České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Lokalizace zdroje rádiového signálu senzorovou sítí.

Praha, 2009 **Autor: Marek Juras**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Chtěl bych poděkovat především vedoucímu práce ing. Jiřímu Trdličkovi za cenné připomínky a rady k této bakalářské práci. Dále bych chtěl poděkovat také rodičům a všem ostatním, kteří mě podporovali ve studiu.

Abstrakt

Náplní této bakalářské práce je lokalizace zdroje rádiového signálu senzorovou sítí. Práce obsahuje upravenou implementaci volně šiřitelného vizualizačního programu Octopus Dashboard, který je určen pro použití v bezdrátových senzorových sítích s operačním systémem TinyOs. Aplikace je použita pro měření síly přijatého rádiového signálu v místnostech, kde vzdálenost mezi senzory přesahuje 2 metry. Na naměřená data jsou pro zjištění polohy aplikovány metody rozpoznávání. Ty jsou implementovány v prostředí Matlab. Vyhodnocení výsledků je uvedeno v závěru práce.

Abstract

The content of this bachelor thesis is the tracking of radio transmitter with a sensor network. This thesis contains modified implementation of open-source visualization application Octopus Dashboard. This application is written in nesC and Java languages and is determined for usage in wireless sensor networks with TinyOs operating environment. The application is used for signal strength measuring in rooms. Distances between particular sensors exceed 2 meters. Pattern recognition methods are applied on measured data for determination of the location. These methods are implemented in Matlab environment and their performance evalution is presented in the conclusion of the thesis.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Marek Juras**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Lokalizace zdroje rádiového signálu senzorovou sítí**

Pokyny pro vypracování:

1. Seznamte se s problematikou senzorových sítí, nesC a operačního systému TinyOs.
2. Navrhnete a implementujte program pro senzorovou část systému. (měření dat a jejich komunikace do PC)
3. Vyhledejte a prostudujte metody použitelné pro lokalizaci zdroje rádiového signálu.
4. Připravte experiment a naměřte data potřebná pro testování vybraných metod.
5. Implementujte a otestujte vybrané metody na naměřených datech.
6. Vyhodnoťte.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Jiří Trdlička

Platnost zadání: do konce letního semestru 2009/10

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 13. 3. 2009

Obsah

Seznam obrázků

Kapitola 1

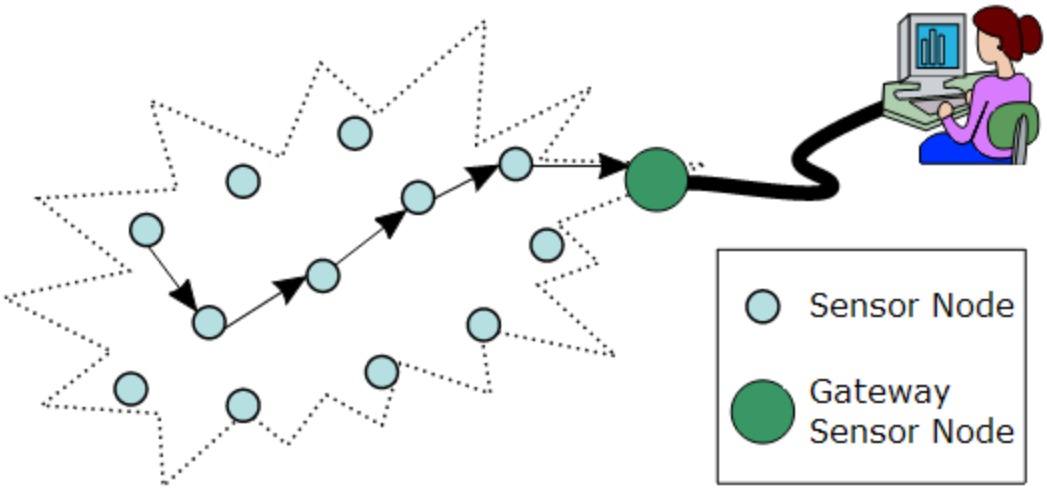
Úvod

1.1 Cíl této práce

Cílem této bakalářské práce je naimplementovat a vyzkoušet použitelnost některých metod lokalizace zdroje rádiového signálu pomocí bezdrátové senzorové sítě na větší vzdálenosti (řádově desítky metrů), čehož by se dalo využít pro monitorování osob či objektů uvnitř budov. Tato práce volně navazuje na práci dřívější [2], která se zabývala metodami lokalizace na kratší vzdálenosti (řádově desítky centimetrů), a která využívala metod založených na přímém měření intenzity rádiového signálu a její převedení na vzdálenost pomocí závislosti intenzity na vzdálenosti. Experimenty však prokázaly, že tento způsob lokalizace není příliš vhodný, protože zmíněná závislost intenzity rádiového signálu na vzdálenosti je nejen nelineární, ale dokonce není ani monotónní (obsahuje lokální maxima). Tuto nelinearitu způsobuje anténa senzoru, která je integrována do desky plošného spoje. Ve své práci se zabývám metodami lokalizace, které nevyužívají síly signálu na přímé určení vzdálenosti, ale jen nepřímo pomocí strojového učení. Zejména se v této práci zabývám metodou k nejbližších sousedů. Prováděl jsem také testování metody Support Vector Machines.

1.2 Co je to senzorová síť?

Bezdrátová senzorová síť je distribuovaný bezdrátový systém složený typicky z velkého počtu autonomních zařízení (angl. motes) využívajících senzorů ke kooperativnímu monitorování fyzikálních veličin jako například vlhkost, teplota, světlo nebo třeba síla rádiového signálu. Naměřená data se pomocí přenosových protokolů šíří celou sítí nebo jen její částí k základně (angl. base station nebo také gateway), která mimo to, že zastává funkci obyčejného senzoru, je ještě navíc zodpovědná za sběr dat a umožnění interakce s uživatelem.



Obrázek 1.1: Ilustrace senzorové sítě

V senzorové bezdrátové síti je vždy alespoň jedna základna, která vysílá uživatelské příkazy ke každému zařízení připojeného do sítě. Architekturu bezdrátové senzorové sítě si potom lze představit jako stromovou strukturu, jejíž kořen je tvořen základnou a každý další uzel je tvořen novým připojeným zařízením. Každý účastník sítě potom musí být schopen příkazy vysílané sítí zpracovat, stejně tak jako musí umět vyslat svá naměřená data směrem k základně.

1.3 Aplikace senzorových sítí

V technické praxi je celá řada aplikací, kde se senzorové sítě využívají. Jedná se například o systémy [7], které jsou instalovány na zemědělských pozemcích za účelem měření teploty a vlhkosti. Z naměřených dat pak lze například optimálně dávkovat zavlažování. Senzory lze vybavit tenzometry a nainstalovat senzorovou síť na mostní konstrukci. Z naměřených dat pak lze vizualizovat dynamiku či zatížení mostu.

Další oblastí, kde použití senzorových sítí nachází uplatnění, je lokalizace. V [15] je uveden popis rádiového čipu CC2431, který je vybaven hardwarovým modulem pro výpočet polohy neznámého senzoru. Lokalizace je založena na měření síly rádiového signálu. Musí být ale předem známy pozice tzv. *referenčních uzlů*.

V [16] je k určení polohy senzorovou sítí použita kombinace rádiového signálu a ultrazvuku. Lokalizace je založena na různé době šíření rádiového a ultrazvukového signálu. Z rozdílu časů je potom určena správná poloha.

Kapitola 2

Hardwarové prostředky

2.1 Senzory

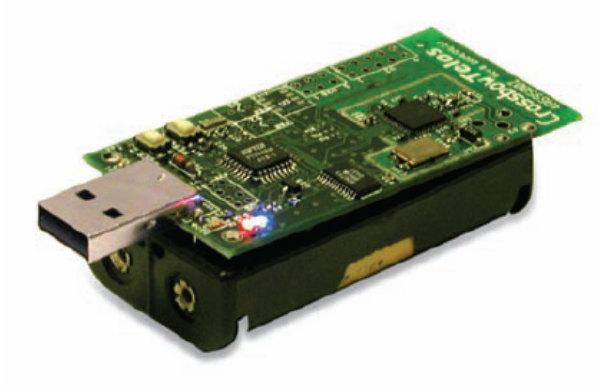
Použitá zařízení (senzory) v senzorové síti, která byla použita v této bakalářské práci, jsou dvojího typu.

- TelosB od firmy Crossbow
- TmoteSky od firmy MoteIV

Jedná se o shodná zařízení od dvou různých výrobců.

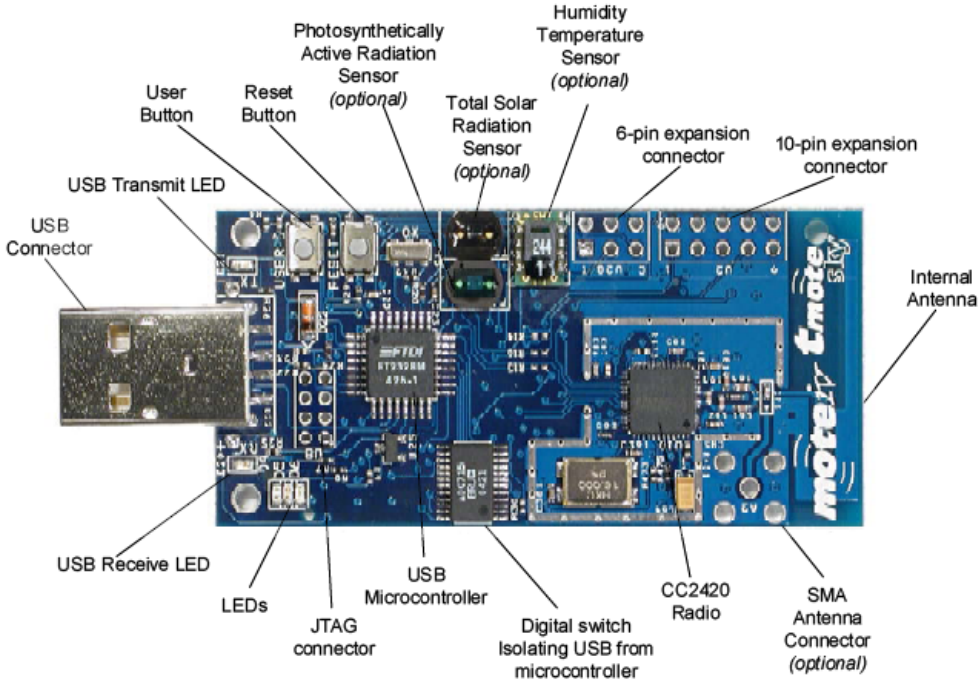
2.2 Popis hardware

Senzor TelosB [6] je vyobrazen na obr. 2.1. Obsahuje 16-ti bitový mikropočítač MSP430 typu RISC od firmy Texas Instruments pracující na frekvenci 8 MHz. Tento mikropočítač je vybaven RAM pamětí o velikosti 10 KB. Programování senzoru se provádí pomocí USB konektoru. Každý senzor je schopen rádiové komunikace na frekvencích od 2,4 do 2,4835 GHz podle standardu IEEE 802.15.4. Přenosová rychlost je 250 kbps.

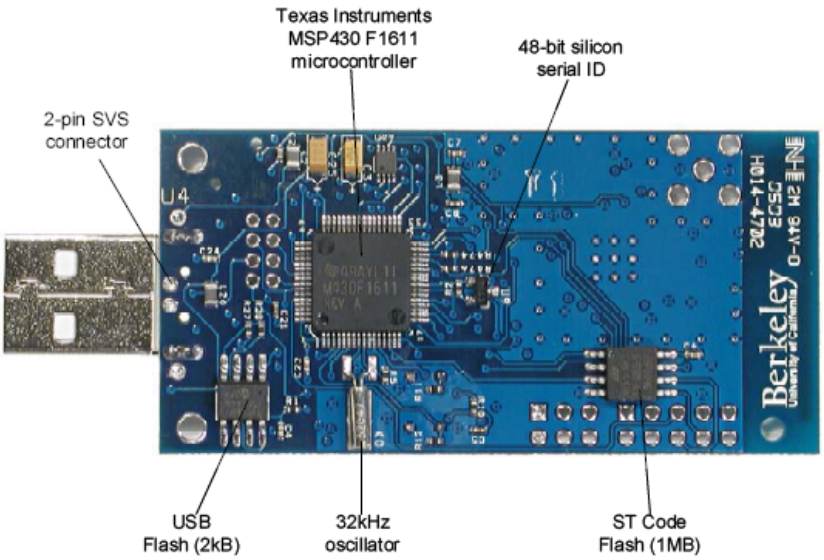


Obrázek 2.1: Senzor TelosB od firmy Crossbow

Zařízení může být osazeno senzorem teploty, světla a vlhkosti. Rozložení součástek je ilustrují obr. 2.2 a obr. 2.3. Pro účely měření v této bakalářské práci se však nevyužije ani jeden z výše uvedených senzorů. Využije se schopností rádiového čipu CC2420 [8].



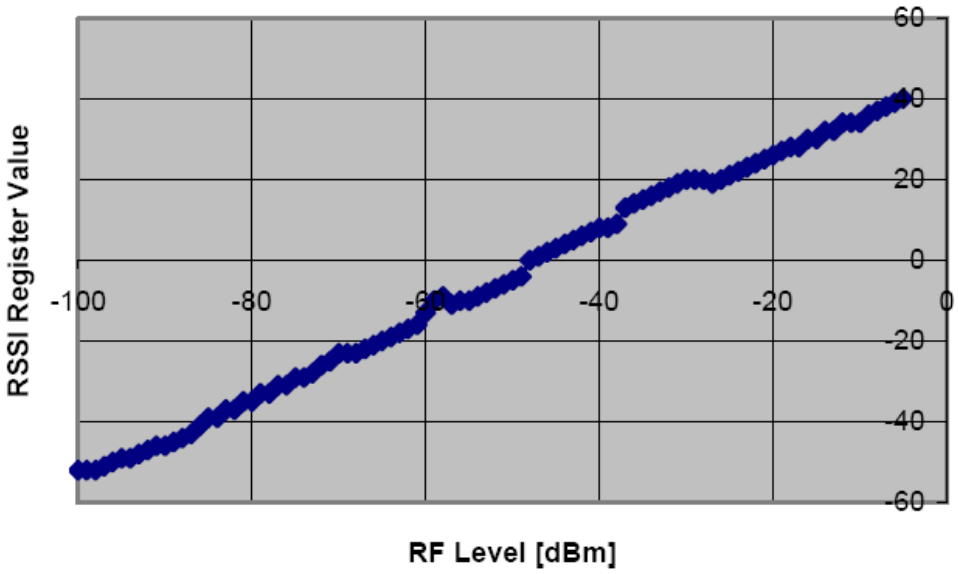
Obrázek 2.2: Rozmístění součástek na plošném spoji (strana A)



Obrázek 2.3: Rozmístění součástek na plošném spoji (strana B)

2.2.1 CC2420

Díky rádiovému čipu CC2420 je senzor schopen měřit veličinu RSSI (Received Signal Strenght Indication). Je to bezrozměrná hodnota, která určuje sílu signálu, s jakou byla přijata zpráva. Na obr. 2.4 je znázorněna převodní charakteristika mezi hodnotou RSSI a odpovídající intenzitou v jednotkách dBm.



Obrázek 2.4: Převodní charakteristika mezi hodnotou RSSI a intenzitou rádiového signálu v jednotkách dBm

Převodní charakteristika je téměř lineární. V této bakalářské práci se však zabývám metodami rozpoznávání, které nejsou závislé na linearitě převodní charakteristiky, ani na použité jednotce, proto nikde nepřepočítávám hodnotu RSSI do jednotek dBm. Z obr. 2.4 je patrno, jakých hodnot reálně nabývá veličina RSSI.

Vysílací výkon lze nastavit na 32 diskrétních úrovní. Vysílací výkon senzorů použitých v této práci byl ponechán na implicitní hodnotě 32, což je maximální výkon.

Kapitola 3

Softwarové prostředky

3.1 Operační systém TinyOs

3.1.1 Historie

První implementace operačního systému TinyOs [4],[5] vznikly v roce 1999 na kalifornské univerzitě v Berkeley. Jednalo se o první verze s označením TinyOs 0.x, které byly naimplementovány jazyce C a Perl. Později v roce 2002 proběhla spolupráce mezi univerzitou v Berkeley a firmou Intel, která dala vzniknout jazyku NesC. Ze září roku 2002 pochází verze TinyOs 1.0, která je kompletně napsána v NesC. V současnosti (květen 2009) je k dispozici nejnovější verze TinyOs 2.1.

Pro účely této bakalářské práce jsem zvolil verzi operačního systému TinyOs 2.0.2 z důvodu kompatibility s vizualizačním systémem Octopus Dashboard, který v této praci využívám.

3.1.2 Architektura

TinyOS je volně šiřitelný operační systém navržený speciálně pro použití v bezdrátových senzorových sítích. Již samotný název napovídá, že se jedná o systém, který vyžaduje velmi málo systémových zdrojů. Tak málo, že může být spuštěn i na mikropočítači s velmi omezenou operační pamětí řádu jednotek kB. Celý systém je řízen událostmi, což má za následek úplné odstranění blokujících operací. Tím dochází k výraznému snížení spotřeby a také ke zrychlení odezvy celého systému.

3.1.3 Komponenty

Každá aplikace, která běží na platformě TinyOS je složena z komponent (angl. components) a z interfaců (angl. interfaces). Komponenty jsou dvojího druhu.

- Moduly (angl. modules), které obsahují implementaci, alokují stav, obsahují programovou logiku
- Konfigurace (angl. configurations), které slouží jako pojivo (angl. wiring) mezi moduly a spolu s využitím interfaců umožňují spojovat dílčí moduly do modulů větších, které jsou mnohdy lépe a přívětivěji využitelné. Oba druhy komponent navíc mohou být jako singletony nebo jako obecně využitelné (generické), které se pak v programu mohou vyskytovat ve více instancích.

3.1.4 Interface

Komunikace mezi komponentami probíhá výhradně za pomocí interfaců, které jsou obousměrné. K interfacům se tedy dá přistupovat z pohledu uživatele (angl. user) nebo z pohledu poskytovatele (angl. provider). Použitím interfaců je dosaženo toho, že komponenty jsou mezi sebou volně vázány a lze z nich poskládat i značně komplexní aplikace.

3.2 NesC

NesC je programovacím jazykem, který je dialektem jazyka C a je používán pro zápis programů pro platformu TinyOS. Je totiž doplněn o podporu komponent, které jsou stavebními kameny TinyOS aplikace.

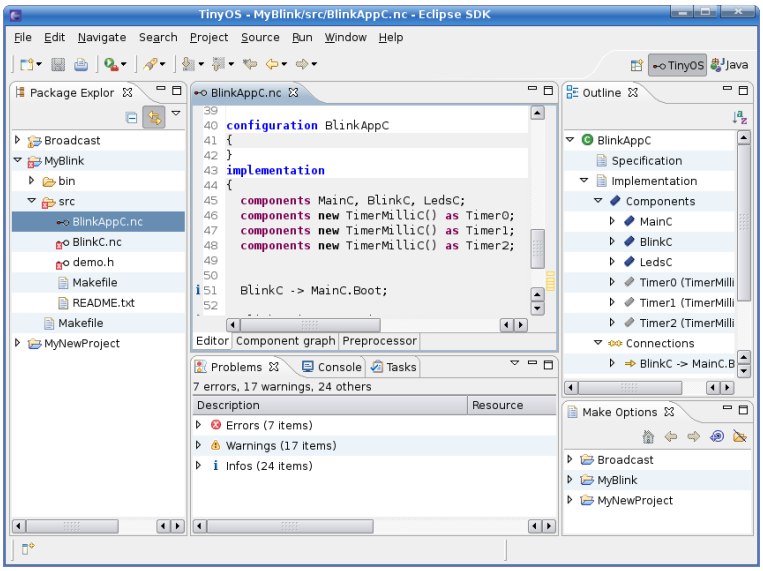
3.3 Vývojové prostředí

Po celou dobu vývoje jsem pracoval s linuxovou distribucí Ubuntu 8.10, do které jsem si doinstaloval podporu pro TinyOs 2.0.2. Prostředí pro vývoj aplikací pro operační systém

TinyOs opravdu není mnoho. Na internetu jsem narazil jen na jedno. Jedná se o plugin do vývojového prostředí Eclipse s názvem Yeti 2.

3.3.1 Yeti 2 Plugin for Eclipse

Jedná se o plugin do populárního vývojového prostředí Eclipse. Plugin je volně ke stažení zde [13]. Ukázka pracovní plochy je zobrazena na obr. 3.1



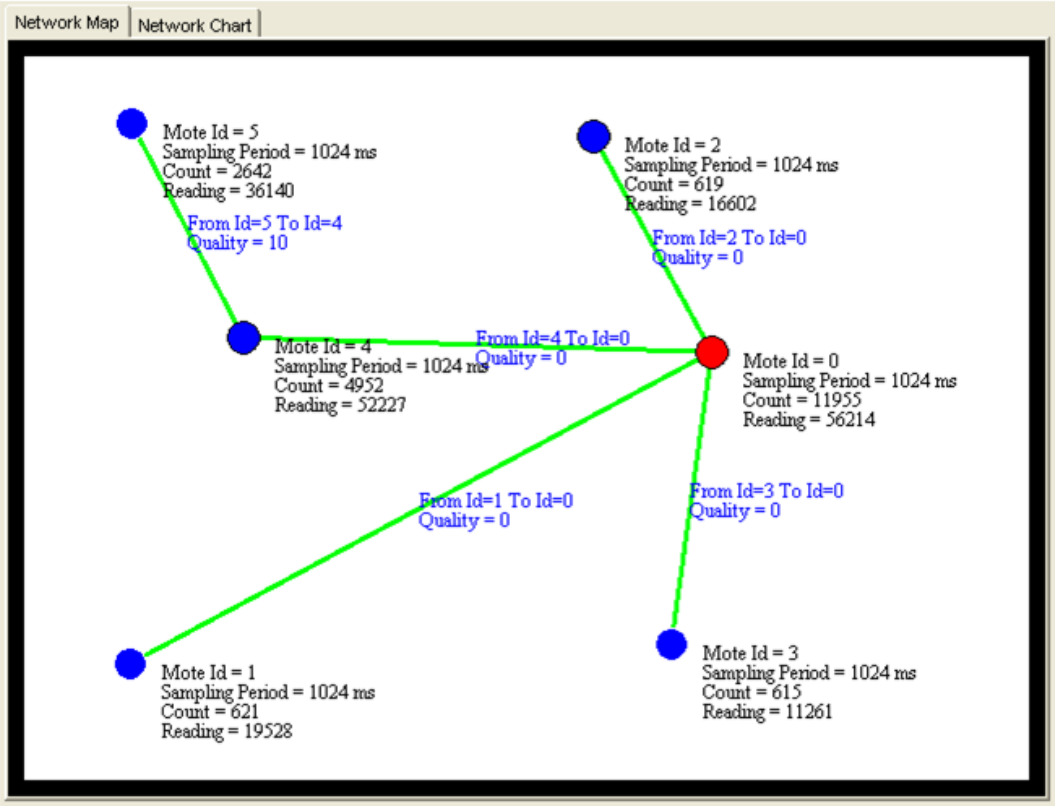
Obrázek 3.1: Ukázka z pracovní plochy pluginu Yeti 2

3.4 Octopus Dashboard

3.4.1 Popis prostředí

Octopus Dashboard [9] (dále jen Octopus) je volně šiřitelný program, který slouží jak pro vizualizaci tak pro řízení senzorové sítě pracující pod operačním systémem TinyOs 2.x. Systém Octopus obsahuje grafické uživatelské rozhraní (GUI), pomocí kterého lze v reálném čase sledovat topologii celé sítě, monitorovat, kdy a mezi kterými dvěma senzory došlo k předání paketu, viz obr. 3.2. Lze také dynamicky, to jest za běhu aplikace, měnit nastavení senzorové sítě. Pomocí rozhraní Octopusu lze snadno měnit například

vzorkovací frekvenci, se kterou senzory posílají naměřené hodnoty. Systém rovněž dovoluje jednoduchou vizualizaci naměřených dat. Ta lze navíc uložit do *.csv souboru a využít tak k dalšímu zpracování.



Obrázek 3.2: Ukázka vizualizace senzorové sítě a komunikace v systému Octopus

3.4.2 Režimy práce

Neméně zajímavou a nesporně dosti dobře využitelnou schopností Octopusu je možnost práce ve dvou režimech. V automatickém (synchronním) nebo manuálním (asynchronním) režimu. V obou režimech (v automatickém i manuálním) je možné komunikovat buď se všemi zapojenými senzory najednou (angl. broadcast) nebo jen s jedním (angl. unicast).

- *Automatický režim*: data ze senzorů posílána s nastavitelnou frekvencí směrem k základně zcela automaticky.

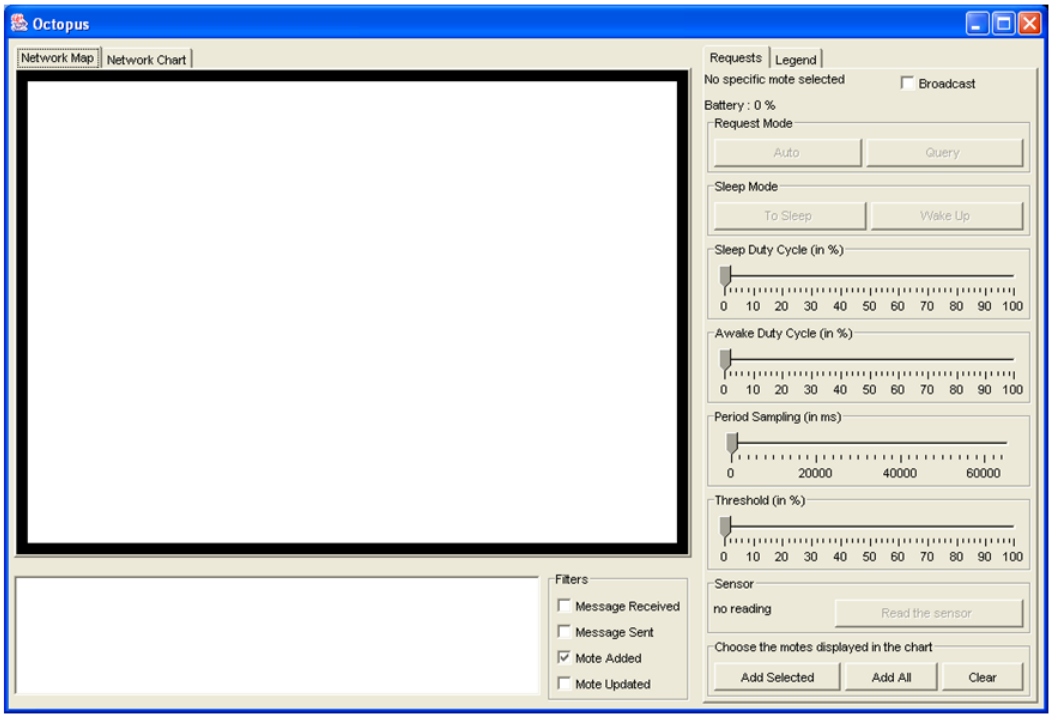
- *Manulání (Query) režim*: data ze senzorů posílána s nastavitelnou frekvencí směrem k základně zcela automaticky.

3.4.3 Programové části systému Octopus

Systém Octopus je úplným řešením vizualizace a řízení senzorové sítě. Je tvořen dvěma úzce spojenými programovými částmi.

3.4.3.1 Java (GUI) aplikace

Java aplikace je první programovou částí. Slouží pro běh na PC a obsahuje grafické uživatelské rozhraní mezi uživatelem a sítí. Na obr. 3.3 je zobrazena základní plocha. Pomocí tohoto rozhraní je uživateli umožněno zadávat příkazy prostřednictvím tlačítek umístěných na řídicím panelu. Rovněž se na konzoli zobrazují data přijatá ze sítě. Tato aplikace komunikuje se základnou pomocí rozhraní USB.



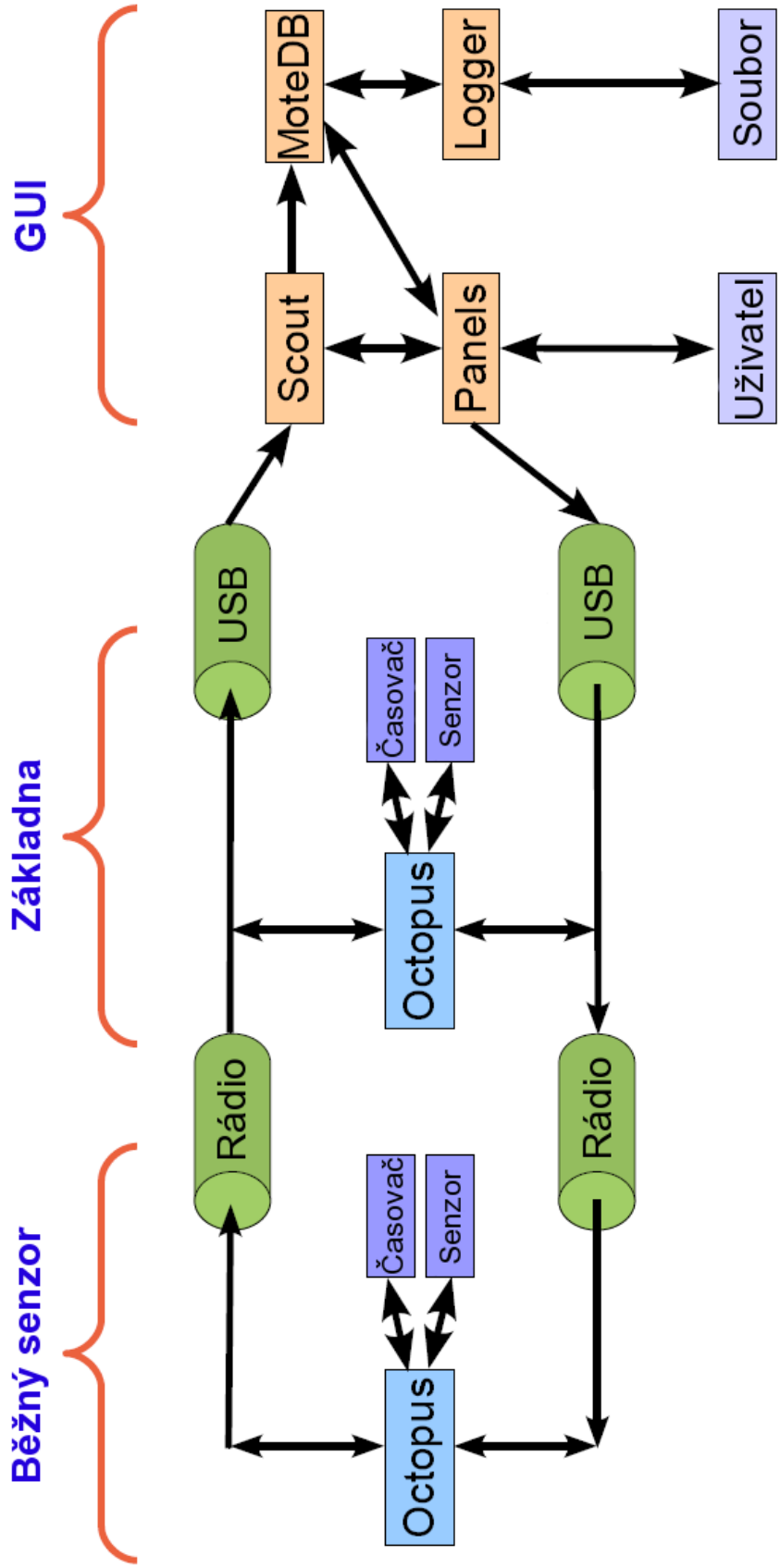
Obrázek 3.3: Pracovní plocha programu Octopus

3.4.3.2 TinyOs (Embedded) aplikace

Druhá programová část je napsána v jazyce NesC. Je tedy určena pro operační systém TinyOs a musí být nainstalována a spuštěna na všech senzorech připojených do systému Octopus.

3.4.4 Propojení obou programových částí Octopusu

Na obr. 3.4 je schematicky znázorněno propojení obou částí. PC aplikace (GUI) je k senzorové síti připojena přes sériové rozhraní USB. Základna (Gateway), která je tvořena obyčejným senzorem, umí navíc posílat data po sériové lince do PC. Z obrázku je také patrno, že PC aplikace (*GUI*) je složena z mnoha bloků. *Scout* je vlákno, které poslouchá na sériovém portu a vykonává příkazy přijaté ze sítě. *MoteDB* je třída, která spravuje databázi právě připojených zařízení a poskytuje nejnovější stavy všech zařízení. Světle modré komponenty na obrázku značí jednotlivé připojené senzory, které jsou vzájemně propojeny rádiovým kanálem (*Rádio*). Hlavní senzor ve funkci *základny* je připojen k PC a zprostředkovává komunikaci mezi ním a *běžnými senzory*.



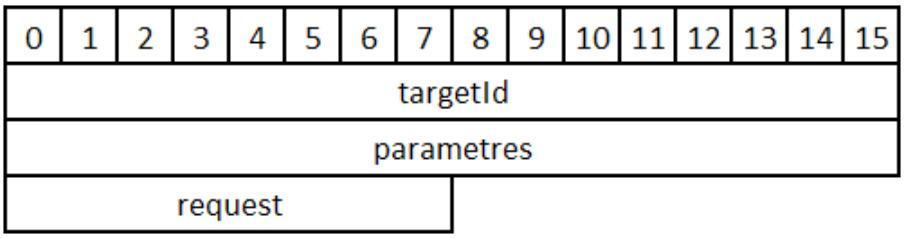
Obrázek 3.4: Propojení jednotlivých programových částí Octopusu

3.4.5 Komunikační protokol

Komunikační protokol systému Octopus je tvořen dvojicí zpráv *OctopusSentMsg* a *OctopusCollectedMsg*.

3.4.5.1 OctopusSentMsg

Zpráva s označením *OctopusSentMsg* (tak je označena ve zdrojovém kódu) je vytvořena pro komunikaci směrem z GUI do základny a také ze základny směrem k ostatním senzorům. Je to zpráva s příkazem od uživatele. Její struktura je znázorněna na obr. 3.5.



Obrázek 3.5: Struktura zprávy *OctopusSentMsg*

Položka *targetId* (16 bitů) slouží jako identifikátor senzoru, kterému je požadavek zasílán. Položka *request* (8 bitů) obsahuje konstantu, která je identifikátorem samotného příkazu. V položce *parametres* jsou případné parametry příkazu. Seznam všech příkazů je uveden v programátorské příručce, kterou lze nalézt zde [9].

3.4.5.2 OctopusCollectedMsg

Zpráva s označením *OctopusCollectedMsg* je komplementární se zprávou *OctopusSentMsg* a slouží pro komunikaci směrem od běžných senzorů k základně a od základny do GUI. Struktura zprávy *OctopusCollectedMsg* je vyobrazena na obr. 3.6.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
moteId															
count															
reading															
quality															
parentId															
replay															

Obrázek 3.6: Struktura zprávy *OctopusCollectedMsg*

V této zprávě je nejdůležitější položka *moteId* (16 bitů), což je identifikátor senzoru, který zprávu vytvořil a položka *reading* (16 bitů), která obsahuje naměřenou hodnotu ze senzoru. Položka *count* (16 bitů) a *quality* (16 bitů) v programu nijak nevyužívám. *parentId* (16 bitů) obsahuje v případě přeposílání zprávy přes prostředníka (tzv. *multihop* komunikace[1]) id senzoru, který zprávu přijal naposled. Poslední položka *replay* (8 bitů) říká, zda se jedná o odpověď na asynchronní dotaz nebo synchronní odeslání naměřených dat. Více v [9].

3.5 Matlab

Pro veškeré zpracování dat nashromážděných pomocí programu Octopus Dashboard jsem použil program Matlab 7.5.0 (R2007b) od firmy The Mathworks. Pro klasifikaci pomocí metody k nejbližších sousedů jsem použil vlastní skripty a pro účely klasifikace pomocí SVM (Support Vector Machines) jsem použil *Statistical Pattern Recognition Toolbox for Matlab* [14], který byl vyvinut na fakultě elektrotechnické ČVUT v Praze.

[1]Pokud senzor (vysílač), chce vysílat data a není v přímém rádiovém dosahu se svým protějškem (přijímačem), sensor (vysílač) využije k doručení dat nejbližších okolních senzorů (prostředníků), které jsou v danou chvíli v rádiovém dosahu. Komunikaci pomocí prostředníků se říká *multihop* komunikace.

3.5.1 STPRToolbox for Matlab

STPRToolbox for Matlab [14] obsahuje implementace algoritmů pro rozpoznávání (angl. pattern recognition). Tento toolbox je neustále ve vývoji. Kromě algoritmu pro Support Vector Machines, který využívám v této práci, toolbox obsahuje například implementaci perceptronového algoritmu i jeho zobecnění pro vícetřídovou klasifikaci, Bayesovský klasifikátor, metodu pro k nejbližších sousedů a další.

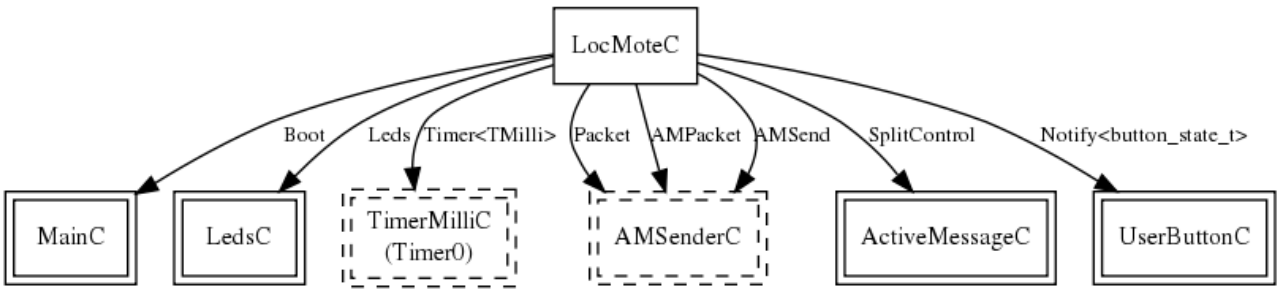
Kapitola 4

Implementace senzorové části

Pro účely této bakalářské práce bylo zapotřebí provést v systému Octopus několik modifikací. Bylo nutné upravit komunikační protokol tak, aby přenášel naměřená data a také bylo zapotřebí vytvořit komponentu, která tato data příjme a rovněž zajistí měření síly přijatého signálu. Navíc byl napsán program, pro vysílač, jakožto senzor, jehož polohu chceme zjišťovat.

4.1 Program pro vysílač - *LocMoteAppC*

LocMoteAppC je aplikace napsaná v jazyku NesC, která zajišťuje, že po stisku uživatelského tlačítka se vyšle po rádiovém kanálu série zpráv (paketů). Diagram aplikace je na obr. 4.1.



Obrázek 4.1: Diagram aplikace pro vysílač - *LocMoteAppC*

Aplikace se chová tak, že čeká na stisk uživatelského tlačítka (*UserButton* na obr. 2.2).

Tento stisk vyvolá událost, která odešle sérii zpráv, kterou zachytí systém Octopus. Zprávu, kterou aplikace *LocMoteAppC* generuje jsem pojmenoval *LocMoteMsg* a její struktura je vyobrazena na obr. 4.2.

0	1	2	3	4	5	6	7
setNumber							
packetNumber							

Obrázek 4.2: Struktura zprávy *LocMoteMsg*

V těle každého paketu se nachází pouze dvojice identifikátorů, která paket jednoznačně identifikuje. Jedná se o

- setNumber - číslo série
- packetNumber - číslo paketu v každé sérii

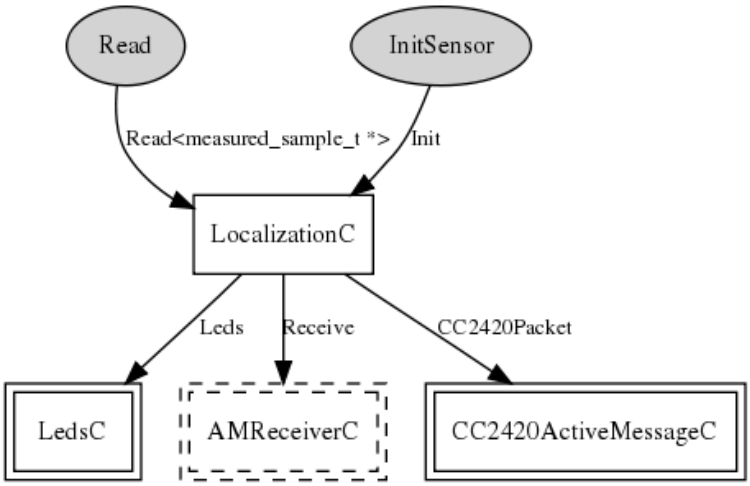
Po každém stisku se inkrementuje hodnota *setNumber* a právě touto hodnotou se odlišují jednotlivé stisky tlačítka. Hodnota *packetNumber* poté identifikuje každý jednotlivý paket v sérii. Čísla sérií a čísla paketů jsou číslována vždy vzestupně počínaje hodnotou 1. Počet paketů v sérii lze nastavit. Implicitně je nastavena hodnota 20. Rovněž časové prodlevy mezi jednotlivými pakety v sérii je možno měnit. Implicitně je nastavena prodleva 200ms. To znamená, že celá série se po stisku tlačítka odešle za 4 sekundy. Samotné odesílání je vizualizováno pomocí 2 signálních LED. Dioda s označením *Led0* svítí, pokud je aktivní vysílání (v případě implicitního nastavení počtu paketů a prodlevy to znamená, že *Led0* svítí celé 4 sekundy) a dioda s označením *Led1* změní svůj stav po každém odeslání jednoho paketu.

Zdrojový kód programu je v adresáři `LocMote` v těchto souborech:

- *LocMote.h*: Hlavičkový soubor s konstantami programu.
- *LocMoteMsg.h*: Hlavičkový soubor s definicí zprávy *LocMoteMsg* (viz. obr. 4.2).
- *LocMoteAppC.nc*: Konfigurační komponenta programu.
- *LocMoteC.nc*: Modul, který obsahuje samotnou implementaci.

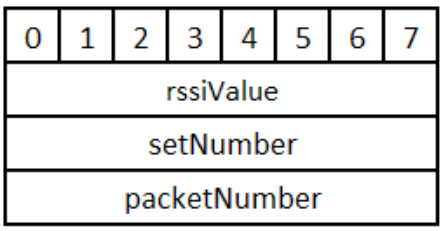
4.2 Komponenta pro příjem dat - *LocalizationAppC*

LocalizationAppC je komponenta, která slouží jako přijímač zpráv ve formátu *LocMoteMsg*. Diagram komponenty je na obr. 4.3.



Obrázek 4.3: Diagram komponenty *LocalizationAppC*

Úlohou komponenty *LocalizationAppC* je přijímat zprávy, které jsou vysílány z vysílače, tyto zprávy uchovávat a poskytovat je systému Octopus, který tyto zprávy pošle do PC. Komponenta obsluhuje hardwarový přijímač senzoru a po přijetí zprávy ve formátu *LocMoteMsg* z ní zjistí číslo série (*setNumber*), číslo paketu (*packetNumber*) a změří velikost RSSI z přijaté zprávy. Tuto trojici dat pak uloží do struktury, která je ve zdrojovém kódu označena jako *measuredSample* a její struktura je na obr. 4.4.



Obrázek 4.4: Datová struktura *measuredSample*

Komponenta *LocalizationAppC* je do stávajícího systému Octopus připojena přes obecně použitelné interfacy *Read* a *Init*. Ty jsou vidět na obr. 4.3. Interface *Init* slouží k iniciali-

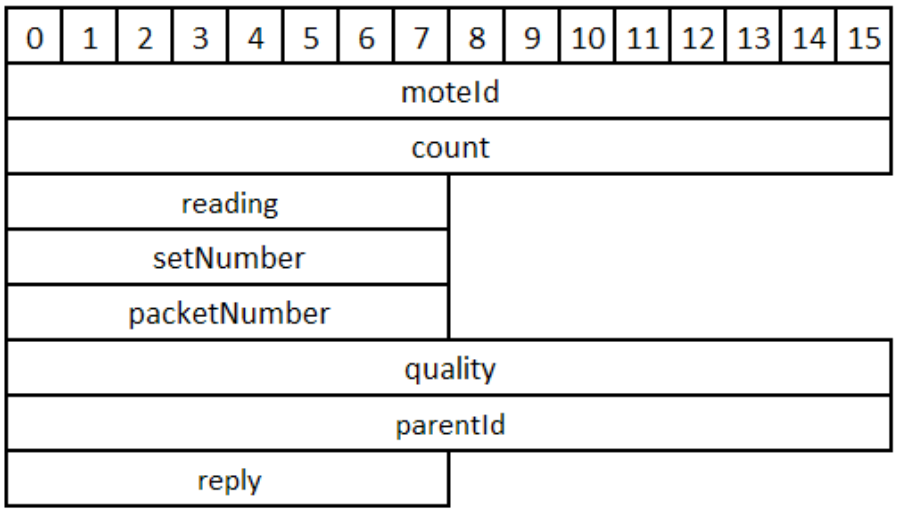
zaci komponenty, tj. k inicializaci vnitřních proměnných. Komunikace systému Octopus se samotnou komponentou je zajišťována přes interface *Read*.

Protože data z vysílače mohou přicházet do Octopusu s větší frekvencí než je vzorkovací frekvence systému Octopus (frekvence, se kterou systém Octopus čte data ze senzoru), je potřeba zajistit, aby se data neztrácela. Po přijetí jsou data převedena do formátu *measuredSample* a poté uložena do kruhové fronty.

Data z fronty jsou postupně odebírána s danou vzorkovací frekvencí systému Octopus. Kdykoliv přijde pořadavek na čtení z fronty, která je prázdná, tj. neobsahuje-li žádná data, vrátí se tzv. *prázdný* vzorek. Od ostatních platných dat se *prázdné* vzorky odlišují tím, že mají číslo série (*setNumber*) a číslo paketu (*packetNumber*) rovny nule. Z důvodu nulovosti těchto položek je lze odfiltrovat. Tyto *prázdné* vzorky lze filtrovat již na úrovni senzorů, tj. k odeslání *prázdných* vzorků vůbec nemusí dojít. Tak by se ale zhoršila odezva celého systému Octopus. Proto se data filtrují až v pozdější fázi.

4.3 Úprava komunikačního protokolu Octopusu

Do struktury zprávy *OctopusCollectedMsg* na obr. 3.6 bylo potřeba přidat dvojici identifikátorů *setNumber* a *packetNumber*, která spolu s hodnotou *reading*, což je hodnota síly rádiového signálu RSSI, tvoří data, která pak jsou směrodatná pro lokalizaci. Upravená struktura zprávy *OctopusCollectedMsg* je na obr. 4.5. Ve zdrojovém kódu je pak fyzicky upravena v souboru *Octopus.h*.



Obrázek 4.5: Modifikovaná struktura zprávy *OctopusCollectedMsg*

4.4 Úprava javovské GUI aplikace

Úprava komunikačního protokolu si vyžádala zásah do následujících tříd v programu

- *Mote.java*: Jedná se o třídu, jejíž instance představují jednotlivé senzory v síti. Byly přidány proměnné *setNo* a *packetNo*, které představují číslo série, resp. číslo paketu přijatého záznamu.
- *Scout.java*: Vlákno, které poslouchá na sériovém vstupu zprávy od základny. Metoda *messageReceived()* byla upravena tak, aby po přijetí nové zprávy docházelo k nastavení proměnných *setNo* a *packetNo*. Také byl přidán výpis hodnot těchto proměnných na konzoli.
- *Logger.java*: Tato třída se stará o logování všech přijatých dat do *.csv souboru. Byly přidány položky *SN (Set number)* a *PN (Packet number)*, jakožto nově přidané sloupce do struktury *.csv souboru.
- *ChartPanel.java*: Tato třída se stará o orientační zobrazení naměřených dat ve formě grafu na panelu. Byl změněn počet hodnot na ose y z původních 65535 na 256.

Kapitola 5

Experiment

5.1 Cíl experimentu

Cílem tohoto experimentu bylo rozmístit v uzavřeném objektu senzory, vytvořit z nich bezdrátovou senzorovou síť a pořídit dva soubory dat. Trénovací a testovací množinu. Na získaná data poté aplikovat metody rozpoznávání a vyzkoušet jejich použitelnost.

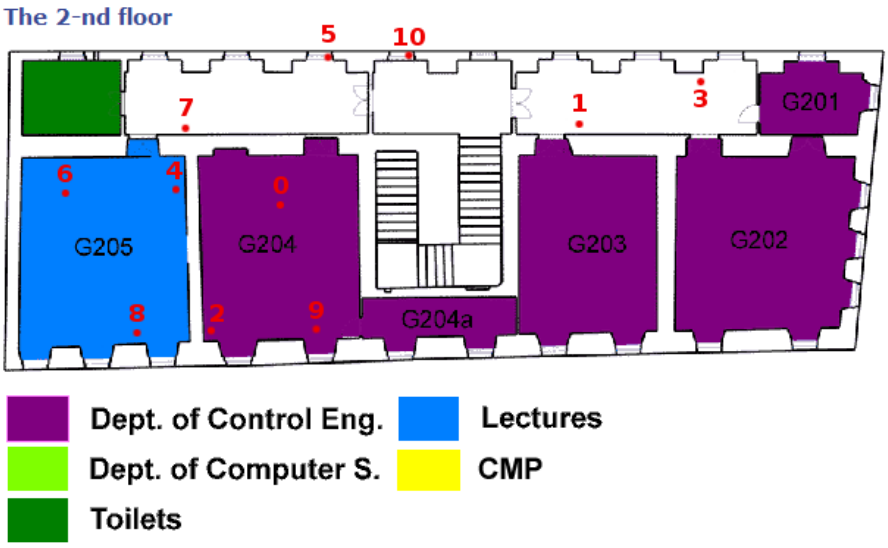
5.2 Instalace senzorové sítě

Jako uzavřený objekt, do kterého se senzorová síť nainstalovala, sloužilo druhé patro budovy G elektrotechnické fakulty ČVUT na Karlově náměstí v Praze. Senzorová síť byla sestavena z 11 zařízení 2 platforem. TelosB od Firmy CrossBow a TmoteSky od firmy MoteIV. Hardwarově jsou si tyto platformy zcela rovnocenné, proto jejich spolupráce nečinila žádné problémy.

5.2.1 Rozmístění senzorů

Celkem bylo do 5-ti místností rozmístěno 11 senzorů. Plánek rozmístění [12] je na obr. 5.1. Senzory byly rozmístěny rovnoměrně po celé monitorované ploše. Natočení a vzdálenost senzorů od země byly přizpůsobeny prostředí a tudíž nejsou stejné. V prostředí se taky nacházely překážky a to jak statické ve formě skříní, stolů a lavic, tak také pohyblivé,

tvořené zaměstnanci katedry.



Obrázek 5.1: Plánek rozmístění senzorů

Fotografická dokumentace poloh všech senzorů je na přiloženém CD.

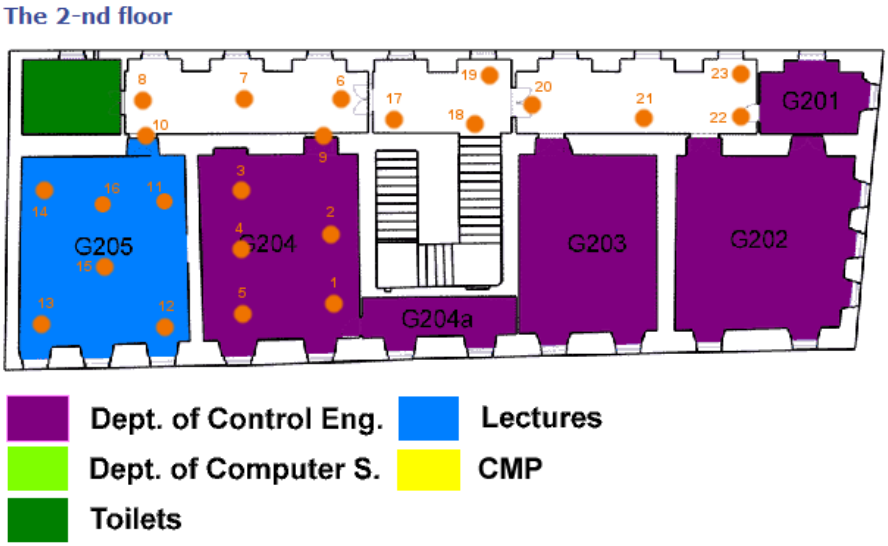
5.2.2 Rozmístění měřicích míst

Každý soubor dat (trénovací i testovací množina) obsahuje data z 23 míst, která jsou vyznačena na obr. 5.2.

5.3 Postup měření

Poté, co byly rozmístěny všechny senzory s nainstalovaným systémem Octopus podle obr. 5.1, postupně jsem procházel všechna měřicí místa podle obr. 5.2 s jedním senzorem ve funkci vysílače. Na každém měřicím místě jsem stisknul uživatelské tlačítko vysílače. Tím došlo k odeslání 20-ti paketů v časových prodlevách 200 milisekund. Během vysílání, které trvalo 4 sekundy, jsem náhodně natáčel anténu vysílače, aby vysílaná data přicházela z každého měřicího místa pod různými úhly. Po projití všech měřicích míst jsem data uložil

a tyto považuji za trénovací množinu. Poté jsem celý pokus opakoval a vznikla množina testovací. Data z trénovací a testovací se liší, protože při každém měření se anténa natáčela jiným způsobem. Také pohyb lidí způsobuje rozdílné hodnoty obou množin.



Obrázek 5.2: Vyznačení měřicích míst

5.4 Naměřená data

Nashromážděná data v podobě *.csv souborů je nutné předzpracovat a uložit do podoby, se kterou by se pohodlně pracovalo.

5.4.1 Předzpracování dat

Jelikož pro rozpoznávání polohy využívám program Matlab, je výhodné data předzpracovat a uložit do souboru *.mat. Naprogramoval jsem skript, který na vstupu načte *.csv soubory a na výstupu uloží tato data do souboru *.mat. Skript provádí filtraci nadbytečných *prázdných* paketů. O *prázdných* paketech je pojednáno v sekci 4.2.

5.4.2 Formát dat

Formát matice s naměřenými daty jsem zvolil tak, aby se s daty pracovalo co nejvíce pohodlně.

Číslo série	Číslo paketu	Data ze senzoru				
		0	1	2		n
1	1					
	2					
	p					
2						
3						
m						

Obrázek 5.3: Formát matice s naměřenými daty

Jak je patrné z obr. 5.3, jsou všechny pakety seřazeny vzestupně podle čísla série (*setNumber*) a také podle čísla paketu v sérii (*packetNumber*). Data ze všech (11-ti) senzorů poté tvoří sloupce. Sloupce jsou seřazeny podle identifikátoru každého senzoru.

Kapitola 6

Klasifikace

6.1 Třídy

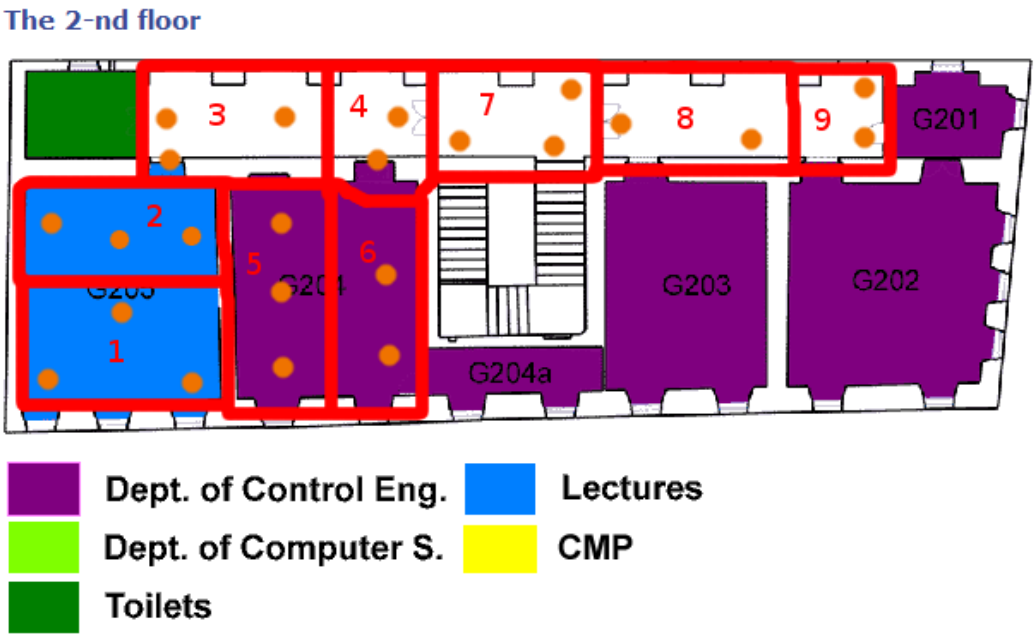
Množiny tříd, do kterých se mají klasifikovat data z testovací množiny jsem volil celkem 3. Jedná se o množiny *místo*, *sektor* a *místnost*. Zvolil jsem tyto 3 množiny, abych zjistil, jak se mění úspěšnost klasifikace, jednotlivá třída je tvořena skupinou měřicích míst. Čísla v závorkách představují počet tříd v množině.

- množina *místo* (23) je tvořena jednotlivými měřicími místy tak, jak je uvedeno na obr. 5.2. Co místo, to třída.
- množina *sektor* (9) - zde jsou měřicí místa sdružena tak, aby vytvořily sektory. Tyto sektory jsou vyobrazeny na obr. 6.1. Velikost každého sektoru je volena tak, aby každý z nich sdružoval 2-3 sousední měřicí místa.
- množina *místnost* (5) - zde jsou měřicí místa sdružena do místností.

6.2 Metoda k nejbližších sousedů

6.2.1 Princip

Úkolem této metody k nejbližších sousedů [1],[10] je klasifikovat, tj. přiřadit vektor $\boldsymbol{x}$, do třídy $y_i \in \mathcal{Y} = \{1, ..., c\}$, kde c je počet tříd.



Obrázek 6.1: Mapa rozložení sektorů

Ve fázi trénování se pouze vektory z trénovací množiny zařadí do tříd. Implicitně existuje tolik tříd, kolik je měřicích míst, ale v této práci je pod pojmem třída myšleno i shluknutí několika měřicích míst tak, aby místa vytvořila sektor či místnost.

Klasifikace pomocí metody k nejbližších sousedů je založena na měření a vyhodnocování vzdáleností mezi vektory v prostoru příznaků. Každý příznakový vektor je tvořen hodnotami RSSI naměřenými v rámci jednoho vyslaného paketu z vysílače. Každý tento vektor má tedy délku 11, podle počtu senzorů v celé síti. Dimenze prostoru příznaku je tedy také 11. Vzdálenost d je vždy počítána v nějaké metrice ρ. Volba metriky ovlivňuje úspěšnost klasifikace.

Spočítám tedy vždy vzdálenosti mezi vektorem $\boldsymbol{x}$ a všemi vektory z trénovací množiny. Parametr k v názvu této metody potom říká, že určujících je jen k nejmenších vzdáleností. Z těchto k vzdáleností poté určím nejčastěji se opakující třídu a ta budiž třídou klasifikovanou.

Nechť $\mathcal{T}_{XY} = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_l, y_l)\}$ je trénovací množina složena z trénovacích vektorů $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$ a příslušných tříd $y_i \in \mathcal{Y} = \{1, ..., c\}$. Nechť funkce $\rho(\boldsymbol{x}, \boldsymbol{y})$ značí použitou metriku a nechť $\mathbb{R}^n(\boldsymbol{x}) = \{\boldsymbol{x}' : \rho(\boldsymbol{x} - \boldsymbol{x}') \leq r^2\}$ je n-rozměrná koule centrovaná do bodu určeného vektorem $\boldsymbol{x}$, ve kterém leží právě k trénovacích vektorů $\boldsymbol{x}_i \in \mathcal{X}$, kde $i \in \{1, ..., l\}$, tj. $|\{\boldsymbol{x}_i : \boldsymbol{x}_i \in \mathbb{R}^n(\boldsymbol{x})\}| = k$. Klasifikační pravidlo pro k nejbližších sousedů $q : \mathcal{X} \rightarrow \mathcal{Y}$ je potom definováno jako

$$q(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} v(\boldsymbol{x}, y), \tag{6.1}$$

kde $v(\boldsymbol{x}, y)$ je počet vektorů z trénovací množiny $\boldsymbol{x}_i$ s třídami $y_i = y$, které leží uvnitř koule $\boldsymbol{x}_i \in \mathbb{R}^n(\boldsymbol{x})$, tj. $v(\boldsymbol{x}, y) = |\{\boldsymbol{x}_i : \boldsymbol{x}_i \in \mathbb{R}^n, (\boldsymbol{x}_i, y) \in \mathcal{T}_{XY}\}|$.

6.2.2 Metriky

Jestliže $\boldsymbol{a}$, $\boldsymbol{b}$ a $\boldsymbol{c}$ jsou vektory, pak metrikou je každá funkce ρ, která má následující vlastnosti

1. nezápornost: $\rho(\boldsymbol{a}, \boldsymbol{b}) \geq 0$,
2. reflexivita: $\rho(\boldsymbol{a}, \boldsymbol{b}) = 0 \Leftrightarrow \boldsymbol{a} = \boldsymbol{b}$,
3. symetrie: $\rho(\boldsymbol{a}, \boldsymbol{b}) = \rho(\boldsymbol{b}, \boldsymbol{a})$,
4. trojúhelníková nerovnost: $\rho(\boldsymbol{a}, \boldsymbol{b}) + \rho(\boldsymbol{b}, \boldsymbol{c}) \geq \rho(\boldsymbol{a}, \boldsymbol{c})$.

6.2.2.1 Euklidovská metrika

Jedná se o důsledek Pythagorovy věta. Vzdálenost mezi dvěma příznakovými vektory $\rho_E(\boldsymbol{x}_1, \boldsymbol{x}_2)$ je počítána podle vztahu

$$\rho_E(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{\sum_i (\boldsymbol{x}_1(i) - \boldsymbol{x}_2(i))^2}. \tag{6.2}$$

6.2.2.2 Manhattanská metrika

Touto metrikou se počítají vzdálenosti mezi městskými bloky ve velkých městech. Odtud také plyne název této metriky. Vzdálenost mezi dvěma příznakovými vektory $\rho_M(\boldsymbol{x}_1, \boldsymbol{x}_2)$

je počítána podle vztahu

$$\rho_M(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_i |(\boldsymbol{x}_1(i) - \boldsymbol{x}_2(i))|. \tag{6.3}$$

6.2.2.3 Manhattanská metrika - modifikace

Jedná se o Manhattanskou metriku, pouze nezapočítáme největší rozdíl. Vzdálenost mezi dvěma příznakovými vektory $\rho_M^*(\boldsymbol{x}_1, \boldsymbol{x}_2)$ je počítána podle vztahu

$$\rho_M^*(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_i |(\boldsymbol{x}_1(i) - \boldsymbol{x}_2(i))| - \max_i(|(\boldsymbol{x}_1(i) - \boldsymbol{x}_2(i)|). \tag{6.4}$$

Tuto metriku jsem ještě upravil tak, že se vynechávaly největší rozdíly dva. Později se ukázalo, že takto upravená metrika (odečítání dvou největších rozdílů) funguje nejlépe ze všech použitých.

6.2.2.4 Nekonečná metrika

Vzdálenost mezi dvěma příznakovými vektory $\rho_\infty(\boldsymbol{x}_1, \boldsymbol{x}_2)$ je počítána podle vztahu

$$\rho_\infty(\boldsymbol{x}_1, \boldsymbol{x}_2) = \max_i(|(\boldsymbol{x}_1(i) - \boldsymbol{x}_2(i)|). \tag{6.5}$$

6.3 Support vector machines

Klasifikační metoda Support Vector Machines (SVM) [1], [10] a [11] byla původně vyvinuta pro binární klasifikaci, tj. klasifikaci do dvou tříd. Tato metoda se dá rozšířit na vícetřídovou (multiclass) například tak, že se kombinují jednotlivé binární klasifikátory a jejich dílčí výsledky se dále matematicky zpracují. V této práci jsem se zabýval jednou vícetřídovou metodou SVM, které dekomponuje daný problém pro více klasifikátorů binárních. Jedná se o metodu *one-against-all* (jeden vůči všem). O této metodě je pojednáno v dalším textu.

6.3.1 Support vector klasifikátor

Binární support vector klasifikátor využívá diskriminační funkci $f : \mathcal{X} \subseteq \mathbb{R}^n \to \mathbb{R}$ v následujícím tvaru

$$f(\boldsymbol{x}) = \langle \boldsymbol{\alpha} \cdot \boldsymbol{k}_s(\boldsymbol{x}) \rangle + b. \tag{6.6}$$

Kde $\langle \cdot \rangle$ značí operaci skalárního součinu a $\boldsymbol{k}_s(\boldsymbol{x}) = [k(\boldsymbol{x}, \boldsymbol{s}_1), ..., k(\boldsymbol{x}, \boldsymbol{s}_d)]^T$ je vektor vypočtených jádrových funkcí centrovaných v podpůrných (support) vektorech $\mathcal{S} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_d\}, \boldsymbol{s}_i \in \mathbb{R}^n$, které jsou obvykle podmnožinou trénovacích dat. Vektor $\boldsymbol{\alpha}$ je váhovací vektor a číslo $b \in \mathbb{R}$ je posunutí (bias). Binární klasifikační pravidlo $q : \mathcal{X} \to \mathcal{Y} = \{1, 2\}$ je definováno jako

$$q(\boldsymbol{x}) = \begin{cases} 1 & \text{pro} \quad f(\boldsymbol{x}) \geq 0, \\ 2 & \text{pro} \quad f(\boldsymbol{x}) < 0. \end{cases} \tag{6.7}$$

Pro vícetřídové (multiclass) zobecnění SVM klasifikátoru zavedeme množinu diskriminačních funkcí $f_y : \mathcal{X} \subseteq \mathbb{R}^n \to \mathbb{R}, y \in \mathcal{Y} = \{1, 2, ..., c\}$ definovaných jako

$$f_y(\boldsymbol{x}) = \langle \boldsymbol{\alpha}_y \cdot \boldsymbol{k}_s(\boldsymbol{x}) \rangle + b_y, \qquad y \in \mathcal{Y}. \tag{6.8}$$

Vícetřídové klasifikační pravidlo $q : \mathcal{X} \to \mathcal{Y} = \{1, 2, ..., c\}$ je definováno jako

$$q(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} f_y(\boldsymbol{x}). \tag{6.9}$$

6.3.2 One-Against-All dekompozice pro SVM

Vstupní množina $\mathcal{T}_{XY} = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_l, y_l)\}$ složená z trénovacích dat $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$ a příslušných stavů $y_i \in \mathcal{Y} = \{1, ..., c\}$. Cílem je natrénovat vícetřídové pravidlo $q : \mathcal{X} \to \mathcal{Y}$ definováno podle (6.8). Tento problém může být vyřešen pomocí dekompozice One-Against-All (OAA). OAA dekompozice převádí problém vícetřídové klasifikace na sérii c binárních podúkolů. Tyto podúkoly pak mohou být vyřešeny pomocí binárního klasifikátoru SVM. Nechť $\mathcal{T}_{XY}^y = \{(\boldsymbol{x}_1, y_1^{'}), ..., (\boldsymbol{x}_l, y_l^{'})\}$ obsahují modifikované třídy definovány jako

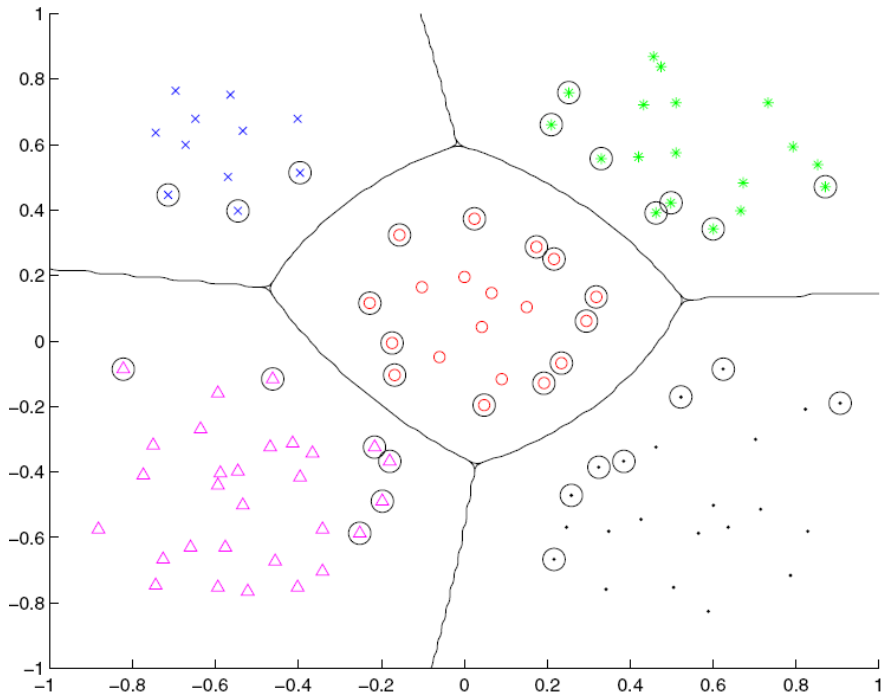
$$y_i^{'} = \begin{cases} 1 & \text{pro} \quad y = y_i, \\ 2 & \text{pro} \quad y \neq y_i. \end{cases}$$

Diskriminační funkce

$$f_y(\boldsymbol{x}) = \langle \boldsymbol{\alpha}_y \cdot \boldsymbol{k}_s(\boldsymbol{x}) \rangle + b_y, \qquad y \in \mathcal{Y},$$

jsou natrénovány binárním SVM algoritmem z množiny $\mathcal{T}_{XY}^y, y \in \mathcal{Y}$. To znamená, že vznikne c takových diskriminačních funkcí. Kde c je počet tříd. Klasifikace probíhá podle (6.9) tak, že se neznámý klasifikovaný vektor dosadí do všech diskriminačních funkcí. Výsledkem klasifikace je takové y, pro které funkce f_y vrátí největší funkční hodnotu.

Vizualizace klasifikace do více tříd pomocí SVM s použitou dekompozicí OAA je uveden na obr. 6.2.



Obrázek 6.2: Ukázka vícetřídového SVM klasifikátoru sestaveného pomocí dekompozice OAA

Kapitola 7

Implementace klasifikační části

Všechny klasifikační skripty jsou naimplementovány v prostředí Matlab. Všechny jsou v podobě funkcí uloženy v souborech *.m. Ke každé implementované metodě existuje hlavní spouštěcí skript. Názvy hlavních skriptů jsou ve formátu main_*metoda*.m.

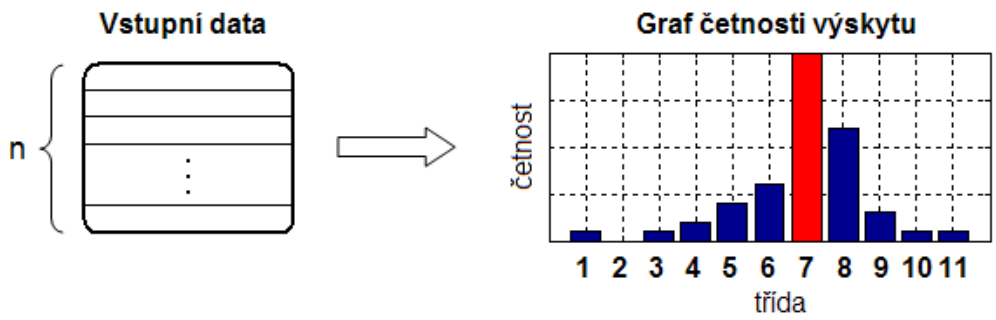
7.1 Metoda k nejbližších sousedů

Vyzkoušel jsem 3 různá zpracování vstupních dat, tj. testovací množiny. Využil jsem faktu, že v jednom měřicím místě je k dispozici 20 vzorků (bylo vysláno 20 paketů) a data naměřená v jednom místě jsem matematicky zpracoval pomocí aritmetického průměru a pomocí mediánu. Otestoval jsem tak vliv těchto zpracování na úspěšnost klasifikace. Ukázalo se, že zpracování vstupních dat úspěšnost klasifikace poměrně výrazně ovlivňuje.

Pro metodu k nejbližších sousedů jsem napsal skripty vlastní a také jsem použil toolbox SPRTool [14], který disponuje funkcí pro klasifikaci touto metodou.

7.1.1 Bez zpracování vstupu

Vstupní data nebyla nijak zpracována, tj. do klasifikátoru vstupovala všechna data z testovací množiny nijak neupravena. Princip tohoto zpracování ilustruje obr. 7.1.



Obrázek 7.1: Ilustrace zacházení se vstupními daty (bez zpracování)

Vstupní data na obr. 7.1 představují n vstupních vektorů z testovací množiny pořízených na jednom měřicím místě m_i. V našem případě je $n = 20$. Ke každému vzorku je přiřazena třída, tj. každý vzorek je klasifikován metodou k nejbližších sousedů. Z těchto n vzorků je pro měřicí místo m_i vypočtena četnost výskytu. Třída s nejvyšší četností (na obrázku červeně) je pak třídou klasifikovanou pro místo m_i.

Tato metoda je naimplementována v souboru `main_kNN2.m`

7.1.2 Zpracování vstupu pomocí aritmetického průměru

Nechť vektory $\boldsymbol{x}_{i,j} \in \mathcal{X} \subseteq \mathbb{R}^n$ tvoří testovací množinu dat. Index $i \in \mathcal{V} = \{1, 2, ..., v\}$ přestavuje číslo série a index $j \in \mathcal{P} = \{1, 2, ..., p\}$ představuje číslo paketu daného vzorku. Potom předzpracováním vznikne nová množina $\mathcal{X}' \subseteq \mathbb{R}^n$, která je tvořena vektory $\boldsymbol{x}_i'$, které jsou definovány jako

$$\boldsymbol{x}_i' = \frac{1}{p} \sum_{j=1}^{p} \boldsymbol{x}_{ij}, \tag{7.1}$$

Nutno dodat, že k-tá složka vektoru $\boldsymbol{x}_i'$ je počítána jako střední hodnota k-tých složek vektorů $\boldsymbol{x}_{ij}$. Jako testovací množina se poté použije množina $\mathcal{X}'$, která bude obsahovat p krát méně vzorků než původní testovací množina $\mathcal{X}$.

Tato metoda je naimplementována v souboru `main_kNN_mean.m`

7.1.3 Zpracování vstupu pomocí mediánu

Nechť vektory $\boldsymbol{x}_{i,j} \in \mathcal{X} \subseteq \mathbb{R}^n$ tvoří testovací množinu dat. Index $i \in \mathcal{V} = \{1, 2, ..., v\}$ přestavuje číslo série a index $j \in \mathcal{P} = \{1, 2, ..., p\}$ představuje číslo paketu daného vzorku. Potom předzpracováním vznikne nová množina $\mathcal{X}' \subseteq \mathbb{R}^n$, která je tvořena vektory $\boldsymbol{x}_i^{'}$, které jsou definovány jako

$$\boldsymbol{x}_i^{'} = \underset{j \in \mathcal{P}}{\text{median}}\, \boldsymbol{x}_{ij}, \tag{7.2}$$

Medián je stejně jako v případě střední hodnoty počítán po složkách. Jako testovací množina se poté použije množina $\mathcal{X}'$, která rovněž jako v případě zpracování pomocí střední hodnoty bude obsahovat p krát méně vzorků než původní testovací množina $\mathcal{X}$. Zpracování vstupních dat pomocí mediánu vykazovalo nejlepší výsledky.

Tato metoda je naimplementována v souboru `main_kNN_median.m`

7.1.4 Použití toolboxu STPRTool

Pomocí toolboxu STPRTool jsem vyzkoušel klasifikaci do všech tří množin tříd (*místo*, *sektor* a *místnost*). Použil jsem stejné předzpracování vstupu jako v předchozích metodách, tj. bez předzpracování, střední hodnotu i medián.

Tato metoda je naimplementována v souborech `main_stpr_kNN.m` (střední hodnota, medián), resp. `main_stpr_kNN2.m` (bez zpracování)

7.2 Metoda Support Vector Machines

7.2.1 Dekompozice One-Against-All

Pomocí dekompozice One-Against-All jsem klasifikoval pomocí toolboxu SPRTool vstupní trénovací data. Opět klasifikace probíhala do množin tříd *místo*, *sektor* a *místnost*. Při klasifikaci jsem používal SMO optimalizátor (Sequential Minimal Optimizer) a jádro (kernel) RBF. Podrobnější informace se lze dočíst v [11]. U tohoto jádra je nastavitelný argument

γ a konstanta C. Konstantu C jsem volil $C = 10$, protože její změna příliš neovlivňovala úspěšnost klasifikace. Měnil jsem proto nastavení jen parametru γ. Vyzkoušel jsem opět klasifikaci do všech tří množin tříd (*místo*, *sektor* a *místnost*). Předzpracování vstupu jsem použil stejné jako v předchozích metodách, tj. bez předzpracování, střední hodnotu a medián.

Tato metoda je naimplementována v souborech `main_svm.m` (střední hodnota, medián), resp. `main_svm2.m` (bez zpracování).

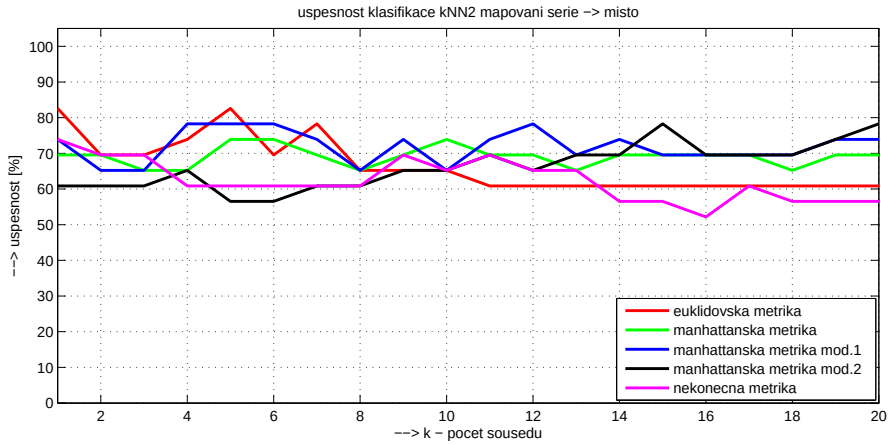
Kapitola 8

Výsledky

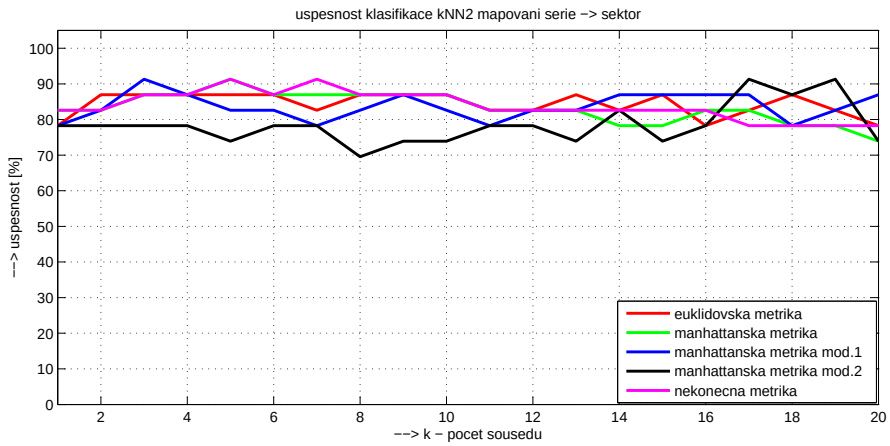
8.1 Metoda k nejbližších sousedů

Úspěšnost klasifikace je počítána jako poměr správně klasifikovaných vstupních dat ku všem vstupním datům. Na ose y je tedy vynesená úspěšnost v procentech, na ose x je parametr k, což je počet nejbližších sousedů. Jak již bylo zmíněno dříve, data jsem klasifikoval do tří množin tříd (*místo*, *sektor* a *místnost*). Přitom jsem vyzkoušel trojí zpracování vstupních dat (bez zpracování, střední hodnota, medián). Rovněž je klasifikace provedena (také pro všechna tři zpracování vstupních dat) pomocí toolboxu STPRTool.

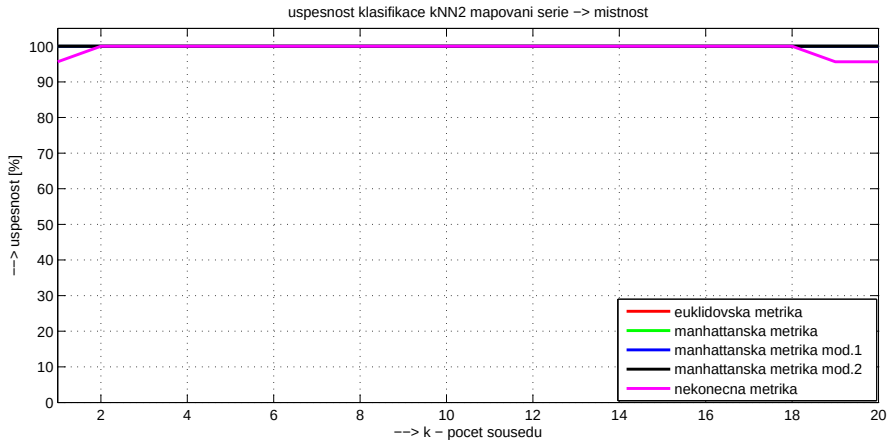
- Vstupní data bez zpracování: viz 8.1, 8.2 a 8.3
- Vstupní data zpracována střední hodnotou: 8.4, 8.5 a 8.6
- Vstupní data zpracována mediánem: 8.7, 8.8 a 8.9
- Klasifikace pomocí toolboxu STPRTool: 8.10, 8.11, 8.12



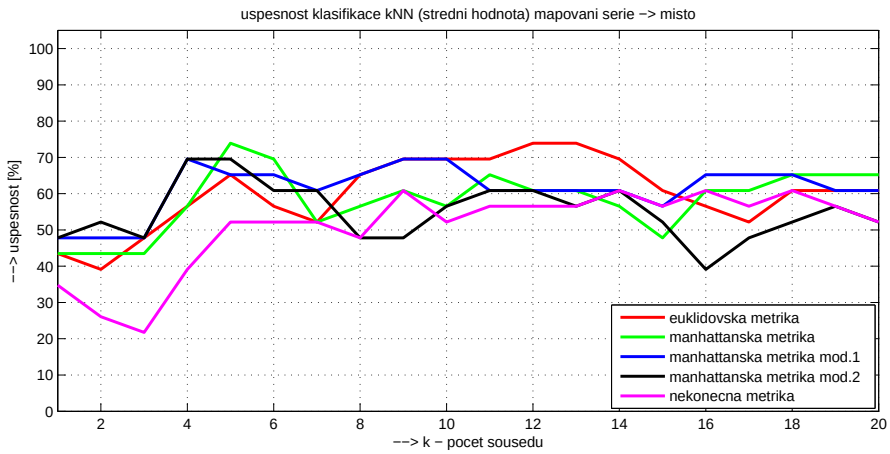
Obrázek 8.1: Úspěšnost klasifikace na *místo*. k nejbližších sousedů (vstupní data bez zpracování)



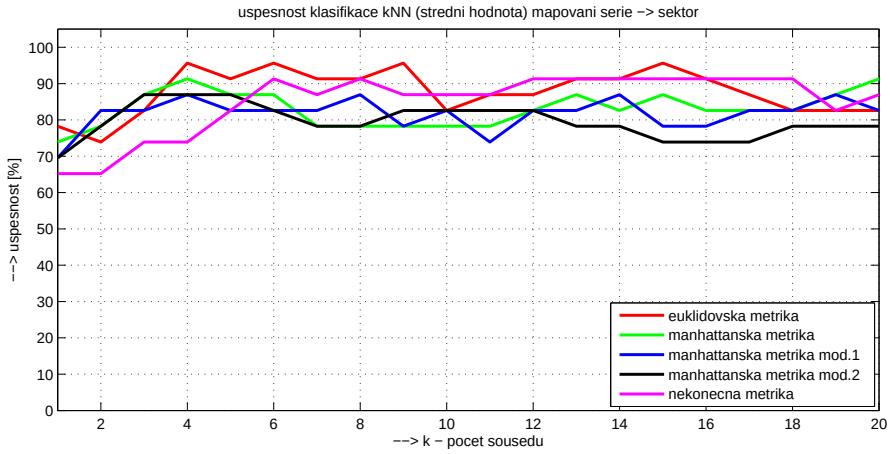
Obrázek 8.2: Úspěšnost klasifikace na *sektor*. k nejbližších sousedů (vstupní data bez zpracování)



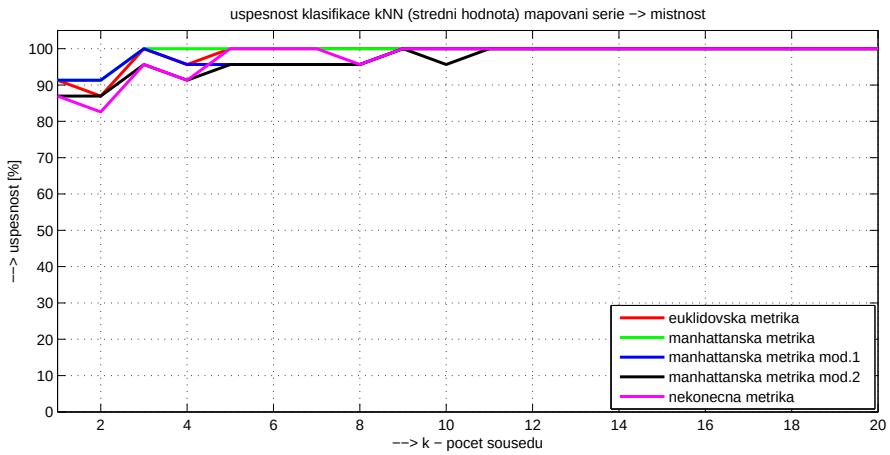
Obrázek 8.3: Úspěšnost klasifikace na *místnost*. k nejbližších sousedů (vstupní data bez zpracování)



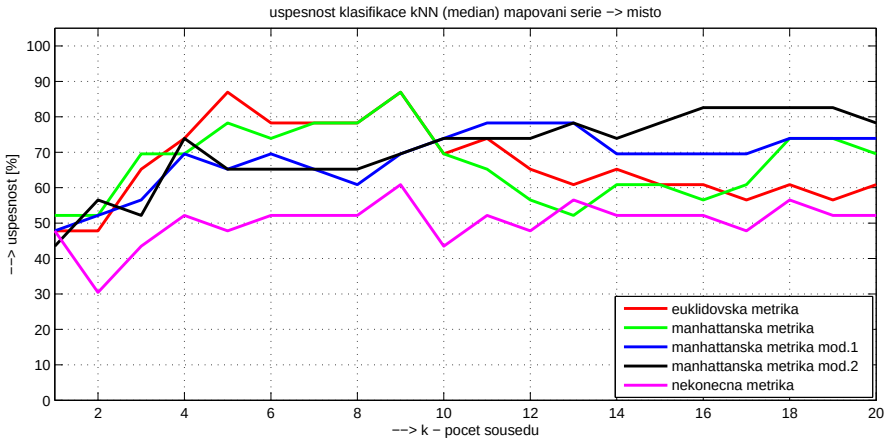
Obrázek 8.4: Úspěšnost klasifikace na *místo*. k nejbližších sousedů (střední hodnota vstupních dat)



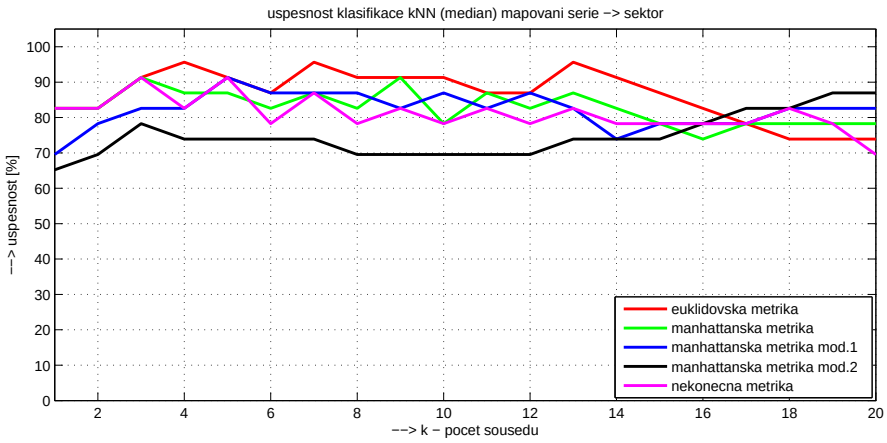
Obrázek 8.5: Úspěšnost klasifikace na *sektor*. k nejbližších sousedů (střední hodnota vstupních dat)



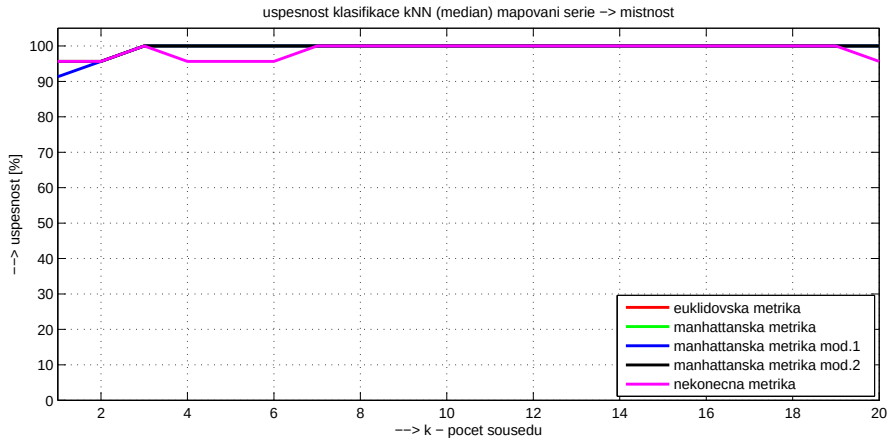
Obrázek 8.6: Úspěšnost klasifikace na *místnost*. k nejbližších sousedů (střední hodnota vstupních dat)



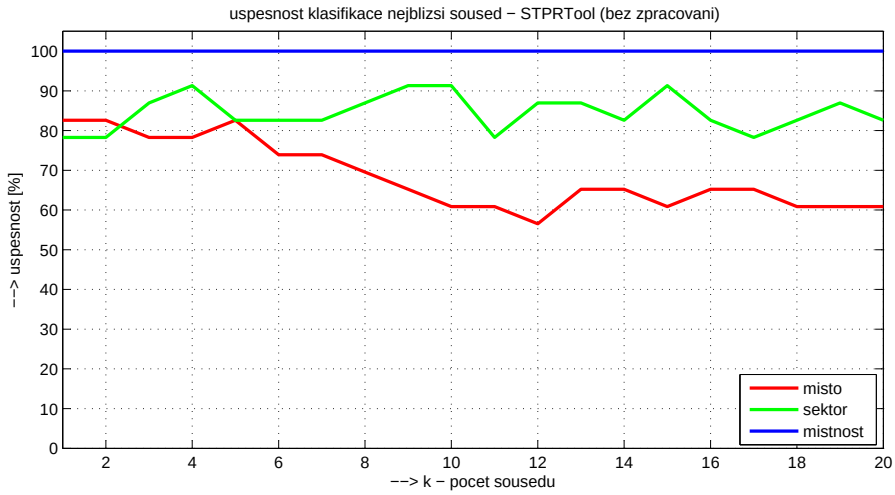
Obrázek 8.7: Úspěšnost klasifikace na *místo*. k nejbližších sousedů (medián vstupních dat)



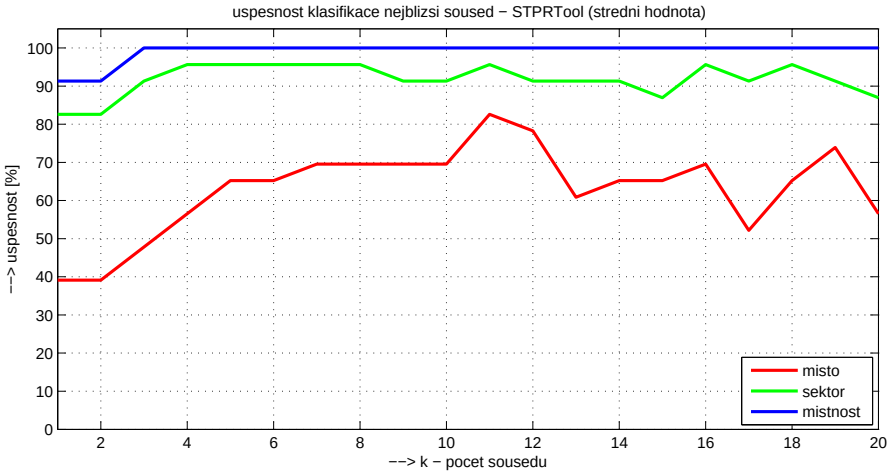
Obrázek 8.8: Úspěšnost klasifikace na *sektor*. k nejbližších sousedů (medián vstupních dat)



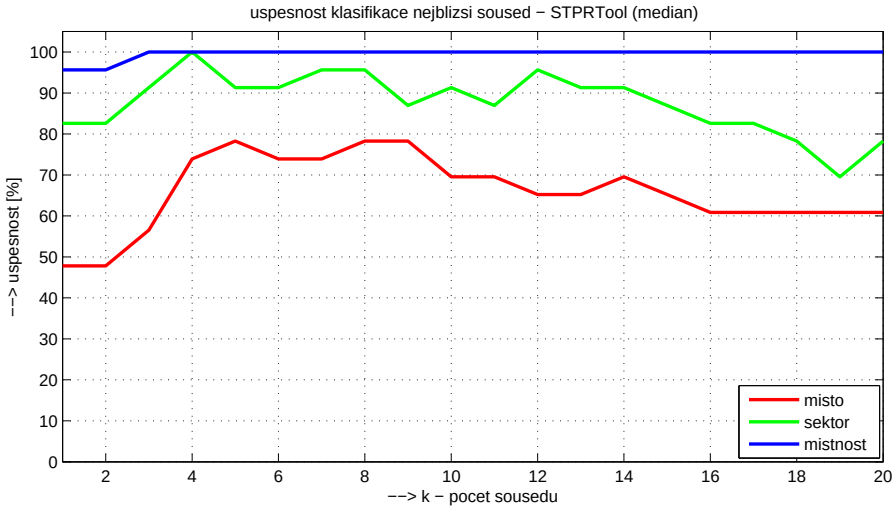
Obrázek 8.9: Úspěšnost klasifikace na *místnost*. k nejbližších sousedů (medián vstupních dat)



Obrázek 8.10: Úspěšnost klasifikace metodou k nejbližších sousedů pomocí STPRTool (vstupní data bez zpracování)



Obrázek 8.11: Úspěšnost klasifikace metodou k nejbližších sousedů pomocí STPRTool (střední hodnota vstupních dat)



Obrázek 8.12: Úspěšnost klasifikace metodou k nejbližších sousedů pomocí STPRTool (medián vstupních dat)

Z výše uvedených grafů vyplývá podle očekávání, že úspěšnost klasifikace se zvýší, jestliže zmenšíme počet klasifikovaných tříd. Nejhorší úspěšnost má klasifikace na *místo*. Naopak nejvyšší úspěšnost je dosažena při klasifikaci na *místnost*.

Použití metody k nejbližších sousedů při klasifikaci na *místo* není moc použitelné. Úspěšnost klasifikace nepřesáhne při všech použitých metodách hodnotu 86%. Při klasifikaci na *sektor* jsem dosáhl maximální úspěšnosti 96%, a při klasifikaci na *místnost* lze dosáhnout úspěšnosti 100% se všemi použitými metrikami.

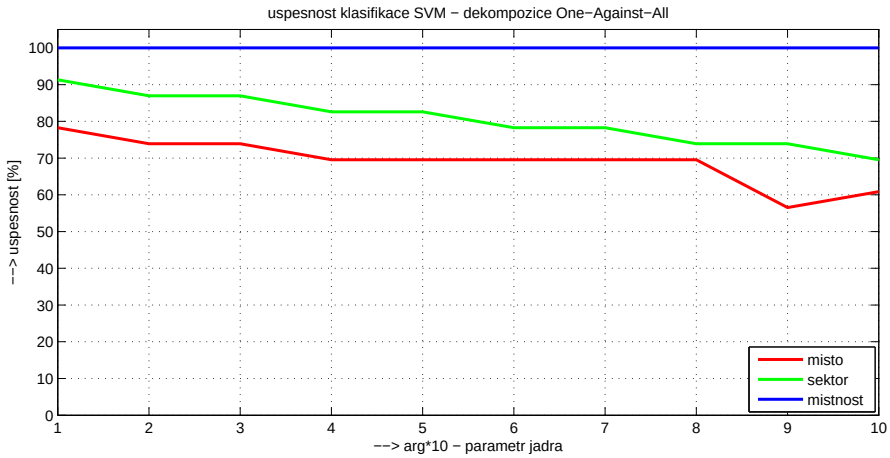
Z uvedených grafů je také vidět, že s použitím toolboxu STPRTool je úspěšnost velmi podobná s mými naprogramovanými skripty. Navíc průběhy úspěšnosti s euklidovskou metrikou jsou podobné těm, které byly vypočteny pomocí toolboxu STPRTool, který pro výpočet euklidovskou metriku používá.

Vliv použité metriky je rovněž z grafů patrný. Ukázalo se, že pro tuto úlohu je velmi dobře použitelná euklidovská metrika.

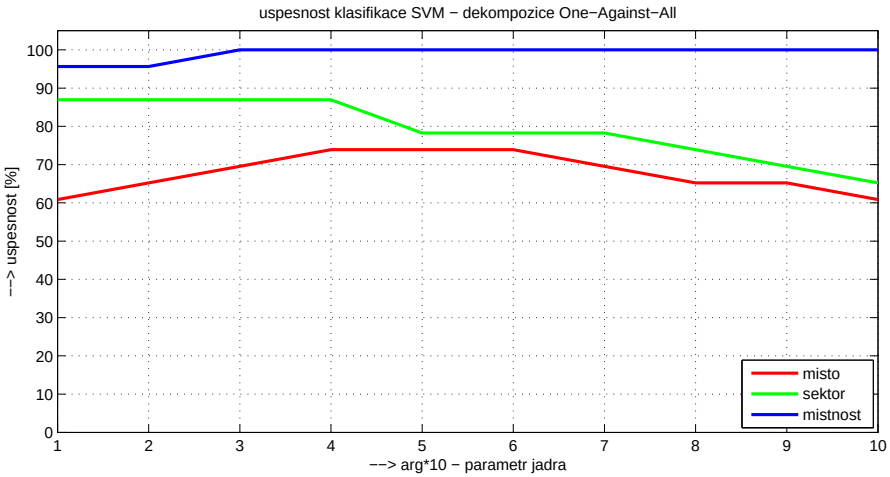
Co se týká zpracování vstupních dat, tak jako nejlepší se jeví zpracování vstupních dat pomocí mediánu. Při klasifikaci na *místo* je rovněž velmi dobře použitelné, když vstupní data nijak nepředzpracovávám.

8.2 Metoda Support Vector Machines

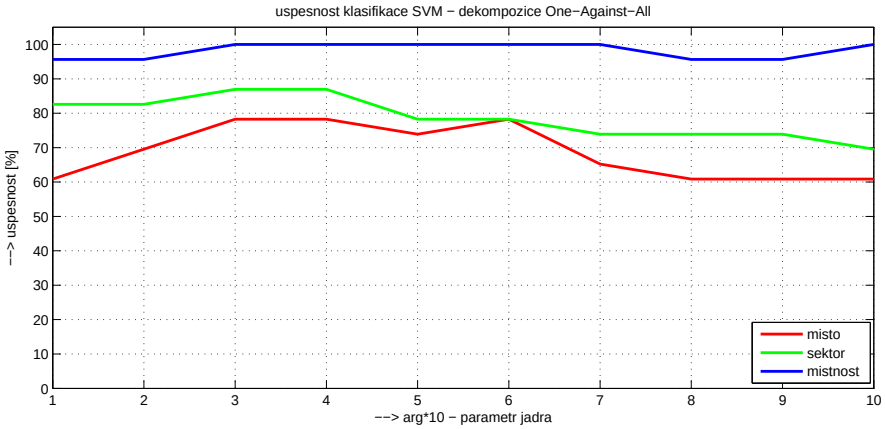
Závislosti jsou vynášeny v procentech. Na ose x jsou různé hodnoty parametru γ jádra RBF. Na obr. 8.13, obr. 8.14 a obr. 8.15 jsou uvedeny výsledky klasifikace pro nastavení konstanty $C = 10$.



Obrázek 8.13: Úspěšnost klasifikace metodou SVM (OAA), $C = 10$ (vstupní data bez zpracování)



Obrázek 8.14: Úspěšnost klasifikace metodou SVM (OAA), $C = 10$ (střední hodnota vstupních dat)



Obrázek 8.15: Úspěšnost klasifikace metodou SVM (OAA), $C = 10$ (medián vstupních dat)

Z uvedených grafů na obr. 8.13, obr. 8.14 a obr. 8.15 je patrné, že metody SVM a k nejbližších sousedů mají srovnatelnou úspěšnost.

Kapitola 9

Závěr

Cílem této práce bylo naimlementovat a vyzkoušet použitelnost metod lokalizace rádiového vysílače bezdrátovou senzorovou sítí.

Byly použity senzory osazené rádiovým čipem CC2420, který dokáže změřit sílu přijatého signálu. Ve druhé kapitole jsem popsal volně šiřitelnou aplikaci Octopus Dashboard určenou k monitorování a také k řízení senzorové sítě. V kapitole 3 jsem tuto aplikaci upravil tak, aby pomocí senzoru CC2420 měřila sílu přijatého signálu a tuto hodnotu poslala do PC. Takto upravenou aplikaci jsem nahrál do senzorů a sestavil z nich senzorovou síť, kterou jsem nainstaloval do uzavřeného objetku se statickými i dynamickými překážkami. Popis experimentu je uveden v katipole 5. V kapitole 6 se podrobněji zabývám metodami klasifikace. Zabýval jsme se metodou k nejbližších sousedů a metodou Support Vector Machines. Popis implementace těchto metod je předmětem kapitoly 7. Výsledky klasifikace jsou uvedeny v kapitole 8.

Měření ukázalo, že jako nejvíce použitelná metoda klasifikace je metoda k nejbližších sousedů s euklidovskou metrikou. Tato metrika má v průměru nejlepší výsledky. Úspěšnost klasifikace 100% byla dosažena, pouze pokud byly rozpoznávány jednotlivé místosti a to s použitím všech metrik.

Jako další rozšíření této práce by mohlo být vyzkoušení měnit sílu signálu vysílaného vysílačem. Jelikož v této práci byla použita pouze jedna (maximální) hodnota síly signálu, nebyl tak zachycen vliv síly signálu na úspěšnost klasifikace.

Literatura

[1] R. O. Duda, P. E. Hart, D. G. Stork. *Pattern Clasification* John Wiley & Sons, 2nd edition, 2001. ISBN 0-741-05669-3.

[2] L. Jirák *Sledování polohy mobilního robota senzorovou sítí* Bakalářská práce, ČVUT, Fakulta elektrotechnická, Praha, 2008.

[3] P. Levis *TinyOS Programming* Revision 1.3 (Oct 27 2008) `http://csl.stanford.edu`.

[4] Werner Weber, Jan M. Rabaey and Emile Aarts *TinyOS: An Operating System for Sensor Networks* [115-148] Springer Berlin Heidelberg, 2005. ISBN 978-3-540-23867-6 (Kniha) 978-3-540-27139-0 (Online) `http://www.springerlink.com/content/l82402t220708233/`, `http://www.tinyos.net/`.

[5] TinyOS Documentation Wiki. `http://docs.tinyos.net/index.php/` duben 2009, cit. 8.6.2009.

[6] Moteiv Corporation Telos (Rev B). PRELIMINARY Datasheet, 2004.

[7] Crossbow Technology. `http://www.xbow.com/` 2009, cit. 8.6.2009.

[8] CC2420. Chipcon products from Texas Instruments, Datasheet, 2006.

[9] Octopus: A Dashboard for Sensor Networks Visual Control `http://www.csi.ucd.ie/content/octopus-dashboard-sensor-networks-visual-control` 2009, cit. 8.6.2009.

[10] V. Franc, V. Hlaváč. *Statistical Pattern Recognition Toolbox for Matlab (User's guide)* Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, Praha, 2004. ISSN 1213-2365.

[11] C.W. Hsu and C.J. Lin. *A comparison of methods for multiclass support vector machines* IEEE Transactions on Neural Networks, 13(2), March 2002.

[12] Plánek budovy G. `http://cyber.felk.cvut.cz/contact/g.phtml` cit. 31.5.2009

[13] N. Burri, R. Schuler, and R. Wattenhofer. *YETI: A TinyOS Plug-in for Eclipse* ACM Workshop on Real-World Wireless Sensor Networks (REALWSN06), June 2006 `http://tos-ide.ethz.ch/wiki/index.php` aktualizace 2.duben, 2009

[14] Statistical Pattern Recognition Toolbox for Matlab. `http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html` aktualizace 15.srpen, 2008

[15] CC2431 Location Engine. Chipcon products from Texas Instruments, Application Note AN042, (Rev. 1.0), SWRA095.

[16] David Moore, John Leonard, Daniela Rus, Seth Teller. Robust Distributed Network Localization with Noisy Range Measurements `http://rvsn.csail.mit.edu/netloc/sensys04.pdf` cit. 8.6.2009.

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy.

- Senzory\ - *implementace senzorové části*

  LocMote\ - *program pro vysílač*

  Octopus\ - *program pro senzorovou síť*

- Klasifikace\ - *implementace rozpoznávací části*

  data\ - *naměřená data*

  import_dat\ - *skripty pro import dat*

  klasikikace_kNN\ - *skripty pro klasifikaci pomocí k nejbližších sousedů*

  klasifikace_svm\ - *skripty pro klasifikaci pomocí SVM*

- Dokumentace\

  bp_pdf\ - *tato bakalářská práce ve formátu pdf*

  bp_latex\ - *zdrojové kódy pro program LaTeX*

  clanky\ - *články o lokalizaci*