POLITECHNIKA WARSZAWSKA

Wydział Mechatroniki

Praca przejściowa

Jakub Mikołaj Szlendak

System wyznaczania odległości na podstawie siły sygnału radiowego

Opiekun pracy:
mgr inż. Łukasz Chechliński

Konsultant pracy:
mgr inż. Daniel Koguciuk

Warszawa, 2017

Spis treści

Spis rysunków

Rozdział 1

Wstęp

Problem lokalizacji robota w pomieszczeniu zamkniętym jest w ostatnich latach często rozważany. Jako że nawigacja satelitarna (GPS, GLONASS) jest niedostępna w pomieszczeniach zamkiętych, konieczne jest opracowanie innych metod lokalizowania robota. Do tych metod należą m. in:

- lokalizacja w oparciu o wizualne znaczniki i system ich rozpoznawania
- lokalizacja na podstawie stereowizji
- odometria
- lokalizacja na podstawie odległości od znaczników (radiowych, akustycznych itp)

Przedmiotem niniejszej pracy jest zaprojektowanie oprogramowania do znacznika radiowego i odbiornika, pozwalającego na wyznaczanie odległości odbiornika do znacznika na podstawie parametru RSSI (Received Signal Strength Indication). Parametr RSSI określa moc odbieranego sygnału radiowego.

Takie znaczniki mogą zostać rozmieszczone w środowisku pracy robota, z kolei robot może zostać niskim kosztem wyposażony w odbiornik radiowy [1]. Dysponując mapą rozmieszczenia znaczników w pomieszczeniu oraz informacją o odleglościach pomiędzy robotem a poszczególnymi znacznikami, można wyznaczać pozycję robota.

Ze względu na niski koszt sprzętu i łatwość implementacji, do implementacji rozwiązania wybrano protokół Bluetooth Low Energy.

Rozdział 2

Protokół Bluetooth

Bluetooth jest standardem komunikacji bezprzewodowej, zaprojektowanym do wymiany danych na krótkie dystanse, korzystający z fal krótkich UHF pasma ISM (częstotliwości rzędu 2.4 GHz). Standard Bluetooth jest zarządzany przez organizację Bluetooth Special Interest Group (SIG), będącą zrzeszeniem firm z branż komunikacyjnej, sieciowej, informatycznej i elektronicznej [7].

Pasmo ISM (ang. *Industrial, Scientific and Medical*, Przemysłowe, Naukowe i Medyczne) wykorzystywane przez Bluetooth jest nielicencjonowane. Dla zapewnienia odporności na zakłócenia, Bluetooth wykorzystuje technologię rozpraszania widma FHSS (ang. *Frequency Hopping Spread Spectrum*) polegającą na "skakaniu" po częstotliwościach dostępnych w paśmie w kolejnych odstępach czasu. Protokół Bluetooth jest oparty o pakiety, ze strukturą master-slave [7].

2.1 Bluetooth Low Energy

Bluetooth Low Energy, znane także pod nazwą handlową Bluetooth Smart, jest czwartą wersją specyfikacji standardu Bluetooth. Standard ten został zorientowany na zapewnienie transmisji o minimalnym koszcie energetycznym. Podstawowe cechy BLE to:

- niskie zużycie energii, pozwalające urządzeniom operować przez długi (rzędu roku) czas na jednej baterii guzikowej
- mały rozmiar i koszt urządzeń
- kompatybilność z popularnymi urządzeniami obecnymi na rynku (smartfony, tablety, laptopy)

Dla osiągnięcia efektywności energetycznej BLE wykorzystuje odmienny w stosunku do klasycznego Bluetooth model komunikacji. Występują dwa tryby pracy:

- tryb rozgłoszeniowy (ang. *advertising mode*)
- tryb połączenia (ang. *connection mode*)

W trybie rozgłoszeniowym urządzenie wysyła pakiety nieskierowane do żadnego odbiornika w stałym interwale czasowym. Pomiędzy nadawaniem procesor oraz radio urządzenia mogą pozostawać w stanie uśpienia, co pozwala na znaczną oszczędność energii. Pojedynczy pakiet rozgłoszeniowy zawiera 31 bajtów użytecznych danych i może być odebrany przez dowolne urządzenie w trybie nasłuchu (ang. *scan mode*). Nasłuchujące odbiorniki, po przechwyceniu pakietu rozgłoszeniowego i tym samym odkryciu urządzenia nadawczego, mogą nawiązać z nim połączenie. Tryb połączenia, podobnie jak w klasycznym Bluetooth, jest nawiązywany pomiędzy dwoma urządzeniami i służy do wymiany danych pomiędzy nimi.

Każde urządzenie BLE, które obsługuje bezpośrednie połączenia, posiada tzw. profil GATT (ang. *Generic ATTribute*) [7] . Profil ten opisuje, w jaki sposób odbywa się komunikacja pomiędzy urządzeniami BLE będącymi w połączeniu. Na profil składa się zbiór atrybutów, do których należą serwisy i charakterystyki. Charakterystyka reprezentuje pojedynczą logiczną wartość. Wartość ta może mieć od 1 do 20 bajtów i jej interpretacja (np. typ reprezentowanej danej) oraz sposób dostępu są opisywane przez tzw. deskryptor. Serwisy są logicznymi zbiorami charakterystyk. Wartość charakterystyki może być odczytywana, zapisywana, odczytywana z powiadomieniem itp. - zależnie od ustawienia deskryptora. Specyfikacja BLE przewiduje kilka gotowych schematów serwisów i charakterystyk, m.in. do obsługi urządzeń z interfejsem BLE takich jak glukometry, termometry lub czujniki pracy serca. Możliwe jest także budowanie własnych schematów serwisów i charakterystyk [7].

Rozdział 3

Znacznik radiowy

3.1 Wstęp

Znacznik radiowy do wykorzystania w systemie lokalizacji robota winien spełniać szereg wymagań:

Mały rozmiar - urządzenie powinno być zwarte i małych rozmiarów, aby możliwe było łatwe umieszczenie go w miejscu pracy robota

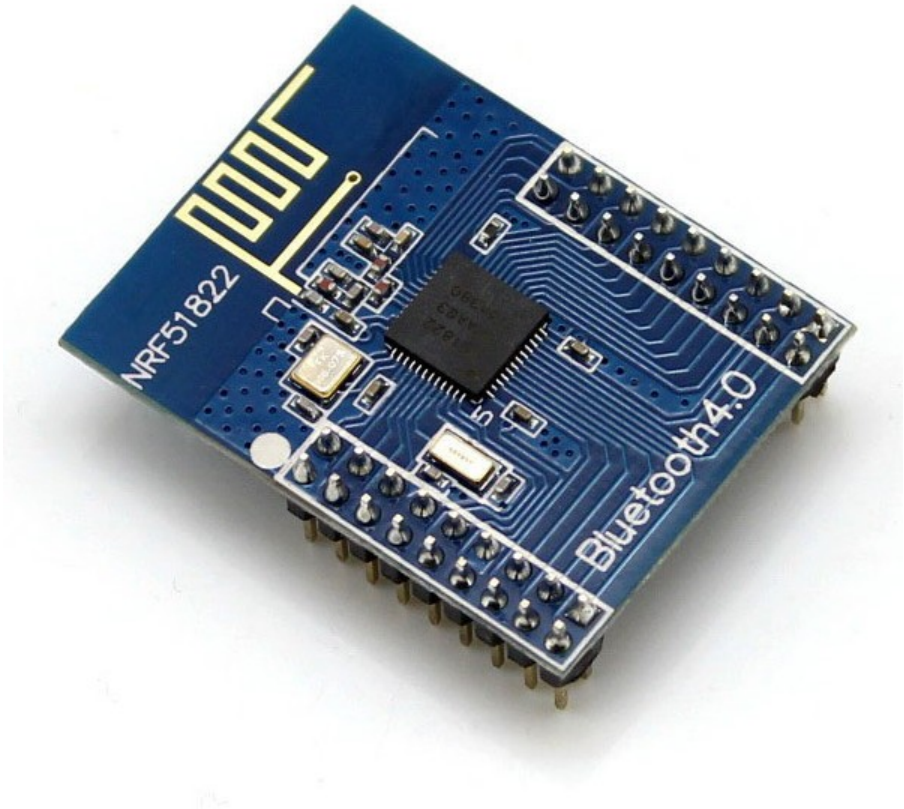
Zasilanie bateryjne - aby zapewnić swobodę rozmieszczenia, urządzenie powinno być zasilane. Ponadto, aby zapewnić bezobsługowość systemu, czas pracy na baterii powinien wynosić co najmniej 1 rok

Niski koszt jednostkowy - system lokalizacji wymaga co najmniej 3 znaczników aby był użyteczny, stąd istotny jest koszt jednostkowy znacznika

Mając na uwadze powyższe wymagania, zdecydowano zastosować mikrokontroler nRF51 firmy Nordic Semiconductor.

3.2 Procesor nRF51

Mikrokontroler Nordic Semiconductor nRF51 jest oparty o 32-bitowy rdzeń ARM Cortex M0+. W ramach pojedynczego układu scalonego, oprócz procesora, zintegrowano radio 2,4 GHz oraz szereg peryferiów, takich jak liczniki, zegary, interfejsy komunikacyjne czy sprzętowy układ szyfrujący wg algorytmu AES128. Procesor jest taktowany z częstotliwością 16 MHz, ponadto dysponuje oscylatorem o częstotliwości 32 kHz do precyzyjnego mierzenia czasu. Wykorzystany w projekcie wariant mikrokontrolera posiada 32 kB pamięci RAM oraz 256 kB pamięci flash, wykorzystywanej do przetrzymywania programu oraz do dyspozycji aplikacji.

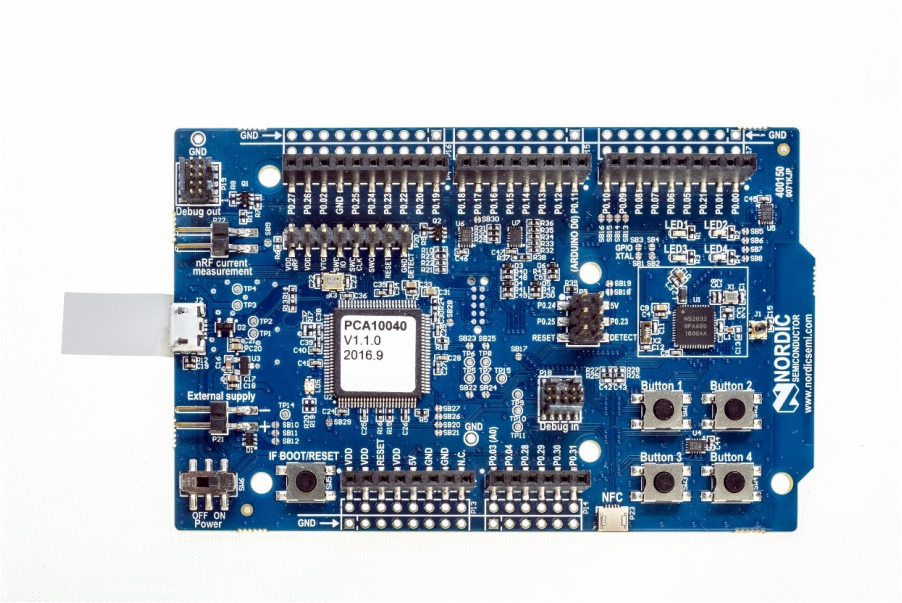


Rysunek 3.1: Moduł BLE z procesorem nRF51

Podstawową zaletą mikrokontrolera nRF51 jest bardzo niski pobór prądu. W trakcie pracy radia pobiera on ok. 8 mA, natomiast w stanie uśpienia zużycie prądu spada do 2,6 μA. Ponieważ czas pracy radia jest o rząd wielkości krótszy od czasu, który procesor spędza w uśpieniu, średnie zużycie prądu wynosi ok 5μA i waha się w zależności od skonfigurowanego interwału rozgłaszania [6].

Jako że niniejsza praca skupia się na zaprojektowaniu oprogramowania, a nie platformy sprzętowej, do projektu wykorzystano gotowy moduł z procesorem oferowany przez sklep Botland (rys. 3.1). Posiada on, oprócz procesora nRF51822, wszystkie elementy do rozwoju aplikacji Bluetooth, tj. 2 oscylatory kwarcowe, antenę, układ BALUN (BALancer-UNbalancer, konwerter sygnału dla anteny) oraz wyprowadzenie wszystkich nóżek procesora na złącza typu goldpin. Moduł posiada interfejs SWO do programowania i debugowania. Po dołączeniu zasilania (np. koszyczka na baterię CR2032) moduł działa jako samodzielny znacznik (beacon) BLE. Koszt takiego modułu wynosi ok. 30 zł.

Do programowania modułu wykorzystano programator J-Link znajdujący się na płytce rozwojowej nRF52 Developement Kit (rys. 3.2)



Rysunek 3.2: Płytka rozwojowa nRF52 z programatorem J-Link

3.3 Środowisko programistyczne

Procesory z rdzeniem ARM Cortex mogą być programowane w języku Assembler oraz C. Do projektu wybrano język C. Konieczne jest także wykorzystanie biblioteki programistycznej (SDK) firmy Nordic Semiconductor, w celu obsłużenia układów peryferyjnych mikrokontrolore oraz stosu Bluetooth. Sam stos Bluetooth jest dostarczany przez producenta w formie binarnej, jako obraz hex, który należy zapisać do pamięci procesora obok obrazu programu głównego [6]. Zgodnie z modelem przyjętym przez producenta, cała logika komunikacji BLE jest enkapsulowana w programie stosu i niedostępna dla programisty (jest to oprogramowanie zamknięte). Do zadań programisty należy skonfigurowanie parametrów takich jak nazwa rozgłaszana urządzenia, zawartość pakietu rozgłoszeniowego, interwał rozgłaszania. Programista jest także odpowiedzialny za napisanie funkcji obsługujących poszczególne charakterystyki.

Rozwój oprogramowania przeprowadzono w środowisku Eclipse w wersji Neon.2, wykorzystując kompilator GNU ARM GCC w wersji 5.4 i sterownik debuggera J-Link w wersji 5.12c.

3.4 Implementacja

Sformułowano następujące wymagania co do funkcjonalności znacznika radiowego:

- praca w trybie rozgłoszeniowym z interwałem 100 ms
- zawartość pakietu rozgłoszeniowego pozwalająca odróżnić beacon systemu od ewentual-

nych innych urządzeń BLE w okolicy

- zawartość pakietu rozgłoszeniowego pozwalająca rozróżnić poszczególne beacony wchodzące w skład systemu
- możliwość konfiguracji zawartości pakietu rozgłoszeniowego za pomocą aplikacji mobilnej

Należy zaznaczyć, że podstawowa funkcjonalność znacznika tj. przekazywanie siły sygnału RSSI jest dostępna dla każdej konfiguracji rozgłaszania, ponieważ jest elementem wbudowanym w stos Bluetooth.

3.4.1 Pakiet rozgłoszeniowy

Tabela 3.1 opisuje strukturę pakietu rozgłoszeniowego beacona. Specyfikacja protokołu BLE przewiduje, że w ramce rozgłoszeniowej znajdą się pola reprezentujące poszczególne informacje [7]. Pole składa się z 2 + N bajtów: pierwszy bajt zawiera długość pola, drugi określa typ informacji, pozostałe zawierają daną informację. Łączny rozmiar wszystkich pól nie może przekroczyć 31 bajtów. Należy także nadmienić, że do długości pola wlicza się bajt typu. Specyfikacja przewiduje typy takie jak nazwa, klasa urządzenia, lista dostępnych serwistów itp.

W ramce znacznika wykorzystano następujące typy (por. tab. 3.1):

Flagi - pole to jest obowiązkowe w każdej ramce. Zawiera jednobajtowe pole bitowe z flagami określającymi m.in. widoczność urządzenia.

Nazwa - pole to zawiera nazwę urządzenia, wyświetlaną przez urządzenia nasłuchujące rozgłaszania (np. smartfon). Nazwa urządzenia to „locationTAG”

Zawartość producenta - pole to jest przeznaczone do dowolnego wykorzystania przez producenta urządzenia. Zawarto w nim główną informacyjną treść ramki.

Pole “Zawartość producenta” zawiera następujące informacje (por. tab. 3.1):

Kod producenta - obowiązkowy kod producenta. Dla producentów niezarejestrowanych w Bluetooth SIG jego wartość wynosi FFFFh

Identyfikator grupy - 4-bajtowa liczba całkowita (unsigned int32), służąca do odróżniania grup beaconów (np. zebranych w jednym pomieszczeniu)

Identyfikator znacznika - 6-bajtowy identyfikator indywidualny dla beacona. Jest on równy adresowi fizycznemu MAC beacona.

Tabela 3.1: Struktura pakietu rozgłoszeniowego

Ramka rozgłoszeniowa [31B]											
Flagi [3B]			Nazwa [13B]			Zawartość [15B]					
Długość [1B]	*Typ* [1B]	Flagi [1B]	*Długość* [1B]	*Typ* [1B]	Nazwa [11B]	*Długość* [1B]	*Typ* [1B]	Kod producenta [2B]	Identyfikator grupy [4B]	Identyfikator znacznika [6B]	Wyrównanie [1B]

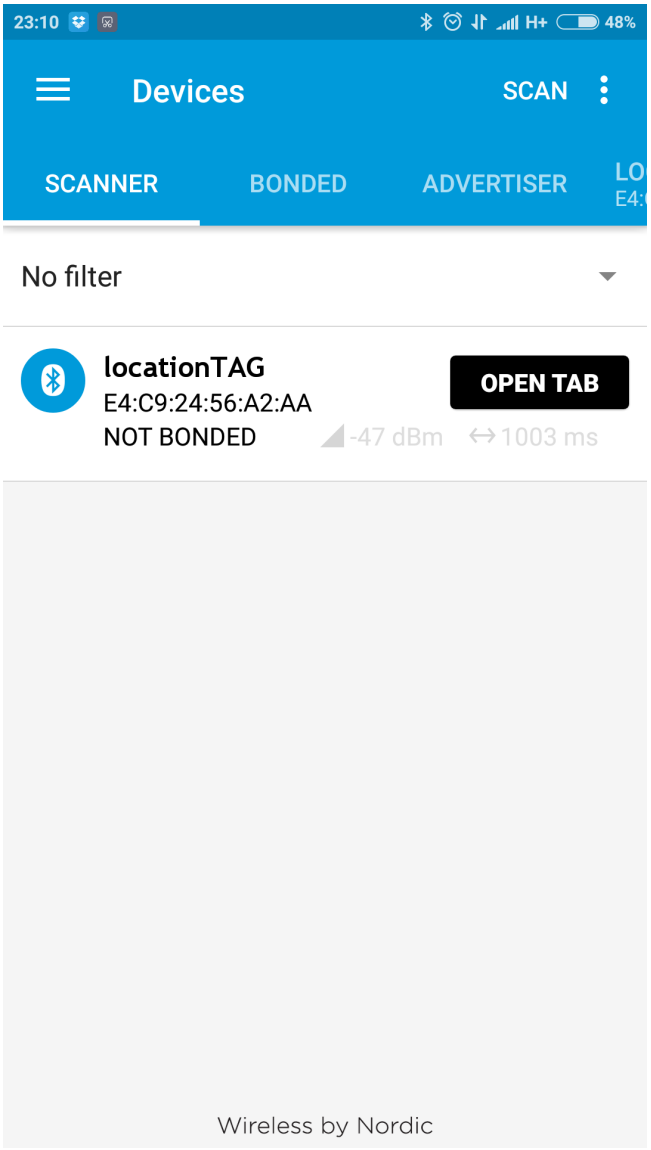
3.4.2 Charakterystyka konfiguracyjna

Aby zapewnić lepszą elastyczność systemu, w ramce rozgłoszeniowej znalazł się identyfikator grupy beaconów. Może być on wykorzystany np. do rozróżnienia beaconów rozmieszczonych w różnych pomieszczeniach albo do rozróżnienia znaczników dla kilku robotów. Identyfikator grupy może być konfigurowany za pomocą smartfona. W tym celu zaimplementowano serwis z charakterystyką konfiguracyjną. Jest to charakterystyka typu „write", co oznacza, że urządzenie połączone może zapisywać do niej dane. Charakterystyka przyjmuje 4 bajty danych, które zostają przypisane jako nowy identyfikator grupy. Ponieważ identyfikator grupy jest zapisywany w pamięci Flash mikrokontrolera, zmiana zostaje zachowana po zaniku zasilania urządzenia.

Do obsługi charakterystyki konfiguracyjnej wykorzystać można aplikację serwisową Nordic Semiconductor nRF Connect for Mobile, dostępną na platformy Android oraz iOS (rys. 3.3) [6].

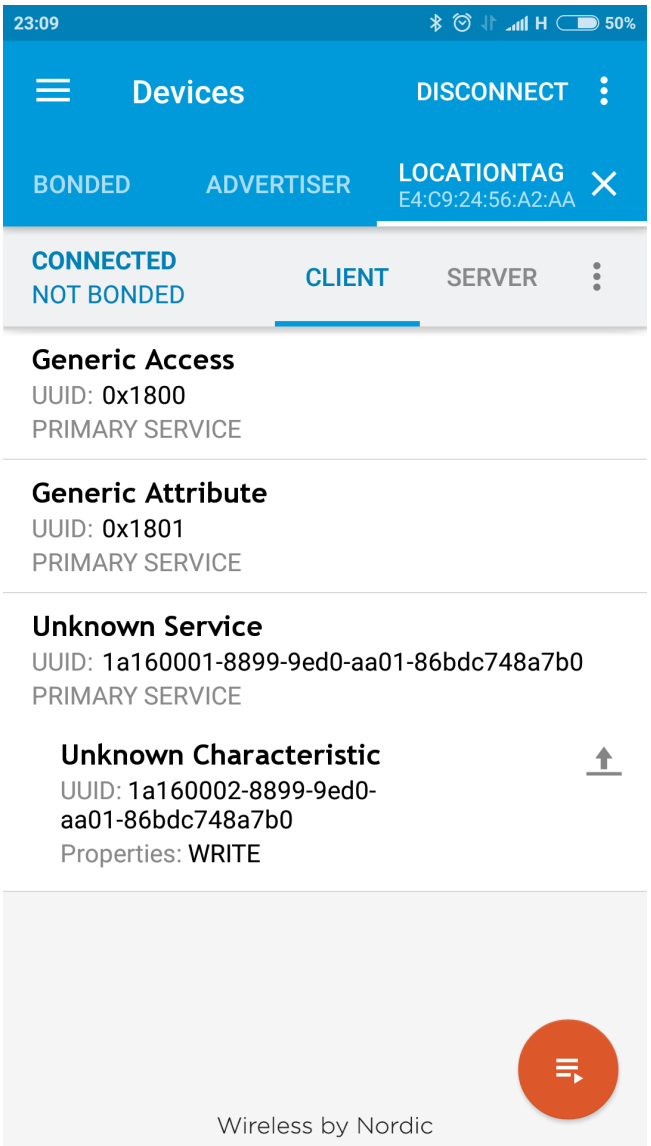
W celu skonfigurowania beacona, należy:

1. Uruchomić aplikację nRF Connect na urządzeniu mobilnym
2. Uruchomić Bluetooth
3. Poczekać, aż na liście pojawi się urządzenie o nazwie „locationTAG"
4. Wybrać opcję „Connect", aby zainicjować połączenie z beaconem (por. rys. 3.3
5. Po nawiązaniu połączenia wybrać ostatni serwis, wybrać ikonę strzałki przy jego charakterystyce (por. rys. 3.4)
6. Wpisać nowy identyfikator grupy jako 8 cyfrową liczbę heksadecymalną i zatwierdzić przyciskiem „Send" (por rys. 3.5)

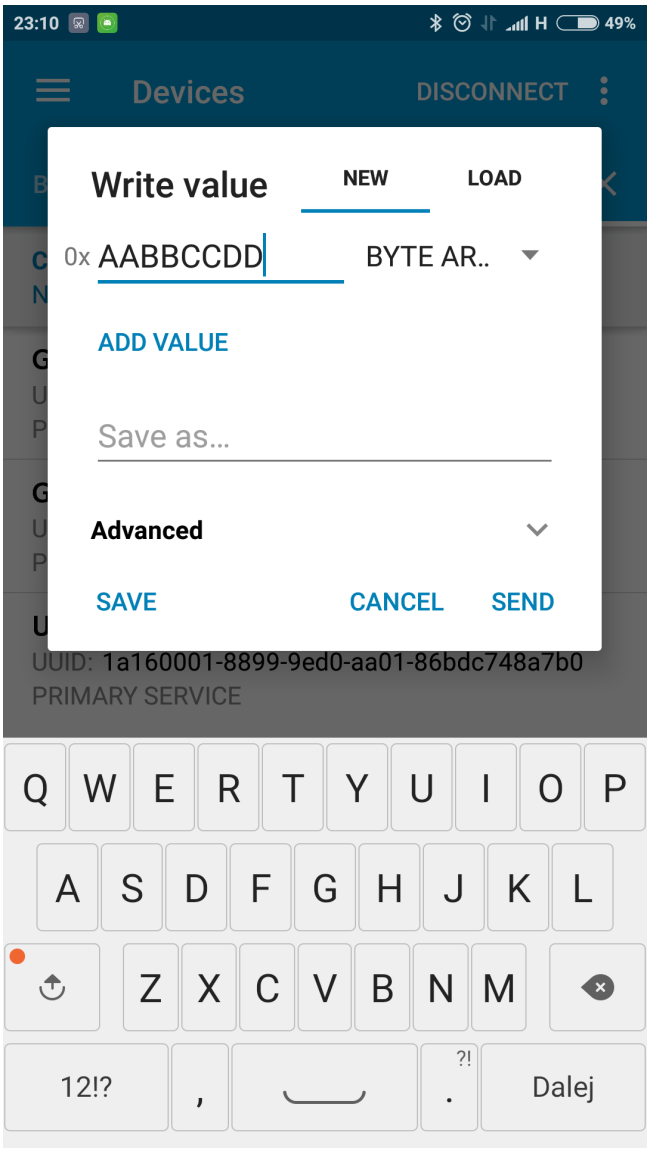


Rysunek 3.3: Aplikacja nRF Connect - widok skanowania

Po zerwaniu połączenia (poprzez przycisk „Disconnect” lub zamknięcie aplikacji) beacon rozpocznie rozgłaszanie z nowym identyfikatorem grupy.



Rysunek 3.4: Aplikacja nRF Connect - lista serwisów i charakterystyk. Charakterystyka i serwis są przedstawiane jako “unknown” (ang. nieznane), ponieważ nie są to standardowe atrybuty opisane przez specyfikację Bluetooth, i stąd nierozpoznane przez aplikację



Rysunek 3.5: Aplikacja nRF Connect - zapis do charakterystyki konfiguracyjnej

Rozdział 4

Aplikacja nasłuchująca

Aby możliwe było wykorzystanie informacji niesionej przez rozgłaszanie, konieczne jest użycie aplikacji nasłuchującej. Zadanie nasłuchu samo w sobie jest trywialne, bowiem algorytm nasłuchiwania został zaimplementowany we wszystkich sterownichach Bluetooth. Jednak do zbudowania użytecznego systemu lokalizacji na podstawie RSSI konieczne jest spełnienie pewnych dodatkowych wymagań:

Nasłuch ciągły - większość interfejsów do sterowników Bluetooth udostępnia możliwość skanowania w poszukiwaniu rozgłaszających beaconów przez pewien określony czas, po czym skanowanie jest przerywane. Tymczasem system lokalizacji wymaga, aby skanowanie przeprowadzać w sposób ciągły. Optymalnie, powinno być możliwe przechwycenie każdego pakietu rozgłoszeniowego.

Filtrowanie urządzeń - w środowisku pracy systemu potencjalnie mogą się znaleźć inne urządzenia BLE. Aplikacja nasłuchująca powinna je odfiltrować

Interfejsy do dalszego przetwarzania - zebrane w postaci pakietów rozgłoszeniowych dane powinny być udostępnione do dalszego przetwarzania poprzez odpowiedni interfejs.

Ponieważ robot, na którym system ma być docelowo wdrożony, działa pod kontrolą systemu ROS, zdecydowano zaimplementować oprogramowanie nasłuchujące jako aplikacje ROS.

ROS umożliwia pisanie aplikacji w trzech językach programowania [5] [4]:

- C++
- Python
- JavaScript (Node.js)

Język C++ pozwala na napisanie programu bardzo wydajnego obliczeniowo, ponieważ jest to język kompilowany. Jednakże przygotowanie programu i jego zmiany wymagają relatywnie dużego nakładu pracy. Język Python, jako język interpretowany, znacznie gorzej radzi sobie w zadaniach obliczeniowych, jednak programy są łatwiejsze do napisania i utrzymania ze względu na prostą składnię, wysokopoziomowość i bogaty zasób bibliotek. Język JavaScript podobnie jak Python, ma przeciętną wydajność obliczeniową. Jednak biblioteki ROS w języku JavaScript zostały zaimplementowane stosunkowo niedawno i można się spodziewać, że zbiór bibliotek jest niekompletny lub niestabilny.

Ostatecznie, do implementacji aplikacji nasłuchującej wybrano język Python. Wydajność obliczeniowa nie jest problemem, ponieważ aplikacja nie wykonuje żadnych złożonych obliczeń na dużych zbiorach danych. Natomiast elastyczność języka Python pozwoliła na zaprojektowanie aplikacji w taki sposób, aby możliwe było łatwe jej modyfikowanie i rozszerzanie, przy relatywnie niewielkim nakładzie pracy.

4.1 Integracja z API Bluetooth

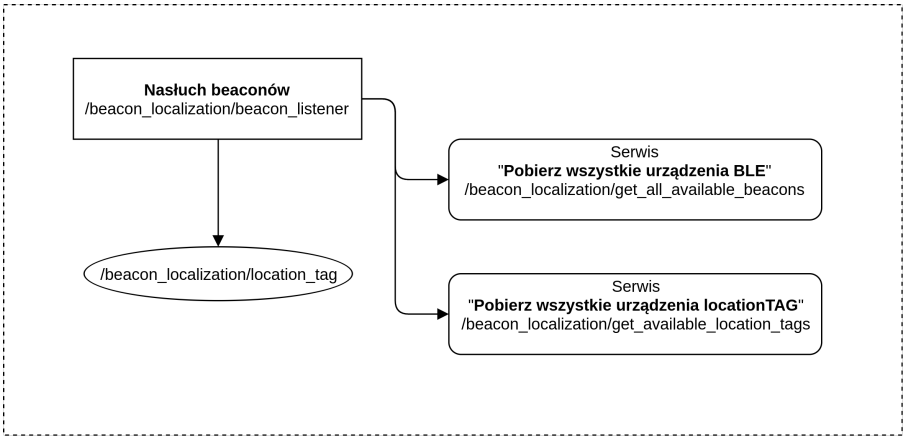
Ponieważ oprogramowanie ROS działa w systemie operacyjnym Linux, konieczne jest zintegrowanie aplikacji ROS ze sterownikiem Bluetooth. W większości dystrybucji systemu Linux, Bluetooth działa pod kontrolą stosu BlueZ, będącego oficjalną implementacją stosu Bluetooth dla systemów Linux.

Do integracji aplikacji w języku Python z API Bluetooth konieczne jest zastosowanie dodatkowej biblioteki. Ponieważ implementacja takiej biblioteki jest zadaniem złożonym i wykraczającym poza zakres niniejszej pracy, wykorzystano bibliotekę bluepy autorstwa Iana Harvey'a. Biblioteka ta stanowi interfejs API BlueZ w języku C do języka Python [8].

4.2 Architektura aplikacji

Tor przetwarzania aplikacji rozłożony jest na dwa wątki wymieniające dane za pomocą chronionej struktury danych.

API biblioteki BlueZ jest zaprojektowane jako synchroniczne. Oznacza to, że funkcja odpowiedzialna za wykonanie skanowania BLE zwraca dopiero po zakończeniu skanowania. Dlatego, aby zapewnić responsywność programu, skanowanie jest uruchamiane w oddzielnym wątku. Funkcja jest wywoływana w pętli, wykonując skanowanie o zadanej długości. Pętla może być przerwana przez odpowiednią funkcję i wtedy wątek kończy pracę. Każde znalezione podczas skanowania urządzenie jest dodawane do kontenera. Kontener jest oparty na mapie (w języku Python nazywanej słownikiem, ang. *dictionary*), w której kluczem jest adres fizyczny MAC



Rysunek 4.1: Struktura aplikacji nasłuchowej ROS

beacona, zaś wartością - obiekt przechowujący informacje odebrane z rozgłaszania. Kontener został wzbogacony o dwa dodatkowe mechanizmy:

Usuwanie starych wpisów - wpisy, reprezentujące pakiety rozgłoszeniowe, starsze niż zadana wartość mogą zostać usunięte z kontenera

Ochrona dostępu - dla zapewnienia bezpiecznej synchronizacji między wątkami, każdy dostęp do danych w kontenerze (odczyt, pobranie całej zawartości, dodanie nowego wpisu) jest chroniony za pomocą obiektu *Lock*, zapewniającego atomowość dostępu.

W wątku głównym programu prowadzone jest publikowanie wiadomości zawierających dane rozgłoszeniowe. W tym celu aplikacja wykorzystuje obiekt opakowujący klasę *Publisher* biblioteki ROS [3]. Jego rolą jest odfiltrowanie urządzeń nienależących do systemu oraz posiadających nieprawidłowy identyfikator grupy (por. 3.4.1).

Ustalono, że wszystkie aplikacje systemu będą działać w przestrzeni nazw *beacon_localization*. Aplikacja nasłuchująca działa pod nazwą *beacon_listener*. Znalezione urządzenia są publikowane do tematu o nazwie *location_tag*. Dodatkowo aplikacja udostępnia dwie usługi:

- *get_all_available_beacons* - zwracającą wszystkie znalezione urządzenia
- *get_available_location_tags* - zwracającą urządzenia typu locationTAG

Schemat węzłów (aplikacji) i tematów (ang. *topics*) ROS systemu przedstawiono na rys. 4.1.

Rozdział 5

Testy rozwiązania

Zaimplementowane rozwiązanie, to jest: oprogramowanie beacona oraz oprogramowanie ROS, zostało przetestowane pod kątem prawidłowej pracy. W pierwszej fazie rozwoju projektu poszczególne elementy systemu testowano oddzielnie. W tym celu wykorzystano wspomnianą w rozdziale 3 aplikacją Nordic Semiconductor nRF Connect. Posiada ona możliwość nasłuchu rozgłaszania BLE. Tą funkcjonalność wykorzystano do testowania oprogramowania beacona. Ponadto, najnowsza wersja nRF Connect posiada także możliwość skonfigurowania smartfona aby działał jak beacon. Tak przygotowana konfiguracja została wykorzystana do testowania aplikacji nasłuchującej, szczególnie pod kątem filtrowania obcych urządzeń oraz poprawności parsowania pakietów.

5.1 Narzędzia testowe

Po sprawdzeniu poprawności implementacji, tj. stwierdzeniu, że postawione systemowi wymagania implementacyjne zostały spełnione, przystąpiono do badania systemu pomiaru odległości jako takiego. W tym celu konieczne było zebranie charakterystyki siły sygnału RSSI w funkcji odległości. Do wykonania tych badań oraz obróbki wyników zaprojektowano odpowiednie narzędzia.

5.1.1 Program RSSI Profiler

Program ten jest niewielką aplikacją ROS napisaną w języku Python. Jej zadaniem jest udostępnianie usługi *rssi_profile* o następującym schemacie:

```
float64 distance
uint16 count
sting bid
---
```

```
float64 avg_rssi
float64 std_dev
float64[] measurements
```

Schemat usługi ROS składa się z zapytania i odpowiedzi, rozdzielonych linią z trzema myślnikami. Tak sformułowany plik jest kompilowany do odpowiednich plików źródłowych języka C++ i Python, w celu implementacji logiki usługi [5]. W wypadku usługi RSSI Profile, zapytanie zawiera dwie dane: *distance* - odległość pomiędzy beaconem a odbiornikiem w metrach, *count* - liczba próbek w pomiarze, oraz *bid* (Beacon ID, por 3.4.1), identyfikator badanego beacona.

Odpowiedź zawiera średnią wartość RSSI z pomiaru (*avg_rssi*), odchylenie standardowe (*std_dev*) oraz tablicę z poszczególnymi pomiarami (*measurements*).

5.1.2 Program Scribe

Program Scribe (ang. *skryba*) jest prostym skryptem napisanym w języku Python, mającym na celu wywołanie usługi *rssi_profile* i zapisanie rezultatu do pliku. Jakkolwiek usługi ROS można wywoływać bezpośrednio z konsoli Bash za pomocą narzędzia rosservice, sposób wyświetlania odpowiedzi nie jest wygodny do zapisywania do pliku celem dalszej obróbki. Program Scribe zapisuje pomiary w formacie TSV (ang. *Tab Separated Values*, wartości rodzielone znakiem tabulacji). Tak sformatowany plik można łatwo zaimportować do dowolnego narzędzia przetwarzania danych, począwszy na arkuszu kalkulacyjnym, poprzez bibioteki takie jak numpy, skończywszy na pakiecie MATLAB.

Program Scribe może pracować w dwóch trybach. Tryb pojedynczego pomiaru pozwala na zebranie wielu wartości RSSI dla pojedynczej zadanej odległości. Do pliku zapisywane są wszystkie zebrane wartości RSSI. Takie dane można przedstawić np. w postaci histogramu. Tryb wielu pomiarów pozwala wielokrotnie wywoływać program i dopisywać do pliku kolejne punkty pomiarowe. W trybie wielu pomiarów zapisywana jest tylko średnia wartość RSSI i jej odchylenie standardowe. Pozwala to wykreślić charakterystykę RSSI w funkcji odległości.

5.1.3 Analiza i wizualizacja danych

Do analizy i wizualizacji danych wykorzystano biblioteki numpy, scipy oraz matplotlib dla języka Python. Pierwsza z nich służy do wykonywania obliczeń numerycznych, udostępniając API podobne do pakietu MATLAB, druga zawiera m. in. zaawansowane metody optymalizacji i dopasowania funkcji, ostatnia zaś służy do rysowania wykresów [9].

5.2 Metodyka testów

Testy urządzenia wykonano w pomieszczeniu o wymiarach 2,1 x 15 x 2,5 m (korytarz). Poziom zakłóceń radiowych należy uznać za wysoki, jako że w pobliżu znajduje się dużo nadajników sieci Wi-Fi, działających na tym samym paśmie radiowym co Bluetooth.

Urządzeniem odbiorczym był laptop Lenovo Y50. Beacon oraz laptop ustawiono na wysokości 0,5 m. Dla każdej odległości zebrano 30 wartości RSSI. Dodatkowo dla odległości 1m i 10m zebrano po 300 pomiarów, celem wykreślenia histogramu.

5.3 Wyniki testów

Wykres 5.1 pokazuje charakterystykę RSSI(d) z zaznaczonymi odchyleniami standardowymi, które można traktować jako niepewności pomiarowe typu A. Wykreślono także dopasowaną do pomiarów funkcję postaci:

$$RSSI(d) = A \cdot e^{-Bx} + C \tag{5.1}$$

Dopasowania funkcji dokonano za pomocą funkcji *curve_fit* z biblioteki scipy. W wyniku dopasowania otrzymano następujące wartości współczynników:

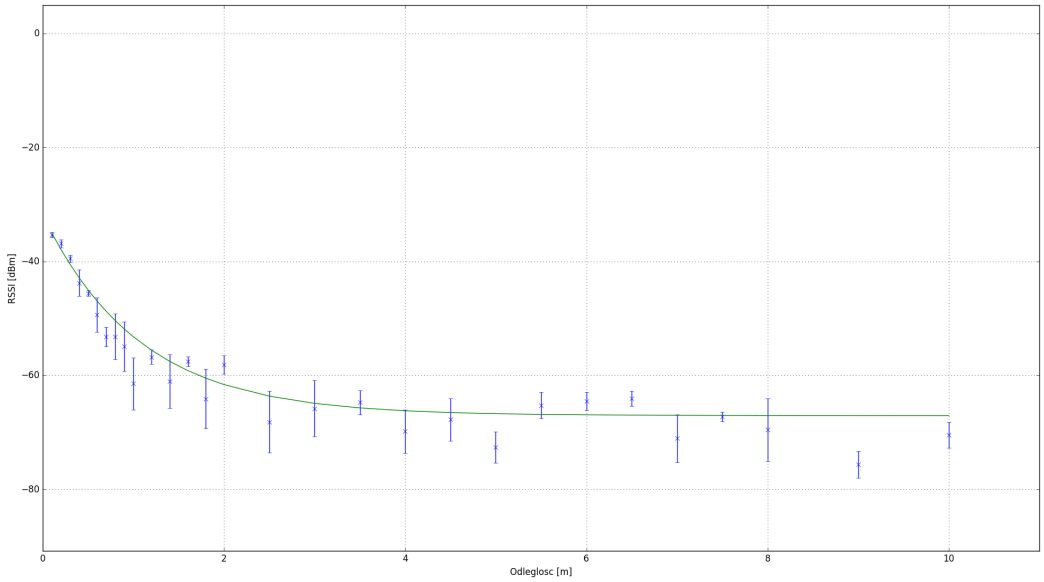
$$A = 35,18 \quad B = 0,9329 \quad C = -67,05 \tag{5.2}$$

Wykres ukazuje znaczny rozrzut wartości RSSI. Odchylenie standardowe sięga 5 dBm i nie zależy od odległości.

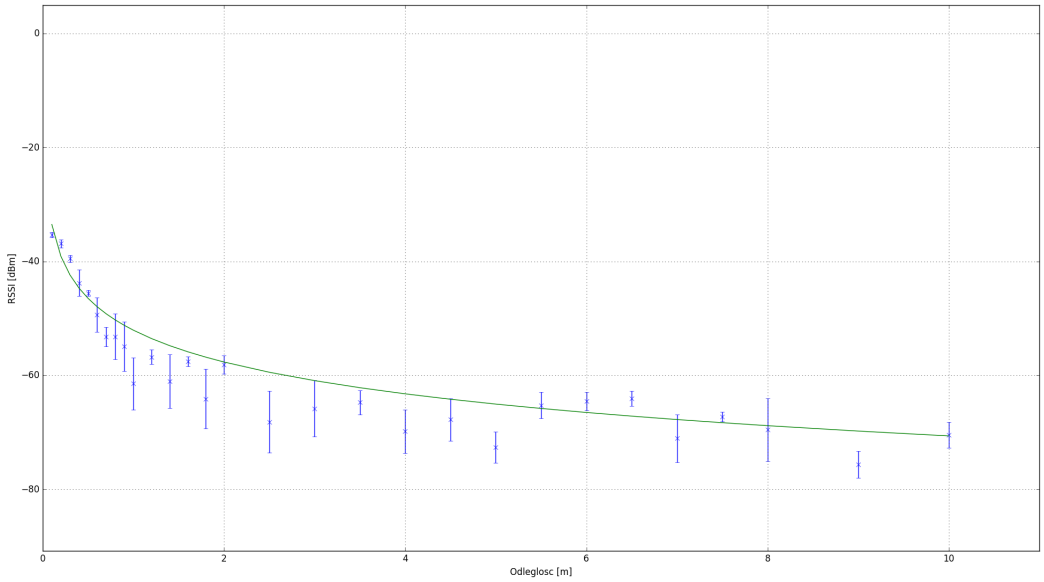
Choć z podstawowego modelu propagacji fali radiowej wynika, że siła sygnału w funkcji odległości powinna być opisywana funkcją logarytmiczną, dopasowanie funkcji eksponencjalnej dało lepszy rezultat (por. wykres 5.2).

Analizując histogramy (wykresy 5.3 i 5.4), można zauważyć że rozkład wartości nie jest rozkładem Gaussa. Zauważalna jest wyraźna asymetria w kierunku wyższych wartości. Można się spodziewać, że taki rozkład będzie utrudniał wykorzystanie pomiarów w lokalizacji za pomocą filtru Kalmana, który wymaga, aby pomiary obarczone były gaussowską niepewnością.

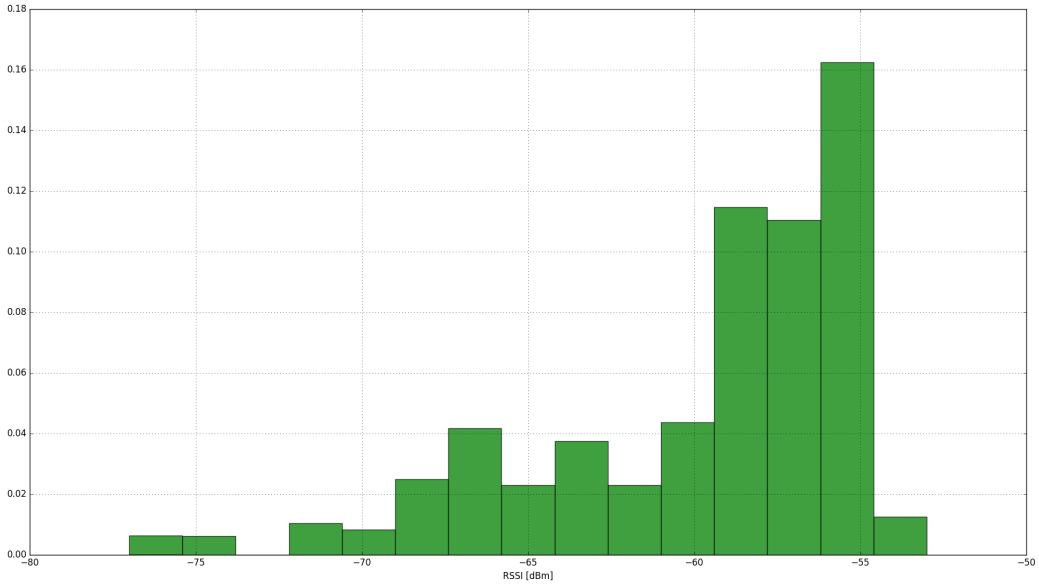
Należy zaznaczyć, że dopasowane wartości są słuszne tylko dla środowiska, w którym wykonano pomiar. Ze względu na naturę propagacji fal radiowych w pomieszczeniach (zjawisko wielokrotnego odbicia itp.), oraz z powodu licznych źródeł zakłóceń, wyznaczony model przeliczania RSSI na odlegość nie będzie słuszny dla innego pomieszczenia.



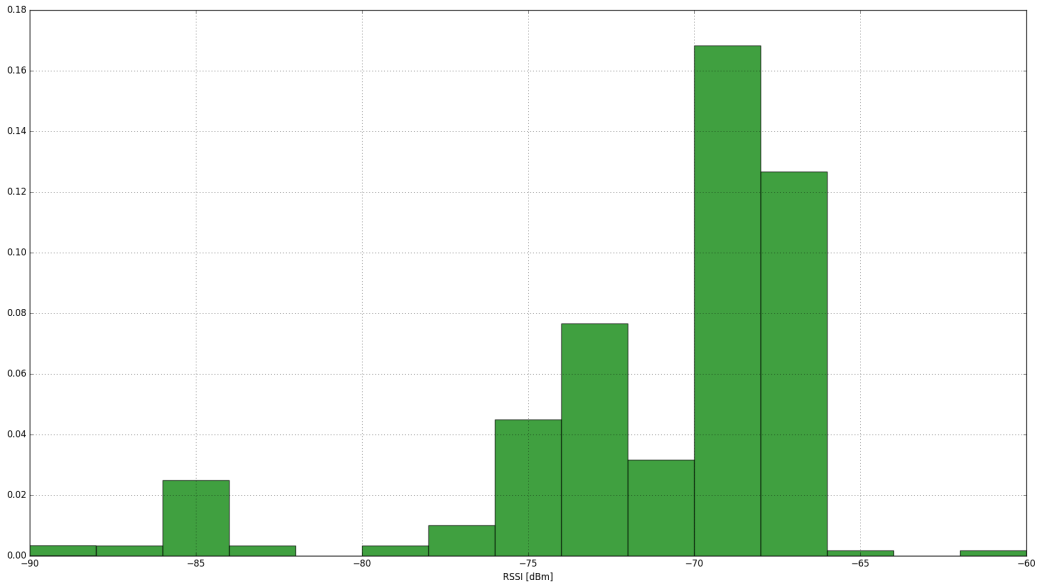
Rysunek 5.1: Charakterystyka RSSI(d) z dopasowaną funkcją eksponencjalną



Rysunek 5.2: Charakterystyka RSSI(d) z dopasowaną funkcją logarytmiczną



Rysunek 5.3: Histogram wartości RSSI dla odległości 1 m



Rysunek 5.4: Histogram wartości RSSI dla odległości 10 m

Rozdział 6

Podsumowanie

Wybór technologii Bluetooth Low Energy do implementacji znacznika radiowego okazał się trafny. Dostępność bibliotek programistycznych w języku Python pozwoliła na szybkie zaimplementowanie prototypu do testów, a następnie właściwej aplikacji nasłuchującej. Także obecność gotowej aplikacji nRF Connect na platformę Android znacznie ułatwiła testy rozwiązania i dostarczyła wygodne narzędzie do konfiguracji systemu.

Architektura zaimplementowanego oprogramowania pozwala na korzystanie z wielu znaczników. W razie potrzeby, możliwe jest zastąpienie technologii BLE inną technologią radiową przy relatywnie niewielkich zmianach w aplikacji nasłuchującej, dzięki wyabstrahowaniu warstwy gromadzącej dane od technologii radiowej.

Przeprowadzone testy wykazały znaczny rozrzut siły sygnału RSSI dla stałej odległości. Jest to wynik zgodny z oczekiwaniami, choc przekreśla on zastosowanie RSSI do precyzyjnego mierzenia odległości. Zakładając zmienne warunki propagacji fal elektromagnetycznych (a zatem brak precyzyjnego modelu matematycznego przenoszącego RSSI na odległość), pomiar można traktować jedynie jako zgrubny, z dokładnością rzędu metrów. Przykładowym zastosowaniem może być stwierdzanie, czy robot znajduje się w danym pomieszczeniu etc.

Jednakże, zastosowanie kilku znaczników i zaawansowanych algorytmów filtracji, predykcji oraz fuzji sensorycznej rokuje na znacznie lepsze wyniki w zadaniu lokalizacji [1]. Dlatego też zaprojektowane znaczniki zostaną wykorzystane do realizacji pracy magisterskiej, dotyczącej lokalizacji robota mobilnego w pomieszczeniu za pomocą znaczników radiowych.

Bibliografia

[1] Aswin N Raghavan, Harini Ananthapadmanaban, Manimaran S Sivamurugan, and Balaraman Ravindran. Accurate mobile robot localization in indoor environments using bluetooth. *ANRH Ananthapadmanaban*, 2010.

[2] Bing-Fei Wu, Cheng-Lung Jen, and Kuei-Chung Chang. Neural fuzzy based indoor localization by kalman filtering with propagation channel modeling. *IEEE International Conference on Systems, Man and Cybernetics*, 2007.

[3] Jason M. O'Kane. *A Gentle Introduction to ROS*. University of South Carolina, 2013.

[4] Aaron Martinez and Enrique Fernández. *Learning ROS for Robotics Programming*. Packt Publishing, 2013.

[5] Dokumentacja online pakietu ROS. `wiki.ros.org`. Dostęp: 2017-07-11.

[6] Dokumentacja sdk mikrokontrolerów rodziny nrf51. `https://infocenter.nordicsemi.com/index.jsp`. Dostęp: 2017-07-11.

[7] Specyfikacja standardu Bluetooth. `https://www.bluetooth.com/specifications/bluetooth-core-specification`. Dostęp: 2017-07-11.

[8] Dokumentacja biblioteki PyBlueZ. `https://github.com/karulis/pybluez/wiki`. Dostęp: 2017-07-11.

[9] Dokumentacja biblioteki SciPy. `https://docs.scipy.org/`. Dostęp: 2017-07-11.

Wykaz skrótów

AES	Advanced Encryption Standard
API	Application Programming Interface
BLE	Bluetooth Low Energy
FHSS	Frequency Hopping Spread Spectrum
GATT	Generic Attibute
GCC	GNU Compiler Collection
ISM	Industrial, Scientific, Medical (pasmo częstotliwości)
MAC	Media Access Control
RAM	Random Access Memory
ROS	Robot Operating System
RSSI	Radio Signal Strength Indicator
SDK	Software Development Kit
UHF	Ultra High Frequency