Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Urządzenia peryferyjne

GPS

wysyłanie współrzędnych geograficznych kanałem ble

Prowadzący:

dr inż. Jan Nikodem

Grupa:

Piątek 13:15 TP

Autorzy:

Daniel Glazer, 252743

Paweł Helisz, 252779

Wrocław 05.01.2022

Spis treści

1 Cel ćwiczenia **3**

2 GPS **3**

2.1 Segment kosmiczny . . . 3

2.2 Segment naziemny . . . 3

2.3 Segment użytkowników . . . 4

2.4 NMEA . . . 4

2.4.1 Parsowanie bramek GGA . . . 5

2.4.2 Parsowanie bramek RMC . . . 5

3 Aplikacja na telefon **6**

3.1 Obsługa aplikacji . . . 6

3.2 Kod i działanie . . . 7

3.2.1 MapView . . . 7

3.2.2 Pobieranie lokalizacji GPS . . . 8

3.2.3 Łączność BLE . . . 8

4 Ostateczne wyniki **11**

1 Cel ćwiczenia

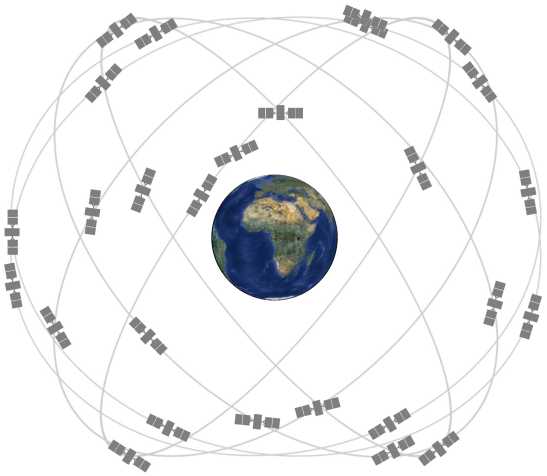
Celem ćwiczenia było napisanie aplikacji mobilnej, która miała umożliwić ustalenie pozycji urządzenia, a także wysyłanie oraz odbieranie lokalizacji GPS poprzez BLE. Ustalona pozycja miała zostać wyświetlona na mapie.

2 GPS

Global Positioning System (GPS), właściwie GPS-NAVSTAR, to satelitarny system radionawigacyjny będący własnością rządu USA i obsługiwany przez Siły Kosmiczne Stanów Zjednoczonych (United States Space Force). Jest to jeden z globalnych systemów nawigacji satelitarnej (GNSS - Global Navigation Satellite System), który dostarcza informacje o geolokalizacji i czasie do odbiornika GPS. System składa się z trzech segmentów:

2.1 Segment kosmiczny

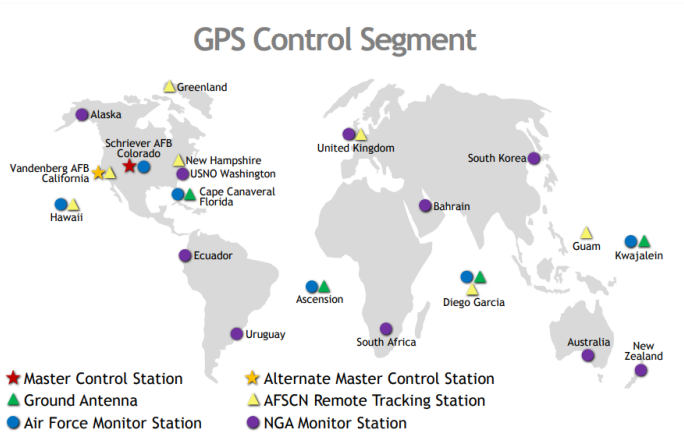
Składa się z 24 satelitów krążących na średniej orbicie okołoziemskiej (MEO) na wysokości około 20200 km. Satelity w konstelacji GPS są ułożone w sześć równo rozmieszczonych płaszczyzn orbitalnych otaczających Ziemię. Płaszczyzny orbitalne są odchylone względem równika o około 55°. Z każdego punktu globu widoczne są zawsze przynajmniej 4 satelity. Dzięki temu odbiornik użytkownika jest w stanie obliczyć trzy odległości do satelitów oraz odchyłki czasu (różnice między tanim i niedostatecznie dokładnym wzorcem kwarcowym zainstalowanym na odbiorniku i precyzyjnym zegarem atomowym na satelicie).



Rysunek 1: Konstelacja 24 satelitów

2.2 Segment naziemny

Segment kontroli GPS składa się z globalnej sieci obiektów naziemnych, które śledzą satelity GPS, monitorują ich transmisje, wykonują analizy oraz wysyłają polecenia i dane do konstelacji. Obecny segment kontroli operacyjnej (OCS) obejmuje główną stację kontroli, alternatywną główną stację kontroli, 11 anten dowodzenia i kontroli oraz 16 miejsc monitorowania. Lokalizację tych obiektów prezentuje poniższa mapa.



Rysunek 2: Mapa stacji kontroli GPS

2.3 Segment użytkowników

Segment użytkowników składa się z dziesiątek milionów cywilnych, komercyjnych i naukowych użytkowników standardowej usługi pozycjonowania. Ogólnie rzecz biorąc, odbiorniki GPS składają się z anteny dostrojonej do częstotliwości nadawanych przez satelity, procesorów odbiornika i bardzo stabilnego zegara (często oscylatora kwarcowego).

2.4 NMEA

National Marine Electronics Association (NMEA) - standard specyfikujący interfejs komunikacyjny i opis protokołów. Umożliwia on komunikację pomiędzy różnymi urządzeniami pomiarowymi oraz integrację modułu GPS z innymi urządzeniami. Ideą standardu NMEA jest przesyłanie linii danych, która zaczyna się od nagłówka i zawiera informacje wysyłane przez urządzenie.

1. dane wysyłane są w sposób tekstowy
2. nagłówek określa, jakie informacji znajdują się w danej linii danych
3. każda linia danych jest niezależna od innych
4. informacje separowane są przecinkiem

Istotne jest ustawienie interfejsu zgodnie z przyjętym standardem:

1. prędkość 4800 bodów
2. 8 bitów danych
3. brak kontroli parzystości
4. 1 bit stopu

Uaktualnienie pozycji co 2 sekundy jest skutkiem ograniczenia prędkości. Suma kontrola jest obliczana za pomocą operacji XOR na wszystkich znakach od początku linii (za wyjątkiem znaku $) do samej sumy kontrolnej (bez sumy kontrolnej i znaku *)

2.4.1 Parsowanie bramek GGA

Przykładowa linia danych z nagłówkiem GPGGA:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

1. GGA – Identyfikator nagłówka
2. 123519 – Aktualność danych- 12:35:19 UTC
3. 4807.038,N – szerokość geograficzna (latitude) - 48 deg 07.038' N
4. 01131.000,E – długość geograficzna (longitude) - 11 deg 31.000' E
5. 1 – jakość pomiaru
6. 08 – ilość śledzonych satelitów
7. 0.9 – horyzontalna dokładność pozycji (HDOP)
8. 545.4m – wysokość w metrach nad poziom morza
9. 46.9m – wysokość geoid (powyżej elipsoidy WGS84)
10. (puste pole) – czas od czasu ostatniego uaktualnienia DGPS
11. (puste pole) – numer ID stacji DGPS
12. *47 – suma kontrolna

2.4.2 Parsowanie bramek RMC

Przykładowa linia danych z nagłówkiem GPRMC:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

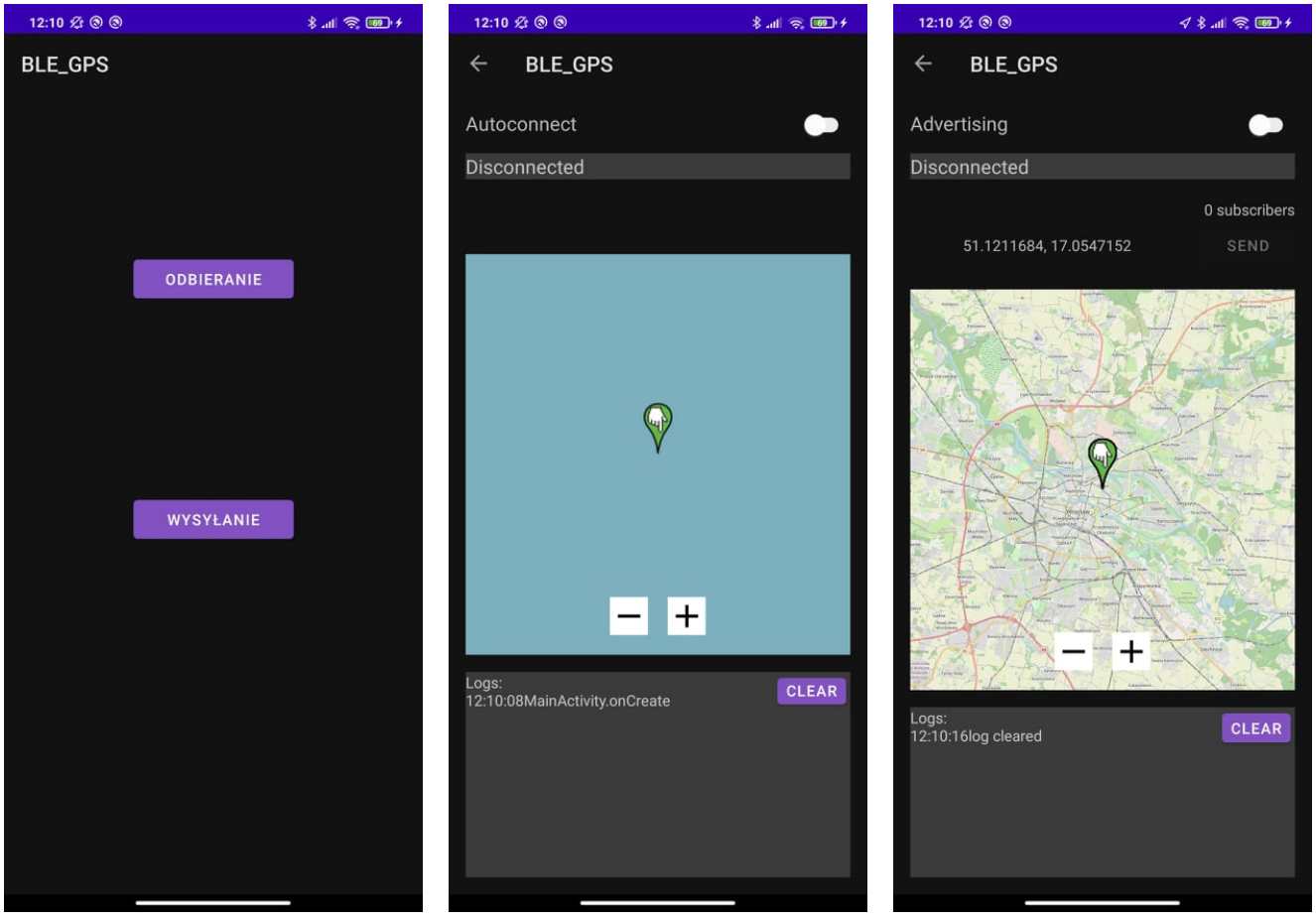
1. RMC – identyfikator nagłówka
2. 123519 – Aktualność danych - 12:35:19 UTC
3. A – status (A – aktywny; V – nieaktywny)
4. 4807.038,N – szerokość geograficzna (latitude) - 48 deg 07.038' N
5. 01131.000,E – długość geograficzna (longitude) - 11 deg 31.000' E
6. 022.4 – prędkość obiektu (liczona w węzłach)
7. 084.4 – kąt śledzenia/poruszania się obiektu (w stopniach) – przydatny w celu określenia kierunku poruszania się obiektu, jeśli urządzenie GPS nie jest wyposażone w kompas
8. 230394 – data (23 marca 1994)
9. 003.1,W – odchylenie magnetyczne ziemi
10. *6A – suma kontrolna

3 Aplikacja na telefon

Do utworzenia aplikacji wykorzystano aplikacje z poprzedniego ćwiczenia dotyczącego akwizycji sygnałów i dostosowano ją pod obsługę GPS.

3.1 Obsługa aplikacji

Aplikacja składa się z trzech okien, okna głównego 3a, okna odbierania danych 3b (centrali) oraz okna wysyłania lokalizacji 3c. Przy pierwszy uruchomieniu jednego z dwóch trybów działania aplikacji, program zapyta o potrzebne uprawnienia. W przypadku gdy uruchomimy aktywność obsługującą wysyłanie lokalizacji i będzie ona wskazywać współrzędne *1, 1* należy w ustawieniach telefonu uruchomić moduł GPS. Aby przesłać dane z jednego smartfona do drugiego należy uruchomić na jednym telefonie aplikacje w trybie odbierania i za pomocą przycisku *Autoconnect* uruchomić skaner, natomiast na drugim telefonie należy włączyć program w trybie wysyłania i zmieniając stan przycisku *Advertising* uruchomić rozgłaszanie. W sytuacji, w której urządzenia się połączą na oby dwóch będzie widniał napis *Connected*, a przycisk *Send* zostanie odblokowany. Po naciśnięciu przycisku *Send* współrzędne lokalizacji zostaną wysłane do drugiego telefonu.



(a) Menu główne (b) Odbieranie lokalizacji (c) Wysyłanie lokalizacji

Rysunek 3: Poszczególne okna aplikacji

3.2 Kod i działanie

3.2.1 MapView

Aplikacja miała wyświetlić lokalizacje na mapie. W tym celu została wykorzystana biblioteka *osmdroid* (`https://github.com/osmdroid/osmdroid`), której wykorzystywanie przeciwieństwie do *Map Google* jest w pełni darmowe. Biblioteka ta umożliwia nam utworzenie komponentu mapy. Po mapie można się poruszać, ustawiać na niej pinezki lub zmieniać jej przybliżenie.

Aby wykorzystać bibliotekę *osmdroid*, należało dodać zależność do pliku *build.gradle* w zakładce *dependencies*:

```
implementation 'org.osmdroid:osmdroid-android:6.1.8'
```

Następnym krokiem było dodanie do plików *activity_central* oraz *activity_peripheral* komponentu mapy:

```
<org.osmdroid.views.MapView
    android:id="@+id/map"
    android:layout_width="0dp"
    android:layout_height="0dp" />
```

Utworzenie komponentu mapy wraz z jej ustawieniami domyślnymi:

```
Map(MapView map, Context ctx) {
        this.map = map;
        map.setTileSource(TileSourceFactory.MAPNIK);
        map.setMultiTouchControls(true);

        Configuration.getInstance().load(ctx,
            PreferenceManager.getDefaultSharedPreferences(ctx));
        mapController = map.getController();
        mapController.setZoom(12.0);

        startMarker = new Marker(map);
    }
```

Ustawienie pinezki na lokalizacji zawartej w *coordinates*:

```
public void setMapController(String coordinates) {
        String[] results = coordinates.split(", ");

        double aLatitude = Double.parseDouble(results[0]);
        double aLongitude = Double.parseDouble(results[1]);
        GeoPoint geoPoint = new GeoPoint(aLatitude, aLongitude);

        startMarker.setPosition(geoPoint);
        startMarker.setAnchor(Marker.ANCHOR_CENTER, Marker.ANCHOR_BOTTOM);
        map.getOverlays().add(startMarker);

        mapController.animateTo(geoPoint);
    }
```

3.2.2 Pobieranie lokalizacji GPS

Aplikacja miała za zadanie pobierać współrzędne geograficzne urządzenia, w tym celu musi uzyskać uprawnienia do lokalizacji w telefonie oraz musi zostać zdefiniowana dokładność oraz czas aktualizacji lokalizacji urządzenia:

```
fusedLocationClient = LocationServices.getFusedLocationProviderClient(peripheralActivity);
if (ActivityCompat.checkSelfPermission(peripheralActivity,
        Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission(peripheralActivity,
        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
        return;
fusedLocationClient
    .getLastLocation()
    .addOnSuccessListener(peripheralActivity, this::setLocation);

locationRequest = LocationRequest.create();
locationRequest.setInterval(1000);
locationRequest.setFastestInterval(500);
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

locationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(@NonNull LocationResult locationResult) {
        setLocation(locationResult.getLastLocation());
    }};
```

Dodano również akcje w przypadku zatrzymania lub wznowienia działania aplikacji:

```
void onResume(Context context) {
    if (ActivityCompat.checkSelfPermission(context,
    Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
    && ActivityCompat.checkSelfPermission(context,
    Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
        return;

    fusedLocationClient.requestLocationUpdates(locationRequest,
            locationCallback, Looper.getMainLooper());
}
void onPause() { fusedLocationClient.removeLocationUpdates(locationCallback); }
```

3.2.3 Łączność BLE

Aplikacja miała za zadanie połączyć się przez BLE z innym urządzeniem i przesłać lub odebrać lokalizację. W tym celu zdefiniowano UUID, które jest takie samo dla urządzenia odbierającego oraz wysyłającego lokalizacje:

```
private final String SERVICE_UUID = "25AE1441-05D3-4C5B-8281-93D4E07420CF";
private final String CHAR_FOR_INDICATE_UUID = "25AE1444-05D3-4C5B-8281-93D4E07420CF";
private final String CCC_DESCRIPTOR_UUID = "00002902-0000-1000-8000-00805f9b34fb";
```

3.2.3.1 Odbieranie danych

Aby odebrać dane należy przełączyć przycisk i uruchomić skaner wykrywający urządzenie:

```
    switchConnect.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                if (isChecked) {
                    IntentFilter filter = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
                    if(start())
                        registerReceiver(bleOnOffListener, filter);
                } else
                    unregisterReceiver(bleOnOffListener);
                bleRestartLifecycle(); }});
```

Następnie należy skonfigurować skaner bluetooth:

```
    BluetoothManager bluetoothManager = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
    BluetoothAdapter bluetoothAdapter = bluetoothManager.getAdapter();
    if (enableBluetooth(bluetoothAdapter)) {
        bleScanner = bluetoothAdapter.getBluetoothLeScanner();
        scanSettings = scanSettingsSinceM;
        return  true; }
```

Po udanym skonfigurowaniu skanera należy uruchomić poszukiwanie urządzenia BLE o UUID zdefiniowanym na samym początku programu:

```
    String serviceFilter = scanFilter.getServiceUuid().getUuid().toString();
    isScanning = true;
    lifecycleState = BLELifecycleState.Scanning;
    textViewLifecycleState.setText("Scanning");

    List<ScanFilter> scanFilterList = new ArrayList<>();
    scanFilterList.add(scanFilter);
    bleScanner.startScan(scanFilterList, scanSettings, scanCallback);
```

Po znalezieniu drugiego urządzenia nawiązywane jest połączenie:

```
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
                String deviceAddress = gatt.getDevice().getAddress();
                if (status == BluetoothGatt.GATT_SUCCESS) {
                    if (newState == BluetoothProfile.STATE_CONNECTED) {
                        appendLog("Connected to" + deviceAddress);

                        new Handler(Looper.getMainLooper()).post(() -> {
                            lifecycleState = BLELifecycleState.ConnectedDiscovering;
                            textViewLifecycleState.setText("ConnectedDiscovering");
                            gatt.discoverServices();
                        }); } }
    ... }
```

W momencie otrzymania danych konwertowane są one z ciągu bitów na ciąg znakowy, które jest rozdzielany na współrzędne:

```
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic){
    if (characteristic.getUuid().equals(UUID.fromString(CHAR_FOR_INDICATE_UUID))) {
        byte [] a = characteristic.getValue();
        String strValue = new String(a, Charset.defaultCharset());
        appendLog("onCharacteristicChanged value= " + strValue);
        runOnUiThread(() ->{
            textViewIndicateValue.setText(strValue);
            mapCentral.setCoordinates(strValue);
        });
        } else
            appendLog("onCharacteristicChanged unknown uuid: " + characteristic.getUuid());
}
```

3.2.3.2 Wysyłanie danych

Podobnie jak w przypadku odbierania danych, przycisk uruchamia BLE, ale w tym wypadku w trybie rozgłaszania:

```
switchAdvertising.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if (isChecked) {
                if (start())
                    prepareAndStartAdvertising();
            } else
                bleStopAdvertising();
        });

private boolean start() {
    bluetoothManager = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
    BluetoothAdapter bluetoothAdapter = bluetoothManager.getAdapter();
    if (enableBluetooth(bluetoothAdapter)) {
        bleAdvertiser = bluetoothAdapter.getBluetoothLeAdvertiser();
        return true; }
    return false;
}
```

W momencie połączenia z drugim urządzeniem zostaje odblokowany przycisk *Send* odpowiadający za wysyłanie lokalizacji:

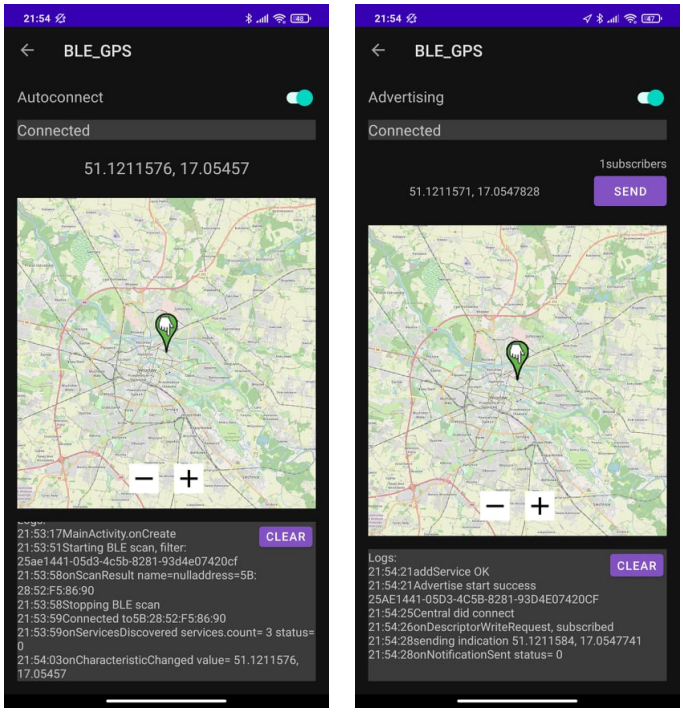
```
public void onConnectionStateChange(BluetoothDevice device, int status, int newState) {
            runOnUiThread(() -> {
                if (newState == BluetoothProfile.STATE_CONNECTED) {
                    textViewConnectionState.setText("Connected");
                    appendLog("Central did connect");
                    buttonSend.setEnabled(true);
                }
            });
        }
```

Naciśniecie przycisku *Send* powoduje wysłanie lokalizacji zapisanej w postaci ciągu znakowego:

```
1     public void onTapSend(View view) { bleIndicate(); }
2
3     private void bleIndicate() {
4         charForIndicate = gattServer
5                         .getService(UUID.fromString(SERVICE_UUID))
6                         .getCharacteristic(UUID.fromString(CHAR_FOR_INDICATE_UUID));
7
8         String text = editTextCharForIndicate.getText().toString();
9         charForIndicate.setValue(text);
10        for (BluetoothDevice device : subscribedDevices) {
11            appendLog("sending indication " + text);
12            gattServer.notifyCharacteristicChanged(device, charForIndicate, true);
13        }
14    }
```

4 Ostateczne wyniki

Na poniższych rysunkach zaprezentowano rezultaty działania aplikacji po połączeniu z drugim telefonem. Zrzuty ekranu były robione tylko na jednym telefonie, który na rysunku 4a był w trybie odbierania danych, a na rysunku 4b w trybie wysyłania danych.



(a) Okno odbioru lokalizacji po otrzymaniu danych

(b) Okno wysłania lokalizacji po połączeniu z centralą

Rysunek 4: Okna aplikacji po połączeniu z drugim telefonem i przesłaniu lokalizacji