> getRawDataFromResponse(String response) {
2.          JSONObject jObject;
3.          rpyValuesList = new ArrayList<>();
4.
5.          try {
6.              jObject = new JSONObject(response);
7.          } catch (JSONException e) {
8.              e.printStackTrace();
9.              return rpyValuesList;
10.         }
```

```
11.
12.         try {
13.             double roll = (double)jObject.get("Roll");
14.             double pitch = (double)jObject.get("Pitch");
15.             double yaw = (double)jObject.get("Yaw");
16.             rpyValuesList.add(roll);
17.             rpyValuesList.add(pitch);
18.             rpyValuesList.add(yaw);
19.         } catch (JSONException e) {
20.             e.printStackTrace();
21.         }
22.         return rpyValuesList;
23.     }
```

Listing 20 Funkcja odczytująca dane o kątach

Powyższa funkcja w prosty sposób odczytuje dane o położeniu kątowym z pliku json oraz zwraca wszystko w postaci listy z liczbami.

Kolejne dwa listingi pochodzą już z kolejnego activity odpowiedzialnego za realizację opcji dla wykresów. Pierwszy z nich jest fragmentem metody onCreate() i mamy tutaj pokazany wycinek jak przekazywane są dane na temat aktualnego adresu IP oraz czasu próbkowania pomiędzy widokiem wykresu, a jego opcji oraz inicjalizacja spinnera odpowiedzialnego za wybór jednostki.

```
1.  //Inicjalizacja pól editText umieszczenie w nich informacji pobranych z intencji
2.  //Definicja aktualnego IP oraz TP
3.  ipEditText = findViewById(R.id.userInputRpyIP);
4.  String ip = configBundle.getString(CommonData.CONFIG_IP_ADDRESS, CommonData.DEFAULT
    _IP_ADDRESS);
5.
6.  ipEditText.setText(ip);
7.  sampleTimeEditText = findViewById(R.id.userInputRpyTp);
8.
9.  int tp = configBundle.getInt(CommonData.CONFIG_SAMPLE_TIME, CommonData.DEFAULT_SAMP
    LE_TIME);
10.
11. sampleTimeEditText.setText(Integer.toString(tp));
12.
13. //Inicjalizacja spinnera
14. Spinner rpyUnitPicked = findViewById(R.id.rpyUnitPicked);
15. ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this, R.array.
    rpyUnit, android.R.layout.simple_spinner_item);
16.
17. adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
18.
19. rpyUnitPicked.setAdapter(adapter);
20.
21. //Ustawienie wybranej ze spinnera jednostki
22. rpyUnitPicked.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

23.             @Override
24.             public void onItemSelected(AdapterView<?> parent, View view, int positi
    on, long id) {
25.                 rpyUnit = parent.getItemAtPosition(position).toString();
26.             }
27.
28.             @Override
29.             public void onNothingSelected(AdapterView<?> parent) {
30.                 //nothing
31.             }
32.         });
```

Listing 21 Fragment metody onCreate() w widoku opcji

Drugi fragment natomiast ukazuje funkcję onBackPressed() która wywołuje się po wyjściu z opcji(cofnięciu do widoku wykresów). W jej wnętrzu przekazywane do „intencji" są informacje o wybranych parametrach przez użytkownika z zabezpieczeniem przed pozostawieniem pustego pola(wtedy wybrana zostaje wartość domyślna). Intencja ta zostaje później odczytywana w odpowiedniej funkcji już w activity odpowiedzialnym za same wykresy(listing 22).

```
1.  @Override
2.      public void onBackPressed() {
3.          Intent intent = new Intent();
4.          if (!ipEditText.getText().toString().equals("") && !sampleTimeEditText.getT
    ext().toString().equals(""))
5.          {
6.              intent.putExtra(CommonData.CONFIG_IP_ADDRESS, ipEditText.getText().toSt
    ring());
7.              intent.putExtra(CommonData.CONFIG_SAMPLE_TIME, sampleTimeEditText.getTe
    xt().toString());
8.          }
9.          else
10.         {
11.             intent.putExtra(CommonData.CONFIG_IP_ADDRESS, CommonData.DEFAULT_IP_ADD
    RESS);
12.             intent.putExtra(CommonData.CONFIG_SAMPLE_TIME, "500");
13.         }
14.         intent.putExtra(CommonData.CONFIG_RPY_UNIT, rpyUnit);
15.         setResult(RESULT_OK, intent);
16.         finish();
17.     }
```

Listing 22 Funkcja onBackPressed dla widoku opcji

```
1.  @Override
2.      protected void onActivityResult(int requestCode, int resultCode, Intent dataInt
    ent) {
3.          super.onActivityResult(requestCode, resultCode, dataIntent);
4.          if ((requestCode == CommonData.REQUEST_CODE_CONFIG) && (resultCode == RESUL
    T_OK)) {
5.
6.              //Pobranie intencji o ustawionym IP, jednostce oraz TP w opcjach
7.              ipAddress = dataIntent.getStringExtra(CommonData.CONFIG_IP_ADDRESS);
8.              rpyUnit = dataIntent.getStringExtra(CommonData.CONFIG_RPY_UNIT);
9.              String sampleTimeText = dataIntent.getStringExtra(CommonData.CONFIG_SAM
    PLE_TIME);
10.             assert sampleTimeText != null;
11.             sampleTime = Integer.parseInt(sampleTimeText);
12.         }
13.     }
```

Listing 23 Odczyt informacji o czasie próbkowania, jednostkach oraz IP

2.3.2 Prezentacja danych w formie listy

W przypadku tego widoku najistotniejszą funkcją jest ta poniżej, która umożliwia cykliczne odczytywanie oraz aktualizowanie wartości wyświetlanych na ekranie. Wykorzystujemy tutaj handler którego to metoda postDelayed() umożliwia wywoływanie funkcji typu GetRequest co określony czas, będący naszym ustalonym czasem próbkowania.

```
1.  @Override
2.      protected void onResume() {
3.          handler.postDelayed(runnable = new Runnable() {
4.              public void run() {
5.                  handler.postDelayed(runnable, sampleTime);
6.                  sendGetRequestWeather();
7.                  sendGetRequestRpy();
8.              }
9.          }, sampleTime);
10.         super.onResume();
11.     }
```

Listing 24 Cykliczne uruchamianie pobierania danych z serwera

Poniżej prezentuję jedną z funkcji wywoływanych przy uzyskaniu odpowiedzi z serwera, jako że w przypadku wykresów zaprezentowane zostało położenie kątowe, tak tutaj wybrałem fragment odpowiedzialny za wyświetlanie informacji o pogodzie.

```
1.  private void responseHandlingWeather(String response) {
2.
3.          if (getRawDataFromResponseWeather(response).size() != 0) {
4.              temperatureRawData = round(getRawDataFromResponseWeather(response).get(
    0), 2);
5.              pressureRawData = round(getRawDataFromResponseWeather(response).get(1),
     2);
6.              humidityRawData = round(getRawDataFromResponseWeather(response).get(2),
     2);
7.          }
8.
9.          if (isNaN(temperatureRawData) || isNaN(pressureRawData) || isNaN(humidityRa
    wData)) {
10.             errorHandling(CommonData.ERROR_NAN_DATA);
11.         }
12.         else {
13.             final String temperatureRawDataString = Double.toString(temperatureRawD
    ata);
14.             final String pressureRawDataString = Double.toString(pressureRawData);

15.             final String humidityRawDataString = Double.toString(humidityRawData);

16.             temperatureValue.setText(temperatureRawDataString);
17.             pressureValue.setText(pressureRawDataString);
18.             humidityValue.setText(humidityRawDataString);
19.         }
20.     }
```

Listing 25 Obsługa odpowiedzi z serwera

Najpierw do zmiennych zapisywane są odpowiednie dane(zaokrąglone do dwóch miejsc po przecinku), następnie konwertowane są do typu string oraz zapisywane w przeznaczonych do tego polach tekstowych. Funkcja odpowiedzialna za odczytywane danych z pliku json jest bardzo podobna do tej przedstawionej w punkcie o prezentacji wykresów z jedyną różnicą, że w przypadku danych o pogodzie znajduje się w tejże funkcji sprawdzenie aktualnie wybranej jednostki, by odczytać odpowiednią wartości z obiektu json.

Omawiany w tym punkcie widok umożliwia lokalną, chwilową(do momentu wyjścia z widoku) zmianę IP oraz TP. Po naciśnięciu jedynego umieszczonego tam przycisku wywoływana jest poniższa funkcja, która ustawia nowo wybrany adres IP oraz czas próbkowania, natomiast gdy użytkownik zostawił jedno z pół pustych to wybrane zostają wartości domyślne oraz wyświetlany jest komunikat z informacją o tym.

```
1.  public void setTpAndIp(View v)
2.      {
3.          if (v.getId() == setIpAndTpAllData.getId())
4.          {
5.              if (!ipAdressEditText.getText().toString().equals("") && !sampleTimeEdi
    tText.getText().toString().equals(""))
6.              {
7.                  ipAddress = ipAdressEditText.getText().toString();
8.                  sampleTime = Integer.parseInt(sampleTimeEditText.getText().toString
    ());
9.              }
10.             else
11.             {
12.                 ipAddress = CommonData.DEFAULT_IP_ADDRESS;
13.                 sampleTime = CommonData.DEFAULT_SAMPLE_TIME;
14.                 Toast.makeText(this, "Nie podałeś IP lub TP, ustawiono wartości dom
    yślne", Toast.LENGTH_LONG).show();
15.             }
16.             sampleTimeEditText.setText(Integer.toString(sampleTime));
17.             ipAdressEditText.setText(ipAddress);
18.         }
19.     }
```

Listing 26 Ustawienie lokalnych wartości IP oraz czasu próbkowania

2.3.3 Obsługa matrycy LED

W naszej aplikacji mobilnej możliwe jest wykorzystanie matrycy LED w celu zapalenia jednej wybranej diody na jeden z możliwych trzech kolorów lub wyświetlenie „przepływającego” przez matrycę tekstu. Ponadto możliwa jest tutaj również szybka zmiana IP. Logika aplikacji w przypadku dwóch wspomnianych aktywności jest bardzo podobna, różniąca się jedynie w przekazywanych do pliku php danych, poniżej przedstawione zostały fragmenty kodu odpowiedzialne za wyświetlanie na matrycy tekstu.

```
1.  public Map<String, String> getLedDisplayParams() {
2.          Map<String, String> params = new HashMap<>();
3.
4.          String ledMsgText = ledMsg.getText().toString();
5.          params.put("text", ledMsgText);
6.
7.          String ledColorText = ledColor.getText().toString();
8.          params.put("color", ledColorText);
9.
10.         return params;
11.     }
```

Listing 27 Odczyt danych wpisanych w pola tekstowe przez użytkownika

Pierwsza z funkcji na listingu 27 przedstawia wczytanie informacji z pól tekstowych oraz zapisanie ich do obiektu typu HashMap. Z tejże funkcji korzystamy w zapytaniu POST, którego to kod został zaprezentowany na kolejnym listingu poniżej. Na serwer z wykorzystaniem wybranego przez użytkownika IP wysyłane są dane(tekst do wyświetlenia oraz kolor), które później odpowiednio wykorzystywane są przez skrypty PHP oraz Python na samym serwerze.

```
1.  public void sendControlRequestTxt(View v)
2.      {
3.          if (!ipAddressEditText.getText().toString().equals(""))
4.          {
5.              ipAddress = ipAddressEditText.getText().toString();
6.          }
7.          else
8.          {
9.              ipAddress = CommonData.DEFAULT_IP_ADDRESS;
10.             Toast.makeText(this, "Nie podałeś IP, wybrano domyślne!", Toast.LENGTH_
    LONG).show();
11.         }
12.         ipAddressEditText.setText(ipAddress);
13.
14.         //Utworzenie zapytania typu POST, zdefiniowanie działania przy odpowiedzi o
    raz jej braku
15.         StringRequest postRequest = new StringRequest(Request.Method.POST, getURL(i
    pAddress),
16.                 new Response.Listener<String>()
17.                 {
18.                     @Override
19.                     public void onResponse(String response) {
20.                         Log.d("Response", response);
21.                     }
22.                 },
23.                 new Response.ErrorListener()
24.                 {
25.                     @Override
26.                     public void onErrorResponse(VolleyError error) {
27.                         String msg = error.getMessage();
28.                         if (msg != null)
29.                             Log.d("Error.Response", msg);
30.                     }
31.                 }
32.         ) {
33.             //Definicja czynności do wykonania przy uzyskaniu odpowiedzi z serwera

34.             @Override
35.             protected Map<String, String> getParams() {
36.                 return getLedDisplayParams();
37.             }
38.         };
39.         postRequest.setRetryPolicy(new DefaultRetryPolicy(2500, 0,
40.                 DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
41.
42.         //Dodanie zapytania do kolejki
43.         queue.add(postRequest);
44.     }
```

Listing 28 Wysłanie zapytania typu POST na serwer

2.3.4 Położenie joysticka

Widok prezentujący położenie joysticka jest w zasadzie połączeniem widoku wykresu oraz danych w postaci listy, by cyklicznie wysyłać zapytanie na serwer wykorzystany został handler w analogiczny sposób jak w przypadku listy z wartościami, natomiast dla prezentacji graficznej położenia joysticka została wykorzystana ta sama biblioteka z wykresami, z tą różnicą, że należało skorzystać z wykresu punktowego, odpowiednio kasując poprzedni punkt po pojawieniu się nowego impulsu(przesunięcia joysticka). Poniżej więc zaprezentowane zostały jedynie dwie funkcje na listingu 29 mamy obsługę handlera, natomiast na listingu 30 mamy działanie w przypadku uzyskania odpowiedzi z serwera.

```
1.  @Override
2.      protected void onResume() {
3.          handler.postDelayed(runnable = new Runnable() {
4.              public void run() {
5.                  handler.postDelayed(runnable, delay);
6.                  sendGetRequest();
7.              }
8.          }, delay);
9.          super.onResume();
10.     }
```

Listing 29 Cykliczne odpytywanie serwera

```
1.  private void responseHandling(String response) {
2.
3.          joystickDataSeries.resetData(new DataPoint[]{});
4.
5.          int xAxisRawData = getRawDataFromResponse(response).get(0);
6.          int yAxisRawData = getRawDataFromResponse(response).get(1);
7.          int centerRawData = getRawDataFromResponse(response).get(2);
8.
9.          if (isNaN(xAxisRawData) || isNaN(yAxisRawData) || isNaN(centerRawData)) {
10.             errorHandling(CommonData.ERROR_NAN_DATA);
11.         }
12.
13.         else {
14.
15.             joystickDataSeries.appendData(new DataPoint(xAxisRawData, yAxisRawData)
    , false, dataGraphMaxDataPointsNumber);
16.             joystickDataGraph.onDataChanged(true, true);
17.
18.             final String centerRawDataString = Integer.toString(centerRawData);
19.             centerClickNb.setText(centerRawDataString);
20.         }
21.     }
```

Listing 30 Obsługa odpowiedzi z serwera

2.3.5 Globalna zmiana ustawień

W większości wcześniej omówionych widoków zaprezentowana została możliwość zmiany adresu IP oraz czasu próbkowania. Jednakże zmiany te miały jedynie charakter lokalny i po zamknięciu danego widoku zostały zapominane. W celu zmiany np. adresu IP na dłuższy czas został dodany widok ustawień globalnych, który korzysta z SharedPreferences, dzięki czemu nawet po zamknięci aplikacji zapamiętywane są wybrane preferencje użytkownika. Dzięki takiemu podejściu możliwe jest ustawienie na stałe parametrów IP oraz TP, które chciałby mieć użytkownik, natomiast w poszczególnych widokach możliwa jest lokalna zmiana np. czasu próbkowania, gdy chwilowo chcemy go chociażby zwiększyć.

Na poniższych listingach zaprezentowane zostały funkcje znajdujące się w activity odpowiedzialnym za globalne ustawienia.

Pierwsza z funkcji to znajdująca się w każdym widoku onCreate(), gdzie znajduje się inicjalizacja wszelakich elementów(listing 31), funkcja wczytująca aktualne preferencję(listing 32) oraz „nasłuchiwacz” wciśnięcia przycisku zapisującego wybrane ustawienia(funkcja na listingu 33).

```
1.  @Override
2.      protected void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.activity_settings);
5.
6.          //Inicjalizacja preferencji, pól tekstowych oraz przycisku
7.          userSettings = getSharedPreferences("userPref", Activity.MODE_PRIVATE);
8.          userInputSettingsIp = findViewById(R.id.userInputSettingsIp);
9.          userInputSettingsTp = findViewById(R.id.userInputSettingsTp);
10.         setPrefBtn = findViewById(R.id.setPrefBtn);
11.
12.         //Wczytanie aktualnych preferencji
13.         loadPref();
14.
15.         //Zapisanie preferencji wpisanych w polach tekstowych
16.         setPrefBtn.setOnClickListener(new View.OnClickListener() {
17.             @Override
18.             public void onClick(View v) {
19.                 savePref();
20.             }
21.         });
22.     }
```

Listing 31 Funkcja onCreate() z inicjalizacją elementów

Poniższa funkcja wczytuje aktualne preferencje(IP, TP) oraz umieszcza je w polach EditText

```
1.  private void loadPref() {
2.          ipAddressPref = userSettings.getString(CommonData.CONFIG_IP_ADDRESS, Common
    Data.DEFAULT_IP_ADDRESS);
3.          sampleTimePref = userSettings.getInt(CommonData.CONFIG_SAMPLE_TIME, CommonD
    ata.DEFAULT_SAMPLE_TIME);
4.          userInputSettingsIp.setText(ipAddressPref);
5.          userInputSettingsTp.setText(Integer.toString(sampleTimePref));
6.      }
```

Listing 32 Funkcja wczytująca aktualne preferencje

Natomiast kolejna funkcja jest odpowiedzialna za zapisanie wybranych preferencji, jak w przypadku wcześniejszych widoków mamy tutaj zabezpieczenie przed zostawieniem przez użytkownika pustego pola.

```
1.  private void savePref() {
2.          SharedPreferences.Editor editor = userSettings.edit();
3.          if (!userInputSettingsIp.getText().toString().equals("") && !userInputSetti
    ngsTp.getText().toString().equals(""))
4.          {
5.              editor.putString(CommonData.CONFIG_IP_ADDRESS, userInputSettingsIp.getT
    ext().toString());
6.              editor.putInt(CommonData.CONFIG_SAMPLE_TIME, Integer.parseInt(userInput
    SettingsTp.getText().toString()));
7.          }
8.          else
9.          {
10.             editor.putString(CommonData.CONFIG_IP_ADDRESS, CommonData.DEFAULT_IP_AD
    DRESS);
11.             editor.putInt(CommonData.CONFIG_SAMPLE_TIME, CommonData.DEFAULT_SAMPLE_
    TIME);
12.         }
13.
14.         editor.apply();
15.     }
```

Listing 33 Funkcja zapisująca preferencje

W celu wykorzystywania SharedPreferences w każdym z widoków w funkcji onCreate() należało zainicjalizować preferencje. Inicjalizacja następowała praktycznie zawsze z wykorzystaniem następującego fragmentu kodu.

```
1.  userSettings = getSharedPreferences("userPref", Activity.MODE_PRIVATE);
2.  String ipAddressPref = userSettings.getString(CommonData.CONFIG_IP_ADDRESS, CommonD
    ata.DEFAULT_IP_ADDRESS);
3.  int sampleTimePref = userSettings.getInt(CommonData.CONFIG_SAMPLE_TIME, CommonData.
    DEFAULT_SAMPLE_TIME);
4.  ipAddress = ipAddressPref;
5.  sampleTime = sampleTimePref;
```

Listing 34 Przykładowa inicjalizacja SharedPreferences w różnych widokach

2.3.6 System kontroli wersji

Aplikacja była tworzona z wykorzystaniem systemu kontroli wersji GIT oraz strony internetowej github.com. Repozytorium z aplikacją można znaleźć pod adresem: https://github.com/JedrzejGrzebisz/Aveg

2.4 APLIKACJA DESKTOPOWA

Aplikacja desktopowa została napisana w programie Visual Studio. Za logikę odpowiadają funkcje napisane w języku C#, natomiast za opis interfejsu – język XAML. Podczas tworzenia został wykorzystany wzorzec projektowy zapewniający separację interfejsu użytkownika od logiki aplikacji. Jako system kontroli wersji wykorzystałem SVN, wykorzystując do jego obsługi interfejs TortoiseSVN. Przy tworzeniu aplikacji wykorzystany został przykładowy projekt udostępniony przez Prowadzącego zajęcia.

2.4.1 Prezentacja wykresów

```
1.  DataPlotModel = new PlotModel { Title = "RPY" };
2.
3.  DataPlotModel.Axes.Add(new LinearAxis()
4.  {
5.      Position = AxisPosition.Bottom,
6.      Minimum = 0,
7.      Maximum = config.XAxisMax,
8.      Key = "Horizontal",
9.      Unit = "sec",
10.     Title = "Time"
11. });
12. DataPlotModel.Axes.Add(new LinearAxis()
13. {
14.     Position = AxisPosition.Left,
15.     Minimum = 0,
16.     Maximum = 360,
17.     Key = "Vertical",
18.     Unit = "°",
19.     Title = "Position"
20. });
21.
22. DataPlotModel.Series.Add(new LineSeries() { Title = "Roll", Color = OxyColor.Parse(
    "#FFFF0000") });
23. DataPlotModel.Series.Add(new LineSeries() { Title = "Pitch", Color = OxyColor.Parse
    ("#006400") });
24. DataPlotModel.Series.Add(new LineSeries() { Title = "Yaw", Color = OxyColor.Parse("
    #0000FF") });
25.
26. StartButton = new ButtonCommand(StartTimer);
27. StopButton = new ButtonCommand(StopTimer);
28. UpdateConfigButton = new ButtonCommand(UpdateConfig);
29. DefaultConfigButton = new ButtonCommand(DefaultConfig);
30.
31. ipAddress = config.IpAddress;
32. sampleTime = config.SampleTime;
33.
34. Server = new IoTServer(IpAddress);
```

Listing 35 Funkcja inicjalizująca

Kod umieszczony na powyższym listingu odpowiada za inicjalizację wykresu z informacjami o położeniu emulowanej nakładki SenseHat. Analogiczna funkcja jest wykorzystana przy inicjalizacji wykresów pogodowych – z różnicą, że wykresów pogodowych wyświetlana jest większa ilość.

Zamieszczona funkcja odpowiada również za inicjalizację przycisków i przypisanych im funkcji oraz za początkowe przypisanie adresu IP oraz czasu próbkowania (z pliku konfiguracyjnego).

Funkcje odpowiadające za rozpoczęcie i zakończenie generowania wykresu oraz za obsługę zmiany adresu IP i czasu próbkowania zostały stworzone na podstawie projektu przykładowego.

Po kliknięciu przycisku start, uruchamiana jest wcześniej wspomniana funkcja z timerem, która, co określony czas próbkowania, uruchamia funkcję pobierającą informacje nt. położenia z pliku JSON (który został wygenerowany przez znajdujący się na serwerze skrypt Pythona):

```
1.  dynamic resposneJson = JObject.Parse(responseText);
2.  UpdatePlot(timeStamp / 1000.0, (double)resposneJson.roll, (double)resposneJson.pitc
    h, (double)resposneJson.yaw);
```

Listing 36 Pobór danych oraz przekazanie ich do funkcji UpdatePlot

Jak widać na powyższym listingu, po pobraniu odpowiedzi z serwera uruchamiana jest funkcja UpdatePlot:

```
1.  LineSeries lineSeries1 = DataPlotModel.Series[0] as LineSeries;
2.  LineSeries lineSeries2 = DataPlotModel.Series[1] as LineSeries;
3.  LineSeries lineSeries3 = DataPlotModel.Series[2] as LineSeries;
4.
5.  lineSeries1.Points.Add(new DataPoint(t, r));
6.  lineSeries2.Points.Add(new DataPoint(t, p));
7.  lineSeries3.Points.Add(new DataPoint(t, y));
8.
9.  if (lineSeries1.Points.Count > config.MaxSampleNumber)
10. {
11.     lineSeries1.Points.RemoveAt(0);
12.     lineSeries2.Points.RemoveAt(0);
13.     lineSeries3.Points.RemoveAt(0);
14. }
15.
16. if (t >= config.XAxisMax)
17. {
18.     DataPlotModel.Axes[0].Minimum = (t - config.XAxisMax);
19.     DataPlotModel.Axes[0].Maximum = t + config.SampleTime / 1000.0; ;
20. }
21.
22. DataPlotModel.InvalidatePlot(true);
```

Listing 37 Funkcja UpdatePlot

Przekazane argumenty dodaje ona jako kolejne punkty na wykresie. Dodatkowo, gdy na wykresie zostanie osiągnięta maksymalna liczba próbek, najstarsze z nich zostaną usunięte z wykresu. Podobnie z osią opisującą czas – w momencie osiągnięcia przez wykres ostatniego argumentu na osi X, zacznie ona zmieniać swój przedział zgodnie z upływającym czasem.

Sytuacja wygląda bardzo podobnie w przypadku aktualizowania wykresów pogodowych.

2.4.2 Prezentacja danych w formie tabeli

Przy generowaniu tabeli zostały zastosowane podobne zabiegi co przy generowaniu wykresów – funkcja uruchamiana po kliknięciu przycisku start uruchamiała z określonym czasem próbkowania funkcję pobierającą informację nt. wszystkich danych pomiarowych z pliku JSON znajdującego się na serwerze. Ta natomiast uruchamia funkcję UpdateTable, której zawartość przedstawiona jest na poniższym listingu:

```
1.  Roll_info = roll;
2.  Yaw_info = yaw;
3.  Pitch_info = pitch;
4.  Temperature_info = temperature;
5.  Humidity_info = humidity;
6.  Pressure_info = pressure;
7.  xAxis_info = xAxis;
8.  yAxis_info = yAxis;
9.  Center_info = center;
```

```
10.
11. OnPropertyChanged("Roll_info");
12. OnPropertyChanged("Yaw_info");
13. OnPropertyChanged("Pitch_info");
14. OnPropertyChanged("Temperature_info");
15. OnPropertyChanged("Humidity_info");
16. OnPropertyChanged("Pressure_info");
17. OnPropertyChanged("xAxis_info");
18. OnPropertyChanged("yAxis_info");
19. OnPropertyChanged("Center_info");
```

Listing 38 Funkcja UpdateTable

Odczytane z pliku JSON wartości trafiają do ów funkcji jako argumenty i wartości zbindowanych z plikiem XAML stringów ulegają aktualizacji.

2.4.3 Obsługa matrycy LED

Na samym początku funkcji inicjalizującej znajdują się fragmenty odpowiadające za pobranie numeru wiersza, numeru kolumny, koloru oraz tekstu, jeden z ów fragmentów przedstawiony jest na poniższym listingu:

```
1.  private string _DiodaWiersz = "Podaj wiersz (0-7)";
2.  public string DiodaWiersz
3.  {
4.      get
5.      {
6.          return _DiodaWiersz;
7.      }
8.      set
9.      {
10.         if (_DiodaWiersz != value)
11.         {
12.             _DiodaWiersz = value;
13.             OnPropertyChanged("DiodaWiersz");
14.         }
15.     }
16. }
```

Listing 39 Pobór numeru wiersza

Gdy użytkownik zmieni wartość zbindowanego stringu – do zmiennej trafi wpisany przez użytkownika string.

Po wciśnięciu przycisku ustawiającego diody lub tekst, zostanie uruchomiona odpowiednia funkcja przesyłająca do serwera dane informacje. Na listingu poniżej funkcja ustawiająca diody:

```
1.  private void SetDiods()
2.  {
3.      string Data = Web.GetPost("http://192.168.56.16/LAB07_CSS_jQ/grab_diods_data.ph
    p", "postwiersz", DiodaWiersz, "postkolumna",  DiodaKolumna, "postkolor", Kolorek);

4.  }
```

Listing 40 Przesyłanie informacji na serwer

Po stronie serwera ów informacje zostają przechwycone przez skrypt PHP, który następnie przekazuje je do pliku JSON, by ostatecznie trafiły do skryptu Pythona, który zapala odpowiednią diodę.

2.4.4 Obsługa interfejsu

Wygląd interfejsu został zaczerpnięty z przekazanej aplikacji przykładowej, dlatego większość fragmentów kodu XAML oraz C# obsługujących interfejs nie zostało umieszczonych w

sprawozdaniu. Poniżej zamieszczam listing, na którym znajduje się fragment kodu obsługujący przełączanie między poszczególnymi oknami:

```
1.  private void TemperatureButton(object sender, RoutedEventArgs e)
2.  {
3.      Temperature temp = new Temperature();
4.      temp.Show();
5.      this.Close();
6.  }
```

Listing 41 Przełączenie do okna Temperature

Po kliknięciu odpowiedniego przycisku otwierane jest nowe okno z daną funkcjonalnością, natomiast obecne jest zamykane.

```
1.  private void TemperatureChecked(object sender, RoutedEventArgs e)
2.  {
3.      TempUnit.Text = "°F";
4.      CTemperaturePlotView.Visibility = Visibility.Hidden;
5.      FTemperaturePlotView.Visibility = Visibility.Visible;
6.  }
```

Listing 42 Zmiana jednostki

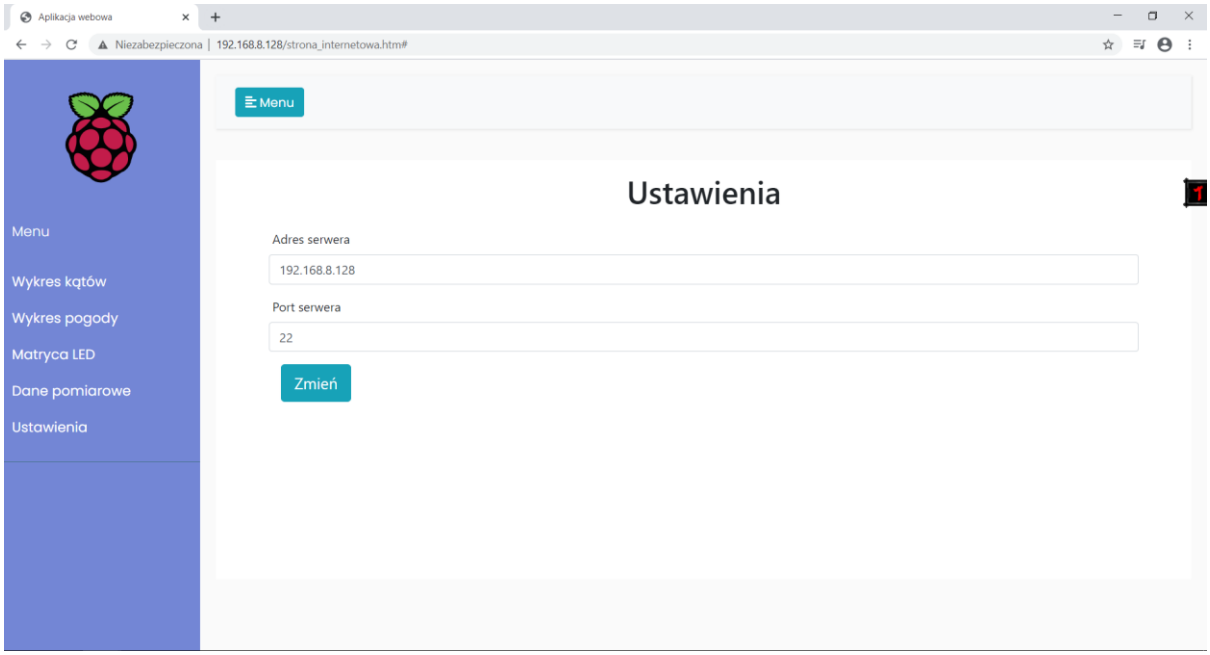
Fragment kodu powyżej odpowiada za zmianę jednostki przy temperaturze – dokładniej zmianę na stopnie Fahrenheita. W rzeczywistości po wciśnięciu przycisku zmiany jednostki chowany jest wykres przedstawiający temperaturę w stopniach Celcjusza, a pokazywany jest wykres z temperaturą w stopniach Fahrenheita. Dzięki takiemu rozwiązaniu obydwa wykresy są cały czas dostępne i jednostkę można zmieniać bez konieczności zatrzymywania wykresów.

Analogiczne funkcje zostały zastosowane do zmian pozostałych jednostek.

3 WYNIKI DZIAŁANIA

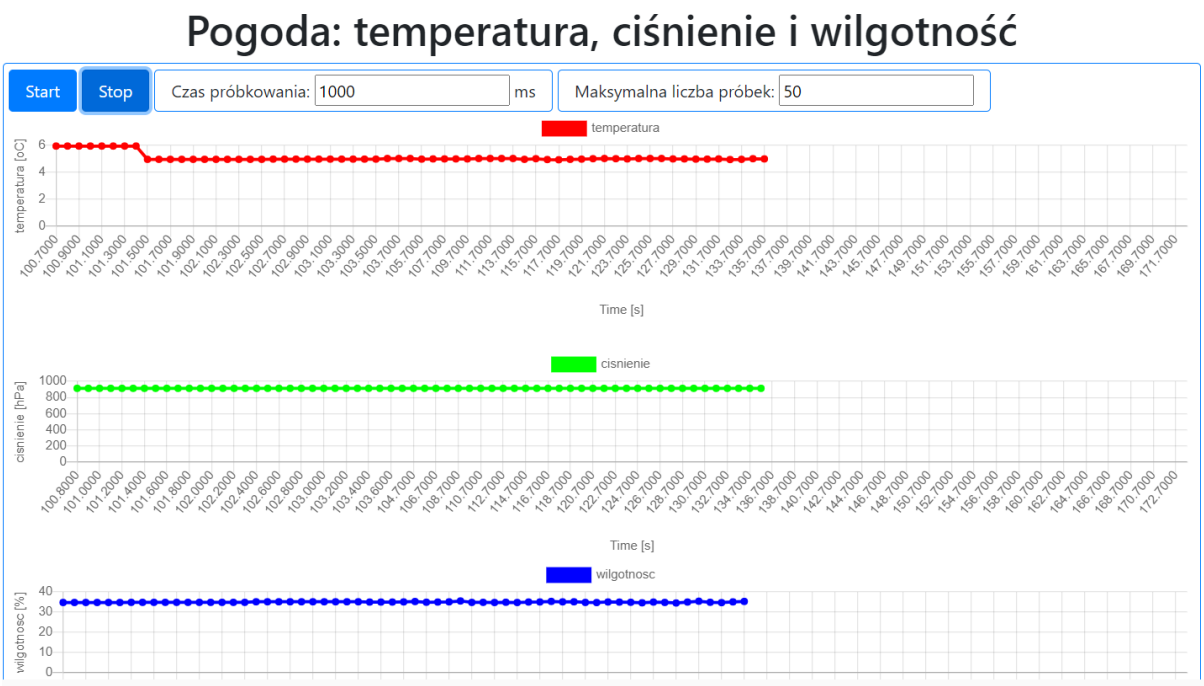
3.1 APLIKACJA WEBOWA

Na Rysunku 1 widzimy widok z całym menu i aktualnie wybrana zakładką z ustawieniami. Mamy tu dostęp do poszczególnych funkcjonalności naszej strony.

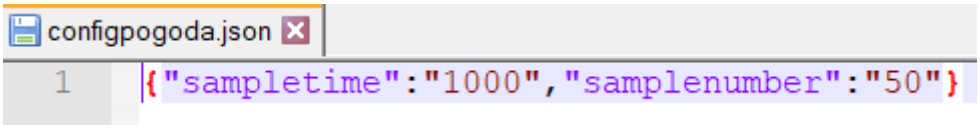


Rysunek 1 Widok główny

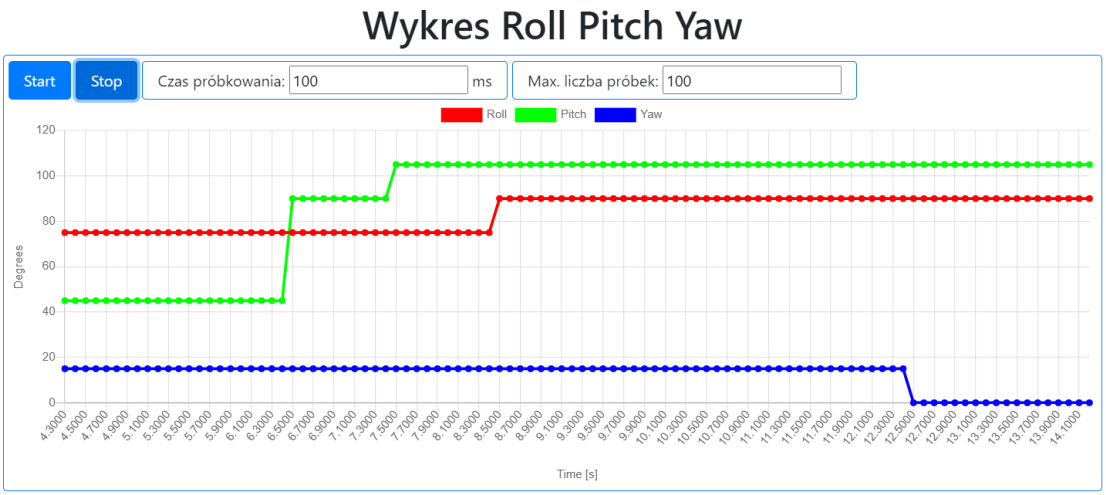
Rysunek 2 pokazuje działanie wykresów z danymi pogodowymi, a na Rysunku 4 możemy zobaczyć wykres danych o kątach roll, pitch oraz yaw. Na Rysunku 3 możemy zobaczyć, ze w pliku zostały zapisane dane o czasie próbkowania i maksymalnej liczbie wyświetlanych próbek.



Rysunek 2 Wykresy pogodowe



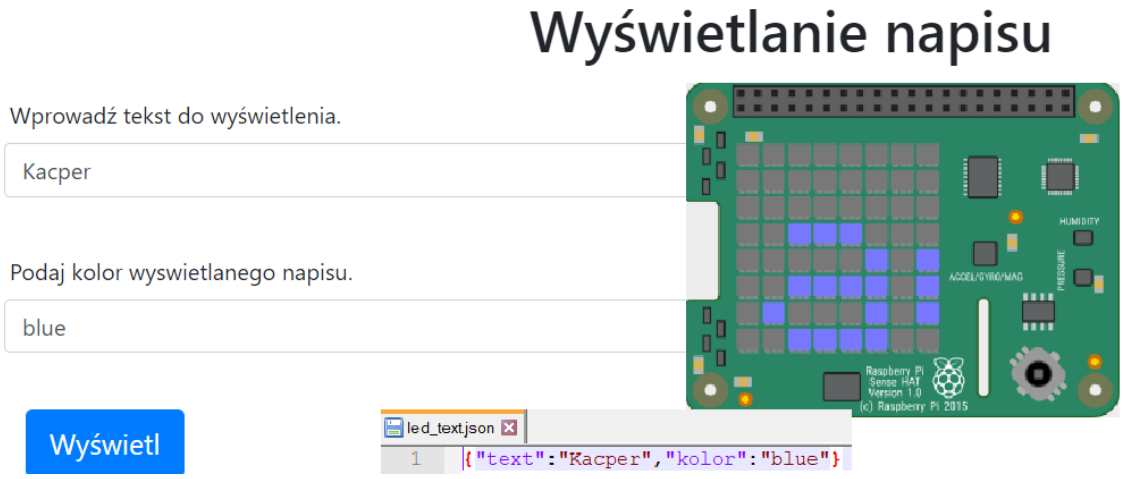
Rysunek 3 Zapisane dane



Rysunek 4 Wykres kątów roll, pitch i yaw

Na Rysunku 5 i 6 widzimy sterowanie kolejno pojedyńczą diodą na matrycy led oraz wyświetlanie napisów na niej.

Rysunek 5 Sterowanie pojedyńczą diodą led

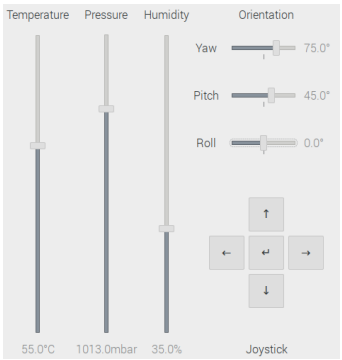


Rysunek 6 Sterowanie wyświetlaniem tekstu na matrycy led

Mamy także możliwość podglądu aktualnej wartości pogody, kątów oraz ilości operacji na joysticku z zachowaniem logiki, że wychylenia joysticka w gore zwiekszamy licznik Y axis a w dół zmniejszamy. Analogicznie działa też zliczanie X axis w lewo i prawo. Na Rysunku 7 możemy zobaczyć ten widok, a na Rysunku 8 ustawione wartości w emulatorze.

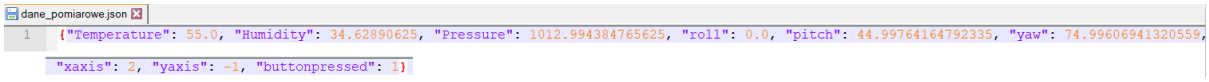
Dana	Wartość	Jednostka
Temperature	54.984375	°C
Pressure	1013.015869140625	hPa
Humidity	35.078125	%
Roll	0	°
Pitch	44.99764164792335	°
Yaw	74.99606941320559	°
X axis	2	[-]
Y axis	-1	[-]
Button pressed	1	[-]

Rysunek 7 Tabela dynamiczna z wszystkimi wielkościami



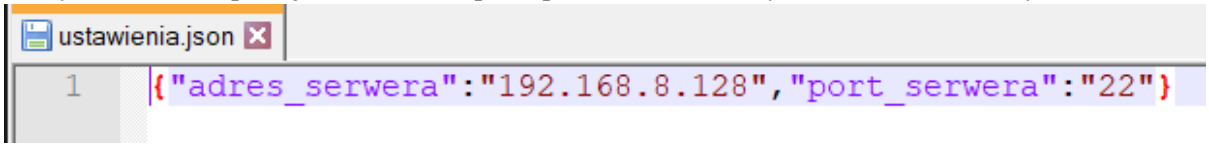
Rysunek 8 Nastawione wartości na emulatorze

Dodatkowo, wszystkie aktualne dane są zapisywane do pliku w formacie JSON, na Rysunku 9 widzimy zdjęcie pokazujące zawartość pliku.



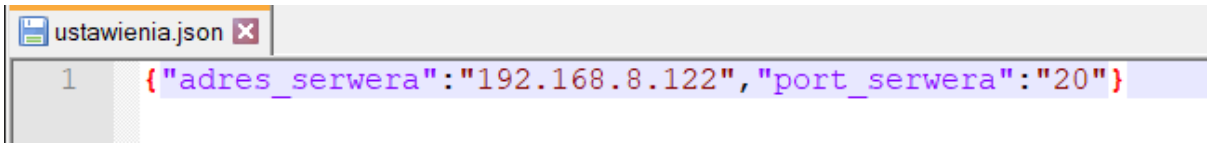
Rysunek 9 Plik z danymi pomiarowymi

Zmiana adresu IP oraz portu jest automatycznie zapisywana do pliku. Dane przed zmiana znajdują się na Rysunku 10, a operacja oraz dane w pliku po zmianie możemy zaobserwować na Rysunku 11 i 12.



Rysunek 10 Dane w pliku przed zmiana

Rysunek 11 Zmiana adresu IP oraz portu



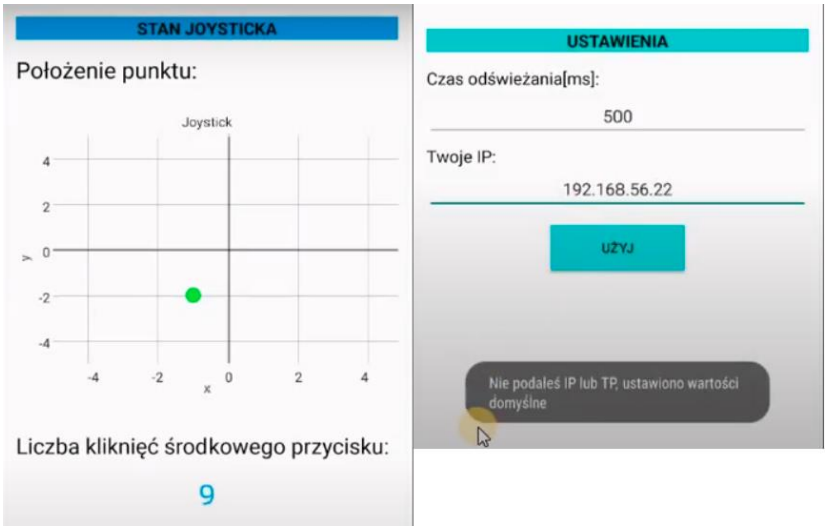
Rysunek 12 Dane w pliku po zmianie

3.2 Aplikacja Mobilna

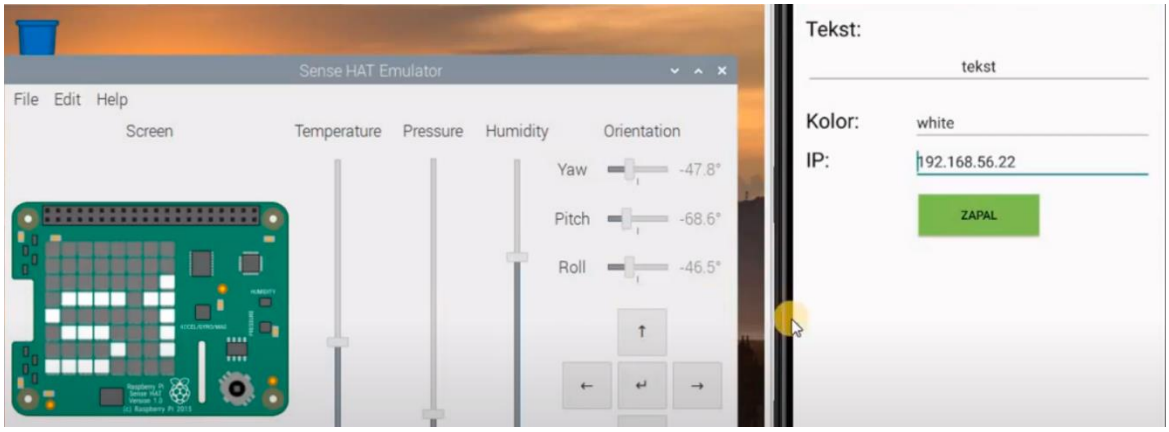
Na poniższych screenach pokazane zostało działanie aplikacji mobilnej w praktyce. Zrzuty ekranu pochodzą z filmiku prezentującego działanie aplikacji, w którym to dokładnie widać integrację aplikacji z emulatorem nakładki Sense Hat do Raspberry Pi.

Rysunek 13 Widok listy z danymi, możliwość zmiany jednostek oraz czasu próbkowania

Rysunek 14 Działanie wykresów, widok opcji, po prawej wykres położenia po zmianie jednostki



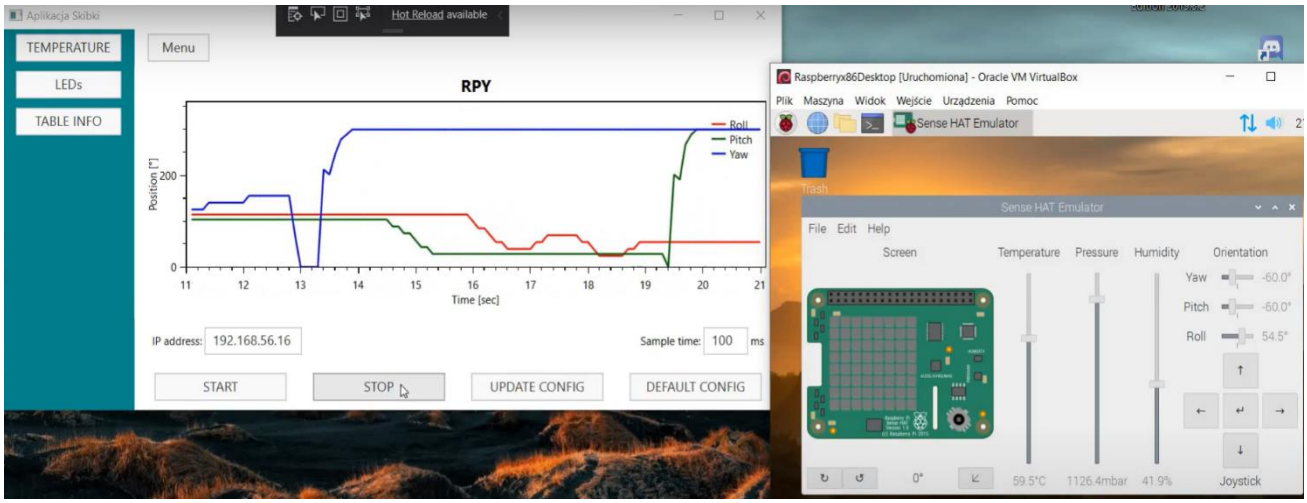
Rysunek 15 Widok stanu joysticka oraz przykładowy komunikat dla użytkownika



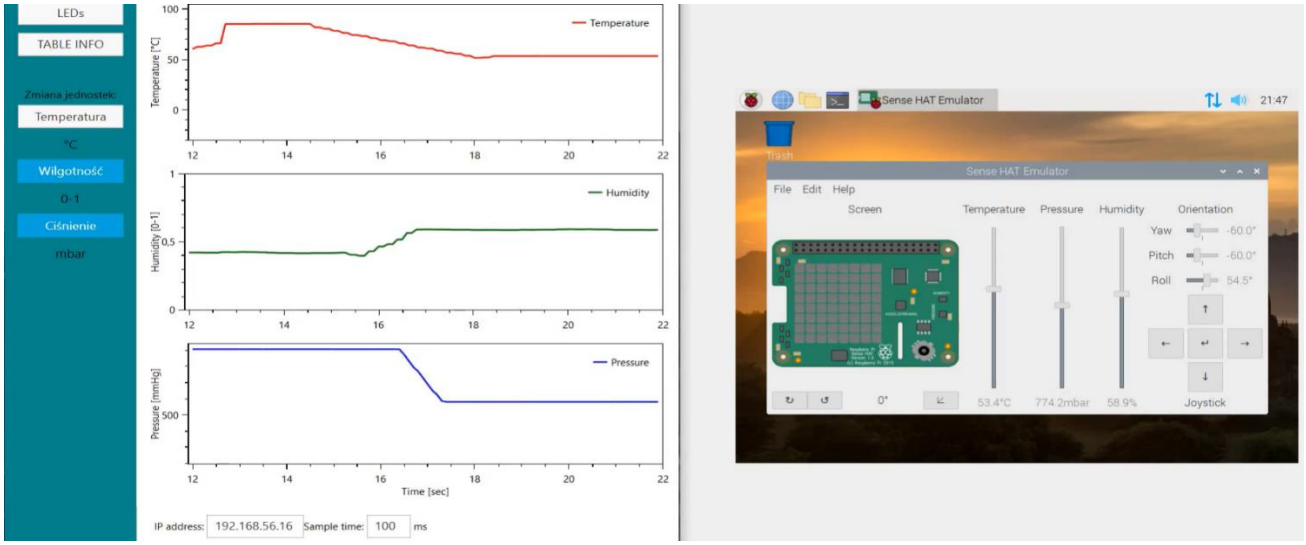
Rysunek 16 Wyświetlanie tekstu na matrycy LED

3.3 APLIKACJA DESKTOPOWA

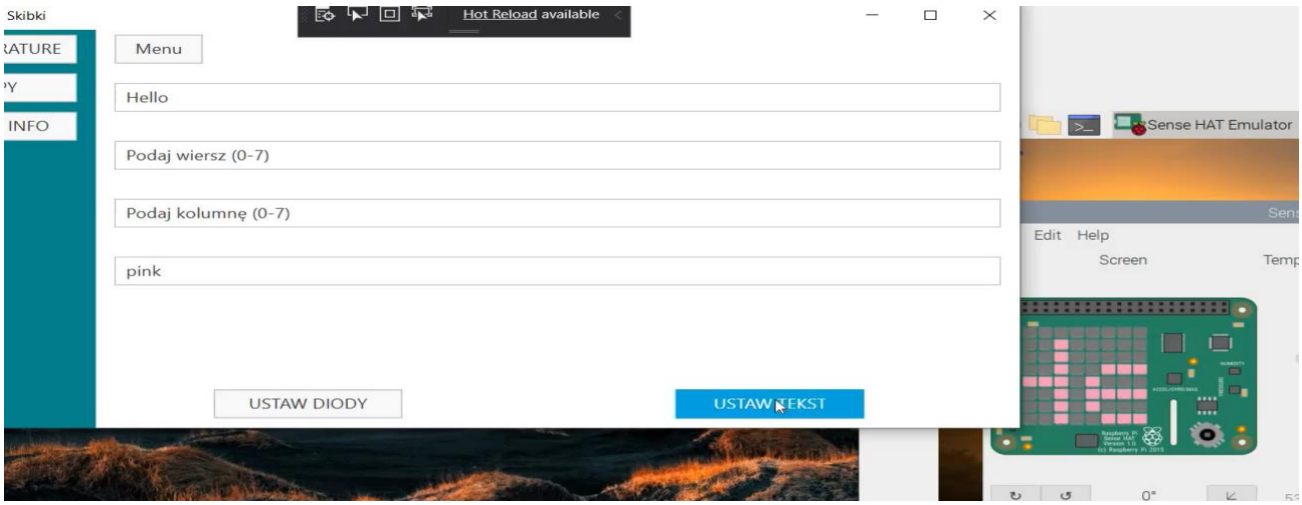
Na poniższych Rysunkach zostały zamieszczone przykłady wykorzystania aplikacji:



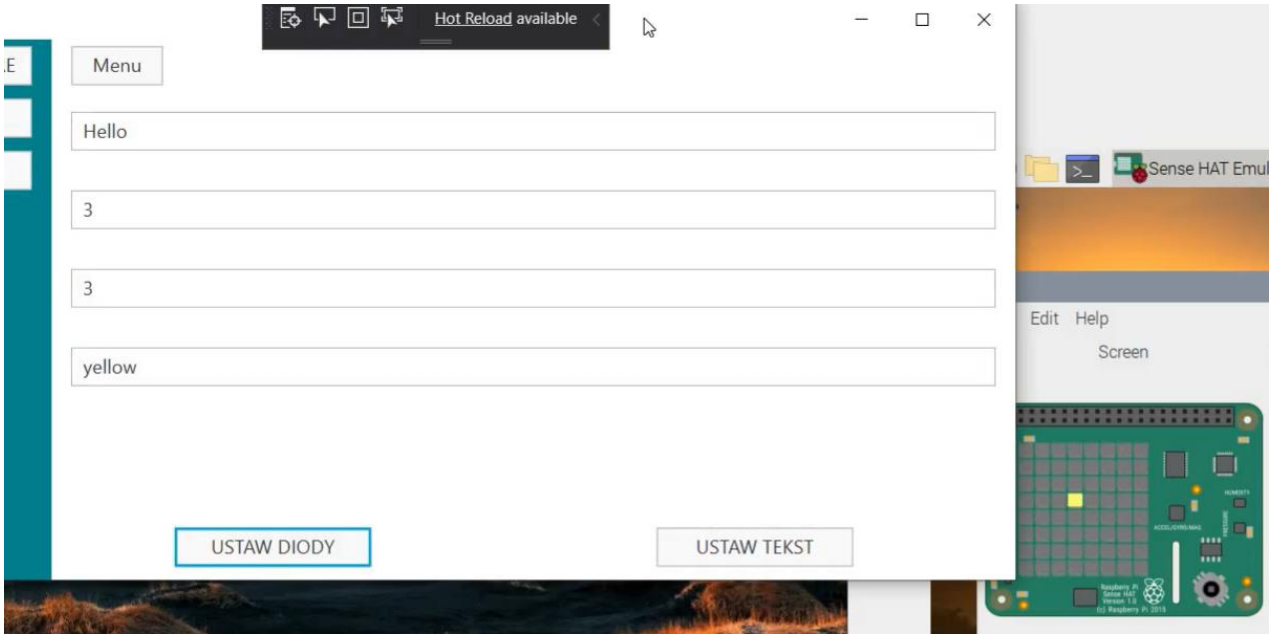
Rysunek 17 Wykres kątów RPY



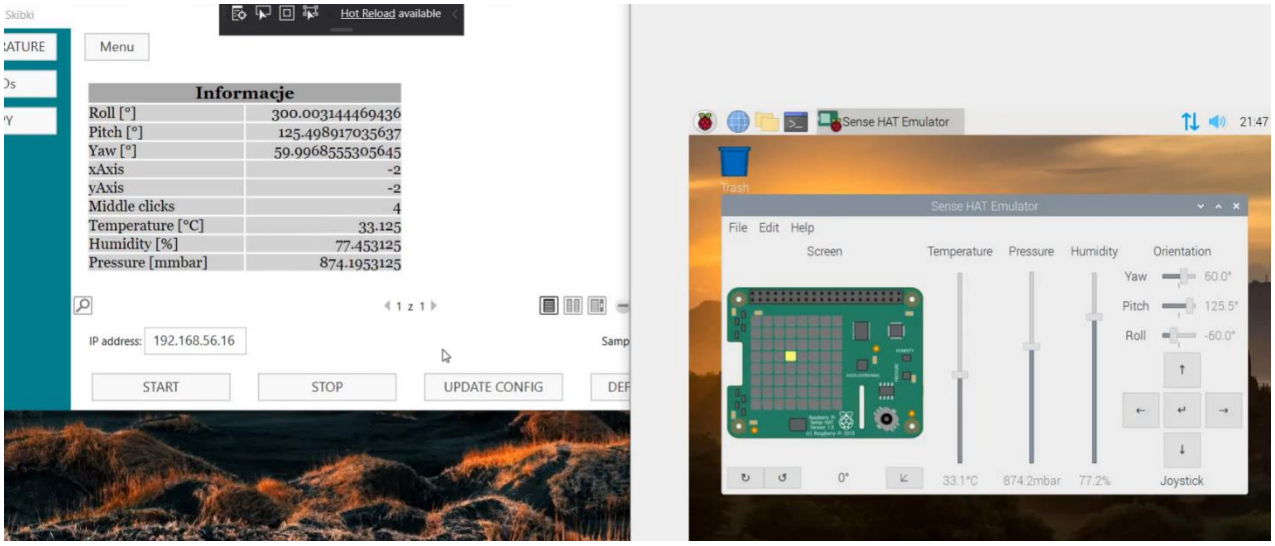
Rysunek 18 Wykres pogodowy



Rysunek 19 Ustawianie tekstu



Rysunek 20 Ustawienie diody



Rysunek 21 Dane pomiarowe w formie tabeli

4 WNIOSKI I PODSUMOWANIE

Zostały spełnione wszystkie wymogi wymienione w punkcie 1.

Mamy możliwość wyświetlenia danych pogodowych, orientacji w formie tabeli lub wykresów oraz joysticka, a także możliwość wysyłania danych sterujących do wyświetlania tekstu, bądź zapalania diody. Zatem wszystkie trzy aplikacje pobierają dane i wysyłają odpowiednie dane do sterowania płytką. Mamy także podgląd danych pomiarowych w formie tabeli, gdzie dane są odświeżane dynamicznie i wyświetlane na bieżąco co 0,5s. Wszystkie trzy aplikacje klienckie umożliwiają podgląd danych pomiarowych za pomocą wykresu przebiegu czasowego. Założenie w pełni spełnione, ponieważ mamy możliwość w każdej aplikacji klienckiej wyświetlania wykresów przebiegu czasowego zarówno dla danych pogodowych jak i orientacji. Wszystkie trze aplikacje mają próbkowanie danych z czasem mniejszym niż 1000ms. W każdej aplikacji w GUI zawarte są informacje o danej jednostce pomiarowej, która jest wyświetlana. Każda aplikacja gwarantuje nam

sterowanie pojedynczymi elementami wykonawczymi jakim jest pojedyncza dioda. Mamy możliwość zaświecania dowolnej diody na wybrany kolor, a także istnieje możliwość wyświetlania napisu. Każda aplikacja umożliwia także zmianę adresu ip oraz konfigurację wykresów. System został zaimplementowany zgodnie z architekturą REST, gdzie mamy żądania http request oraz odpowiedź http response. Do komentowania kodu dla wszystkich trzech aplikacji zastosowano standard doxygen. Aplikacje próbkują także dane z czasem mniejszym niż 100ms. Do aplikacji webowej oraz android wykorzystano kontrolę wersji git. Repozytoria zostały założone na stronie github.com. Poniżej podano linki:

https://github.com/JedrzejGrzebisz/Aveg

https://github.com/Djabollos/WebApp

Dla aplikacji desktopowej zastosowano system kontroli wersji SVN, gdzie do jego obsługi wykorzystano interfejs TortoiseSVN.

Dodatkowo dla aplikacji desktopowej wykorzystano wzorzec projektowy zapewniający separację interfejsu użytkownika od logiki aplikacji. Zastosowano Model-View-VieModel.