VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ZVÝŠENÍ PŘESNOSTI A ROBUSTNOSTI BEZDRÁTOVÝCH LOKALIZAČNÍCH TECHNIK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

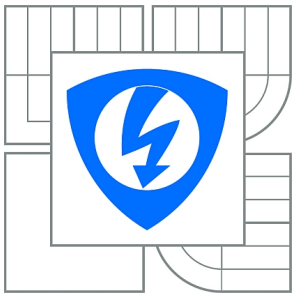
AUTOR PRÁCE
AUTHOR

Bc. DANIEL TOMÁNEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ZVÝŠENÍ PŘESNOSTI A ROBUSTNOSTI BEZDRÁTOVÝCH LOKALIZAČNÍCH TECHNIK

OPTIMIZATION OF WIRELESS LOCALIZATION TECHNIGUES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

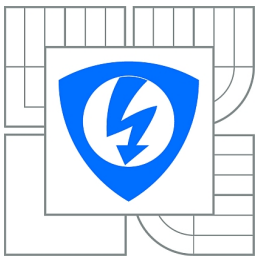
AUTOR PRÁCE — Bc. DANIEL TOMÁNEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. MILAN ŠIMEK, Ph.D.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Daniel Tománek	***ID:***	125673
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Zvýšení přesnosti a robustnosti bezdrátových lokalizačních technik

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je optimalizace algoritmů pro výpočet souřadnic v lokalizačních systémech. Student se ve své práci zaměří na problematiku samotného výpočtu pozice, které je v současném stádiu programu implementována pomocí Levenberg-Marquardt algoritmu. Student si tento algoritmus detailně nastuduje a naprogramuje serverou aplikaci tak, aby dokázal snížit chybu výpočtu při uvažování zarušeného rádiového prostředí, tj. při chybném výpočtu měření vzdálenosti mezi bezdrátovými jednotkami. Cílem diplomové práce je navrhnout nastavení algorimtu tak, aby byl dostatečně odolný při chybném měření vzdálenosti k jednotlivým kotvám.

DOPORUČENÁ LITERATURA:

[1] Stojmenovic I., Handbook of Sensor Networks, Wiley,ISBN:13 978-0-471-68472-5, 2005.
[2] FARAHANI, Shahin. Zigbee Wireless Networks and Transceivers. [s.l.] : Elsevier, 2008. 329 s. ISBN 978-0-

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Milan Šimek, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Předmětem práce je problematika zaměřování bezdrátových jednotek v lokalizačních systémech řešených senzorickými sítěmi. Práce popisuje princip trilaterace a Levenberg-Marquardtův algoritmus. Hlavní částí práce jsou simulace chování lokalizačních systémů a možnosti optimalizace lokalizace.

KLÍČOVÁ SLOVA

okalizační systémy, simulace, Levenberg-Marquardtův algoritmus, MPFIT, trilaterace

ABSTRACT

The main theme of this thesis are problems of locating of wireless units in localization systems represented by wireless sensor networks. The thesis describes principle of trilateration and Levenberg-Marquardt algorithm. The main theme of this thesis is simulation of behavior of localisation systems and possible optimization of localization.

KEYWORDS

Localisation system, simulation, Levenberg-Marquardt algorithm, MPFIT, trilateration

TOMÁNEK, Daniel *Zvýšení přesnosti a robustnosti bezdrátových lokalizačních technik*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 51 s. Vedoucí práce byl Ing. Milan Šimek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Zvýšení přesnosti a robustnosti bezdrátových lokalizačních technik“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Milanu Šimkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Také bych rád poděkoval svým rodičům za jejich bezmeznou podporu při studiu a Kristýně Kutišové a Janu Režnému za morální podporu při tvorbě této práce.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
(podpis autora)

OBSAH

SEZNAM TABULEK

ÚVOD

V dnešní době rychlého rozvoje bezdrátových senzorových sítí je často jedním z požadavků na jejich vlastnosti schopnost lokalizace jednotek v prostoru. Lokalizační systémy vzniklé z bezdrátových senzorových systémů mohou mít v budoucnosti i ve dnešní době mnoho zajímavých využití, ale pro jejich rychlý rozvoj je žádoucí, aby tyto systémy byly schopny provádět lokalizaci jednotek rychle a přesně.

Cílem této Diplomové práce je ozřejmit, jak probíhá měření vzdálenosti v lokalizačním systému na principu senzorové sítě a jak je řešeno zaměření bezdrátové jednotky v prostoru pomocí trilaterace. Dále popisuje princip Levenberg-Markvardtova algoritmu, který zajišťuje rychlé hledání řešení soustavy rovnic, díky kterým lze určit souřadnice jednotky v prostoru. Práce též představuje možnosti knihovny MPFIT, která Levenberg-Markvardtův algoritmus implementuje.

V další části jsou představeny simulace navržené pro zjištění či ověření chování knihovny MPFIT a popisuje možnosti optimalizace algoritmu odhalením rušení v prostředí a minimalizací jeho vlivu na přesnost výpočtu polohy.

V poslední části práce jsou shrnuty výsledky simulací a je navrženo řešení, které odstraní problém s chybným zaměřením jednotky vzniklý kvůli chybnému měření jedné ze základnových stanic senzorové sítě.

1 LOKALIZAČNÍ SYSTÉMY

Pro lokalizaci objektů existuje mnoho různých metod a každá metoda má své výhody a nevýhody. Cílem lokalizace je určit souřadnice objektu ve 2D nebo 3D prostoru, kdy jsou tyto souřadnice součástí kartézského souřadného systému a jsou reprezentovány jako vektor $[x, y]$ nebo $[x, y, z]$. Dle požadavků lze zvolit například lokalizaci v nadřazeném souřadném systému, což umožňuje například Global Positioning system (GPS) nebo zaměření pomocí buňkových telekomunikačních sítí nebo mohou být stanoveny lokální souřadnice, ve kterých probíhá zaměřování vztažené k pevně definovaným bodům v prostoru.

Výhodou lokalizace v lokálních souřadnicích je, že jej lze provádět pomocí bezdrátových senzorových sítí, které vynikají zejména nízkou energetickou náročností ve srovnání s ostatními technologiemi lokalizace (zejména GPS). Z toho plyne výhoda dlouhé životnosti bezdrátových jednotek, které jsou obvykle napájeny z baterií.[5]

1.1 Lokalizace v senzorových sítích

Pro lokalizaci jednotek ve Wireless sensor network (WSN) je potřebné určit souřadný systém a v tomto souřadném systému rozmístit bezdrátové jednotky („kotvy“), jejichž poloha je předem známá. Od těch poté mohou ostatní bezdrátové jednotky („tagy“) měřit vzdálenosti a dle změřených vzdáleností následně lokalizační systém určí polohu zaměřovaného tagu v prostoru.[5]

1.1.1 Možnosti určování vzdálenosti mezi jednotkami

Existuje několik metod pro určení vzdálenosti mezi jednotkami. Pro přesnou lokalizaci jsou však vhodné pouze metody využívající měření doby propagace signálu ve volném prostředí.[5]

Metoda Time of Arrival (ToA)

Jestliže jsou v síti jednotky přesně časově synchronizovány, mohou z rozdílu časových razítek odeslání packetu z tagu a časových razítek přijetí packetu na jednotlivých kotvách určit čas potřebný pro propagaci signálu. Dle zjištěného času lze vypočítat vzdálenost přímo. Tato metoda však vyžaduje přesnou synchronizaci sítě a velmi

přesné oscilátory bezdrátových jednotek, což zvyšuje jejich energetickou i ekonomickou náročnost.

Metoda Round trip time (RTT)

Pro odstranění nevýhod přesné synchronizace sítě byl navržen postup měření vzdálenosti pomocí měření doby odezvy. Lze-li přesně určit čas potřebný pro zpracování odpovědi na packet, může systém měřit čas od odeslání výzvy k měření vzdálenosti do přijetí odpovědi na tuto výzvu. Od tohoto času odečte čas zpracování odpovědi T_zprac a zná dobu propagace od vysílače k přijímači a zpět.

Vzhledem k tomu, že signál se šíří rychlostí světla, lze pomocí vzorce

$$l = c \cdot t, \tag{1.1}$$

kde l je vzdálenost mezi jednotkami v metrech, c je rychlost světla a t je čas propagace, vypočítat vzdálenost mezi jednotkami jako:

$$l = c \cdot (\frac{RTT}{2} - T_{zprac}) \tag{1.2}$$

Po provedení měření mezi tagem a všemi kotvami jsou tyto vzdálenosti předány nadřízenému systému pro provedení lokalizace.

1.2 Trilaterace

Pro výpočet polohy hledané bezdrátové jednotky je využíván princip trilaterace. Lokalizační systém zjistí vzdálenosti mezi třemi či více kotvami a hledaným tagem. Pomocí tří známých vzdáleností se vytvoří pro dvojrozměrný souřadný systém tři kružnice, každá se středem v bodu, kde leží kotva. V průsečíku těchto kružnic se nachází hledaný bod, který reprezentuje souřadnice tagu. Obdobně ve trojrozměrném prostoru se vytvoří tři kulové plochy a hledá se jejich průsečík.

Změřené vzdálenosti mezi kotvami K_i a tagem jsou reprezentovány jako vektor r_i, souřadnice kotev x_i a y_i (pro 2D zaměření), respektive x_i, y_i a z_i (pro 3D zaměření) a neznámé souřadnice hledaného tagu jsou reprezentovány vektorem $[x_t, y_t]$, repsektive $[x_t, y_t, z_t]$. Pro každé měření lze sestavit rovnici o dvou, respektive třech neznámých: Pro 2D zaměření platí:

$$(x_t - x_1)^2 + (y_t - y_1)^2 = (r_1)^2 \tag{1.3}$$

Tuto rovnici lze upravit na:

$$\sqrt{(x_t - x_1)^2 + (y_t - y_1)^2} = r_1, \tag{1.4}$$

Obdobně pro 3D zaměření platí:

$$(x_t - x_1)^2 + (y_t - y_1)^2 + (z_t - z_1)^2 = (r_1)^2 \tag{1.5}$$

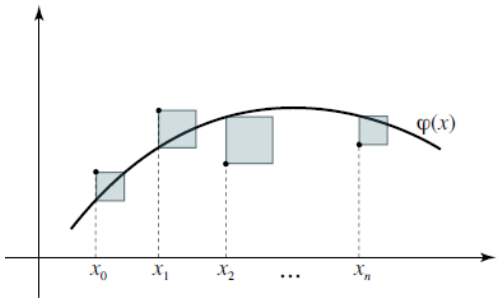
Tuto rovnici lze upravit na:

$$\sqrt{(x - x_1)^2 + (y - y_1)^2 + (z_t - z_1)^2} = r_1, \tag{1.6}$$

Z těchto dat můžeme vytvořit soustavu rovnic:

$$\begin{bmatrix} (x_t - x_0)^2 + (y_t - y_0)^2 \\ (x_t - x_1)^2 + (y_t - y_1)^2 \\ \vdots \\ (x_t - x_M)^2 + (y_t - y_M)^2 \end{bmatrix} = \begin{bmatrix} (r_0)^2 \\ (r_1)^2 \\ \vdots \\ (r_M)^2 \end{bmatrix}, \tag{1.7}$$

a jejím výpočtem určit polohu hledaného tagu. Pro výpočet této soustavy rovnic je vhodné použít některý z regresních algoritmů. Rovnice lze rozložit na matici známých koeficientů (x_k, y_k a z_k - souřadnice kotev), vektor parametrů (souřadnice x, y, z) a vektor vzdáleností k jednotlivým maticím. Jestliže má soustava rovnic více lineárně nezávislých rovnic, než neznámých (neznámou je zde souřadnice tagu v každém směru souřadného systému-tedy 2 neznámé pro 2D zaměřování a 3 neznámé pro 3D zaměřování), stává se soustava přeurčenou a je pravděpodobné, že přestane existovat přesné řešení. V takovém případě je zapotřebí nalézt řešení aproximací pomocí metody nejmenších čtverců1.1. Tato statistická metoda minimalizuje druhé mocniny odchylek dat vypočtených modelem od dat naměřených v systému. Regresí soustavy rovnic poté přímo získáme řešení s nejmenším součtem čtverců odchylek, označované jako chí-kvadrát - suma druhých mocnin rozdílu modelu a původních dat (χ^2).



Obr. 1.1: Součet čtverců odchylek. *Převzato z: [1]*

1.3 Levenberg-Markvardtův algoritmus

Pro rychlou a přesnou regresi modelu naměřených dat byl zvolen často používaný Levenberg-Markvardtův algoritmus (LM). Tento algoritmus spojuje výhody metody největšího spádu (též gradientní metoda řešení soustav rovnic), kterou je rychlá konvergence ve větší vzdálenosti od správného řešení a Gauss-Newtonovy metody, jejíž výhoda spočívá v rychlé konvergenci v blízkosti řešení. Zároveň odstraňuje nevýhody GN, což je u některých funkcí neschopnost nalézt řešení (jestliže je spuštěna s počátečním odhadem příliš daleko od hledaného minima, špatně konverguje). Tento algoritmus adaptivně mění délku kroku dle úspěšnosti poslední iterace. Jestliže je iterační krok velký, algoritmus se chová jako gradientní, ale při přibližování se hledanému řešení krok klesá a algoritmus se začne chovat podobně, jako GN. Velikost kroku je měněna po každé iteraci. Iterace je považována za úspěšnou, jestliže součet kvadratických odchylek proti předchozí iteraci poklesne a v takovém případě je krok zmenšen. Jestliže je iterace neúspěšná, krok je naopak zvětšen.

1.3.1 Matematický popis algoritmu

Nechť $f(x, p)$ je nelineární funkce, kde x je vektor nezávislých proměnných a p je vektor parametrů funkce. Dále definujme vektor $\hat{y}(x)$, který reprezentuje aproximované hodnoty a vektor $y(x)$, obsahující vstupní naměřená data. Pro vektor $\hat{y}(x, p)$ platí, že

$$\hat{y}(x, p) = f(x, p). \tag{1.8}$$

Chyba odhadu je

$$\epsilon(x, p) = y(x) - \hat{y}(x, p). \tag{1.9}$$

Metodou nejmenších čtverců je optimalizován součin $\epsilon(x, p)^T \cdot \epsilon(x, p)]$, to znamená, že se hledá p^+, pro které platí:

$$p^+ = argmin_p F(x, p), \tag{1.10}$$

kde

$$F(x, p) = \frac{1}{2}\sum_{i=1}^{m}(\epsilon_i(x, p))^2 = \frac{1}{2}||\epsilon(x, p)|| = \frac{1}{2}\epsilon(x, p)^T \cdot \epsilon(x, p), \tag{1.11}$$

kde m je počet vzorků vstupních dat. Vstupní proměnné jsou tedy (vektor vstupních dat)^2 $y(x)$, vektor nezávislých proměnných x a vektor počátečního odhadu parametrů p_0. Základem LM je lineární aproximace funkce $f(x, p)$ v okolí p. Pro malá δ_p se může Taylorův rozvoj aproximovat jako

$$f(x, p + \delta_p) \approx f(x, p) + J\delta_p, \tag{1.12}$$

kde J je Jacobiho matice $\frac{\partial f(x,p)}{\partial p}$. LM začíná s prvotním odhadem (IG) p_0 a generuje posloupnost vektorů p_1,p_2,$\cdots$, která konverguje k vektoru p^+, kde p^+ je vektor parametrů funkce, pro něž platí, že

$$\epsilon(x,p^+)^T \cdot \epsilon(x,p^+) \rightarrow \min_p(\epsilon(x,p)^T \cdot \epsilon(x,p)). \tag{1.13}$$

Je potřeba nalézt takové $delta_p$, které minimalizuje

$$||y(x) - \hat{y}(x,p+\delta_p)|| \approx ||y(x) - f(x,p) - J\delta_p|| = ||\epsilon(x,p) - J\delta_p||]. \tag{1.14}$$

Minimum je dosaženo, když $J\delta_) - \epsilon(x,p)$ je ortogonální k J, tedy

$$J^T \cdot (J\delta_p - \epsilon(x,p)) = 0, \tag{1.15}$$

neboli

$$J^T J\delta_p = J^T \epsilon(x,p), \tag{1.16}$$

což je tzv. *normální rovnice*. Matice $J^T J$ je tzv. *Hessian* (matice ($\frac{\partial^2 f(x,p)}{\partial p^2}$). LM tak řeší jemnou odchylku normální rovnice1.16. Tu lze zapsat ve tvaru:

$$N\delta_p = J^T \epsilon(x,p) \tag{1.17}$$

tento tvar je nazýván *rozšířená normální matice*. Prvky mimo diagonálu matice N jsou shodné s odpovídajícími prvky matice $J^T J$, prvky na diagonále se volí jako $N_i i = \lambda + [J^T J]_{ii}$, kde $\lambda > 0$. Tato změna diagonálních prvků se nazývá *damping*, proměnná λ je pak *damping term*.

Jestliže změna vektoru parametrů p o δ_p, kde δ_p bylo vypočteno z 1.17, vede ke snížení chyby $\epsilon(x,p)$, je tato změna akceptována. Tato iterace je následně označena jako úspěšná a hodnota λ je snížena. V opačném případě se hodnota λ zvýší a *rozšířená normální rovnice* se řeší dále, dokud není nalezena taková změna δ_p, pro kterou chyba $\epsilon(x,p)$ klesá. *Normální rozšířená rovnice* se tedy řeší tak dlouho, dokud není nalezena přípustná změna vektoru parametrů p. Pro zabezpečení redukce chyby $\epsilon(x,p)$ se hodnota *damping termu* λ mění po každé iteraci.

Je-li parametr λ nastaven na velkou hodnotu, matice N v *rozšířené normální rovnici*1.17 téměř diagonální a LM nastavuje směr kroku δ_p ve směru největšího spádu. To je vhodné, jestliže je aktuální odhad daleko od správného řešení. Algoritmus v takovém případě konverguje lineárně. Je-li λ malé, algoritmus se chová jako GN metoda. Této vlastnosti se využívá v konečných fázích odhadu, kdy se odhadované hodnoty parametrů blíží skutečným. Algoritmus v této fázi konverguje kvadraticky. LM je tedy adaptivní algoritmus, protože kontroluje vlastní damping. Vede-li krok k poklesu chyby odhadu, damping se zvýší, v opačném případě se sníží. Jestliže se řešení nachází daleko od aktuálního odhadu, algoritmus se přibližuje pomalým postupem, v blízkosti řešení však rychle konverguje.

1.3.2 Podmínky ukončení výpočtu:

Výpočet je ukončen, je-li dosaženo jednoho z následujících kritérií:

- Velikost gradientu $\epsilon(x,p)^T epsilon(x,p)$ poklesne pod prahovou hodnotu (další iterace již neposkytují výrazné zlepšení).
- Relativní změna $delta_p$ poklesne pod prahovou hodnotu (řešení bylo nalezeno).
- Chyba $epsilon^T epsilon$ poklesne pod prahovou hodnotu (řešení bylo nalezeno).
- Počet iterací překročí stanovené maximum (algoritmus nekonverguje).

Popis LM byl převzat z [4] a [2]

1.4 Knihovna MPFIT

Pro výpočet polohy byla zvolena knihovna MPFIT. Knihovna je založena na Fortranové knihovně MINPACK-1, kterou vytvořil kolektiv autorů B. Garbow, K. Hillstrom, J. More z Argonne National Laboratory, MINPACK project z roku 1980. Do jazyka C ji přeložil S. Moshier a je šířena jako Public domain.[3]

Tato knihovna implementuje regresi parametrů matematických modelů libovolné uživatelsky definované funkce pomocí LM algoritmu. Knihovna tedy sleduje vstupní data a mění parametry modelu tak, aby minimalizovala kvadratickou odchylku dat vypočtených pomocí modelu od dat naměřených a dodaných uživatelem. Knihovna umožňuje uživateli lépe řídit průběh regrese, například nastavením omezení parametrů maximální a/nebo minimální hodnotou či fixováním hodnoty parametru (hodnota parametru není během regrese měněna, je ponechána hodnota prvotního odhadu a vstupuje do funkce jako konstanta). Knihovna očekává uživatelskou funkci, která vypočítává vážené odchylky mezi aktuálním modelem a dodanými daty, vstupní data nezávislých a závislých proměnných a přesnost naměřených dat.

1.4.1 Uživatelská funkce

Uživatelská funkce (UF) musí vypočítat hodnotu výsledku za použití parametrů modelu a vstupních nezávislých proměnných a vrátit vypočtené vážené odchylky naměřených dat od dat vypočtených pomocí modelu:

```
    for (i=0; i<m; i++) {
    deviates[i] = (y[i] - f(x[i])) / err[i];
}
```

m reprezentuje počet vstupních datasetů (počet měření pro jeden tag v jednom okamžiku zaměření, čili počet kotev), *y[i]* naměřená data (závislá proměnná), *x[i]* naměřená data (nezávislá proměnná) a *f(x[i])* modelem vypočtenou funkční hodnotu pro zadané nezávislé proměnné, *err[i]* reprezentuje statistickou nepřesnost naměřených dat. Suma pole *deviates[]* je poté celkovou chybou odhadu χ^2 a tuto sumu se knihovna MPFIT snaží pomocí Levenberg-Markvardtova algoritmu minimalizovat.

1.4.2 Způsob volání funkce mpfit

Funkce *mpfit()*, která je hlavní funkcí knihovny mpfit, je deklarována takto:

```
int mpfit(mp_func funct, int m, int npar,
          double *xall, mp_par *pars, mp_config *config,
          void *private_data,
          mp_result *result);
```

Tato funkce vrací tzv. status kód, což je celočíselný kód, oznamující úspěšnost nálezu řešení. Fuknci jsou předávány tyto proměnné: ukazatel na uživatelskou funkci *mp_func funct*, celočíselný počet sad vstupních dat *int m*, celočíselný počet parametrů modelu *int npar*, ukazatel na pole parametrů, které funkce mění při hledání řešení *double *xall*, reprezentované jako desetinná čísla s dvojitou přesností, ukazatel na strukturu vlastností parametrů *mp_par *pars*, ukazatel na strukturu konfigurující chování fuknce *mpfit() mp_config *config*, ukazatel na strukturu vstupních (naměřených) dat *void *private_data* a ukazatel na strukturu obsahující informace o průběhu hledání řešení *mp_result *result*.

Status

Funkce vrací celočíselný kód informující o úspěšnosti nálezu řešení. Jak je znázorněno v tabulce 1.1, je-li status kód kladný, řešení bylo nalezeno, v opačném případě funkce selhala.

Kódy -1 až -15 jsou vyhrazeny pro chybové kódy uživatelské funkce.

Počet sad vstupních dat

Odpovídá například počtu měření pro jeden výpočet. Je vyžadováno kladné celé číslo.

Tab. 1.1: Status kódy funkce *mpfit()*

Kódy označující úspěšný průběh nálezu řešení:		
1	MP_OK_CHI	Konvergence χ^2 byla dosažena
2	MP_OK_PAR	Konvergence hodnoty parametrů byla dosažena
3	MP_OK_BOTH	Konvergence χ^2 i hodnot parametrů byla dosažena
4	MP_OK_DIR	Konvergence ortogonality
5	MP_MAXITER	Bylo dosaženo maximálního dovoleného počtu iterací
6	MP_FTOL	Změna χ^2 proti předchozímu kroku poklesla pod nastavenou mez, odhad by se již dále nezlepšil
7	MP_XTOL	Změna hledaných parametrů proti předchozímu kroku poklesla pod nastavenou mez, odhad by se již dále nezlepšil
8	MP_GTOL	Ortogonalita poklesla pod zadanou mez
Chybové kódy:		
0	MP_ERR_INPUT	obecná chyba vstupních parametrů
-16	MP_ERR_NAN	uživatelská funkce nevrátila platné číslo
-17	MP_ERR_FUNC	nebyla specifikována žádná uživatelská funkce
-18	MP_ERR_NPOINTS	uživatel neposkytl žádná vstupní data
-19	MP_ERR_NFREE	neexistují žádné volné parametry
-20	MP_ERR_MEMORY	nezdařila se alokace paměti
-21	MP_ERR_INITBOUNDS	počáteční odhad mimo dovolené hranice parametrů
-22	MP_ERR_BOUNDS	omezení parametru jsou nekonzistentní
-23	MP_ERR_PARAM	obecná chyba vstupních parametrů
-24	MP_ERR_DOF	nedostatečný počet stupňů volnosti

Počet parametrů modelu

Odpovídá například počtu zjišťovaných souřadnic. Je vyžadováno kladné celé číslo.

Uživatelská funkce

Funkce *mpfit()* si opakovaně volá uživatelskou funkci, která počítá rezidua z uživatelských dat a parametrů. Počet reziduí odpovídá počtu sad dat *m*, funkce *mpfit()* poté adaptivně mění parametry.

Pole parametrů

Uživatel poskytne funkci *mpfit()* pole parametrů *double xall[npar]*, kde *npar* je počet parametrů. Počet sad dat musí být stejný nebo vyšší, než počet volných parametrů *(m>npar)*. Při volání funkce také pole *xall[npar]* musí obsahovat prvotní odhad, označován dále jako Initial guess (IG). Po ukončení funkce *mpfit()* jsou v tomto poli uloženy hodnoty vypočteného nálezu parametrů, tedy hledané souřadnice. Původní obsah IG je přepsán.

Struktura vlastností parametrů

Pro nastavení vlastností parametrů může uživatel předat funkci ukazatel na strukturu *mp_par[npar]*, kde *npar* oznamuje počet parametrů ve struktuře, která pro každý parametr umožňuje nastavit vlastnosti popsané v tabulce 1.3.

V případě, že není potřeba specifikovat žádnou z těchto vlastností, funkci *mpfit()* se předá ukazatel `void* NULL` a funkce použije přednastavené hodnoty.

Struktura konfigurace funkce mpfit()

Pro výpočet parametrů je funkci předán ukazatel na konfigurační strukturu *mp_config* obsahující zejména nastavení kritérií pro ukončení výpočtu. Jestliže uživatel nepotřebuje nastavení měnit, lze funkci předat ukazatel `void* NULL` a funkce použije přednastavené hodnoty. Obdobně lze nastavit pouze některé parametry a ostatní nastavit na hodnotu 0, nulové hodnoty funkce sama nahradí přednastavenými. V tabulce 1.4 je vysvětleno, jaké parametry ovlivňují chování funkce *mpfit()*.

Tab. 1.3: Struktura nastavující vlastnosti parametrů

`.fixed`	Fixace parametru	hodnota parametru se při výpočtu nemění
`.limited[2]`	Limitování parametru	dvouprvkové pole, definující, zda je aplikován horní a dolní limit hodnoty parametru. `.limited[0]=1` aktivuje dolní limit (minimum), `.limited[1]=1` aktivuje horní limit (maximum)
`.limits[2]`	Limity parametru	dvouprvkové pole hodnot limitů. Je-li limitování povoleno pro dolní a/nebo horní mez, bere *mpfit()* v potaz tyto hodnoty a nedovolí parametru pokračovat přes mez.
`.*parname`	Název parametru	ukazatel na textový řetězec, umožňuje pojmenovat parametr pro potřebu výpisu ladících informací.
`.step`	Krok pro výpočet odchylek	pro každý parametr lze specifikovat velikost kroku. Je-li specifikován `relstep`, hodnota `step` je ignorována
`.relstep`	Relativní krok výpočtu	místo hodnoty `step` je použita relativní hodnota parametru.
`.side`	Varianta výpočtu derivací	nastavuje, zda se derivace počítá automaticky, jako $(f(x+h)-f(x))/h$,$f(x)-f(x-h))/h$, $(f(x+h)-f(x-h))/(2*h)$ nebo uživatelsky
`.deriv_debug`	Ladicí výpis derivací	spouští vypisování derivací vypočtených funkcí *mpfit()* a vypočtených uživatelsky pro srovnání
`.deriv_abstol`	Absolutní tolerance v ladicím výpise	
`.deriv_reltol`	Relativní tolerance v ladicím výpise	

Tab. 1.4: Struktura nastavující chování funkce *mpfit()*

`ftol`	relativní změna [chí-kva] pro ukončení výpočtu
`xtol`	relativní změna parametrů pro ukončení výpočtu
`gtol`	mez ortogonality pro ukončení výpočtu
`epsfcn`	nejmenší krok derivací
`stepfactor`	prvotní hodnota χ^2
`covtol`	tolerance výpočtu kovariance
`maxiter`	maximální počet iterací
`maxfev`	maximální počet provedení výpočtu uživatelské funkce
`douserscale`	přepíná, zda se proměnné přepočítávají dle uživ. měřítka v ladicím výpisu
`nofinitecheck`	kontroluje, zda jsou hodnoty pro výpočet platná čísla (hlásí chybu, jestliže je hodnota nekonečno nebo NaN)

Lze využít speciálních nastavení parametru *maxiter*, pokud je hodnota parametru *maxiter* nastavena na 0, funkce pouze provede kontrolu chyb, vypočte χ^2 a například zkontroluje, zda není IG mimo zvolené meze, ale neprovádí žádný regresní výpočet.

Uživatelská data

Funkci *mpfit()* lze předat ukazatel na strukturu uživatelských dat pro aktuální výpočet. Tato data jsou poté předána uživatelské funkci. Příkladem takové struktury je:

```
struct example_private_data {  /* EXAMPLE: fitting y(x) */
    double *x;          /* x - independent variable of model */
    double *y;          /* y - measured "y" values */
    double *y_error;    /* y_error - measurement uncertainty in y */
  };
```

Uživatel může definovat libovolnou obdobnou strukturu. Jestliže není k výpočtu potřeba předávat uživatelská data, lze funkci předat ukazatel `void* NULL`. Funkce *mpfit()* nijak nekontroluje ani nemění data v takové struktuře, jen předá ukazatel struktury uživatelské funkci.

Struktura výsledků

Jestliže uživatel chce znát podrobnosti o průběhu regrese, předá funkci *mpfit()* ukazatel na strukturu *mp_result*, která musí být před voláním funkce *mpfit()* vynu-

Tab. 1.6: Data obsažená ve struktuře výsledků

bestnorm	výsledné χ^2
orignorm	hodnota χ^2 vypočtená pomocí parametrů prvotního odhadu IG
niter	počet iterací funkce
nfev	počet provedení výpočtu uživatelské funkce
status	kód informující o stavu výpočtu, stejný, jako funkce vrací svým voláním
npar	počet parametrů modelu
nfree	počet volných parametrů, jež funkce měnila
npegged	počet fixovaných parametrů
nfunc	počet datových sad
resid	ukazatel na pole konečných residuí
xerror	ukazatel na pole nejistot parametrů při ukončení výpočtu
covar	ukazatel na pole obsahující kovarianční matici (pole velikosti `[npar]`$\times$`[npar]`)
version	textový řetězec s výpisem verze funkce *mpfit()*

lována. V této struktuře není uložen samotný výsledek regrese! Po návratu jsou do struktury zapsány tato data:

Jestliže není potřeba zjistit některé z parametrů reprezentovaných jako pole, lze před voláním funkce *mpfit()* uložit ukazatel `void* NULL`. V opačném případě musí uživatel vyhradit v paměti místo pro tato pole a ukazatele na ně předat do struktury ještě před voláním funkce *mpfit()*.

1.4.3 Trilaterační funkce

Pro výpočty polohy hledaného tagu v ploše a prostoru byla navržena funkce, která počítá vzdálenost mezi domnělou polohou kotvy (specifikovanou parametry *p[0]* pro x-ovou souřadnici a *p[1]* pro y-ovou souřadnici při zaměřování ve 2D, nebo parametry *p[0], p[1] a p[2]* pro souřadnice v osách *x,y,z*) a každou z kotev (jejich poloha je specifikována souřadnicí *x[i]* a *y[i]* pro 2D zaměřování, či *x[i]*, *y[i]* a *z[i]* pro 3D zaměřování). Následně odečte změřenou vzdálenost mezi kotvou a tagem od vzdálenosti vypočtené dle modelu (s použitím aktuálních hodnot parametrů) a vypočte vážená rezidua pro každou naměřenou vzdálenost tak, že tento rozdíl vydělí statistickou chybou měření. Volba, zda funkce pracuje ve 2D nebo 3D režimu, je provedena voláním funkce s proměnnou *n=2* pro 2D a *n=3* pro 3D prostor.

```
int multilatfunc(int m, int n, double *p,
   double *dr, double **dvec, void *vars)
    //multilaterační funkce pro 2D i 3D
{
  int i;  //iterátor
        //přepis pointerů na externí strukturu, oba ukazují na jedny data
  struct vars_struct_xy *v = (struct vars_struct_xy *) vars;     //2D
  struct vars_struct_xyz *v3 = (struct vars_struct_xyz *) vars;  //3D


  double *x, *y, *r, *z, *er, f;      //pomocné proměnné
  if (n==2){           //2D varianta
    x = v->x;          //přemapuje pomocné proměnné na
    y = v->y;                                          //pointery původní
        struktury
    r = v->r;
    er = v->er;
  }
  else{                //3D varianta
    x = v3->x;
    y = v3->y;
    z = v3->z;
    r = v3->r;
    er = v3->er;
  }

  for (i=0;i<m;i++)     //vypočte diferenci R od naměřené pro všechna měření
  {
      if (n==2)     //2D varianta
  // f = odmocnina z ((x_kotva - x_tag)^2 + (z_kotva - y_tag)^2)
        f=sqrt(pow((x[i]-p[0]),2)+pow((y[i]-p[1]),2));  //f odpovídá r
      else          //3D varianta
  // f = odmocnina z ((x_kotva - x_tag)^2 +
  //  + (z_kotva - y_tag)^2 + (z_kotva - z_tag)^2)
        f=sqrt(pow((x[i]-p[0]),2)+pow((y[i]-p[1]),2)+pow((z[i]-p[2]),2));

      dr[i]=(r[i]-f)/er[i];   //dr je diference naměřené od vypočtené se
        //zvážením nejistoty měření
              }
  return 0;
}
```

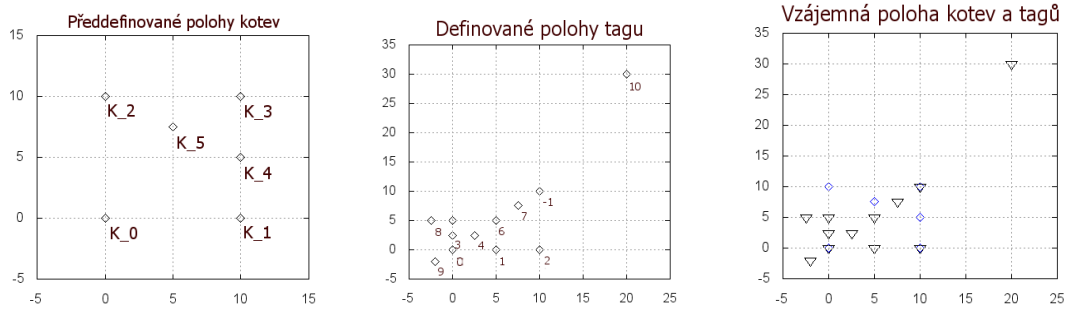
2 SIMULACE LEVENBERG-MARKVARDTOVA ALGORITMU

Hlavní částí této práce je navržení takových simulací, které umožní pochopit chování a vlastnosti funkce *mpfit()*. Dovolí odsledovat, jaké vlastnosti se mění změnou jakých parametrů a poté optimalizovat průběh nálezu řešení a simulovat, zda je tato optimalizace úspěšná.

Pro simulace výsledků byl navržen následující postup: Pro srovnání s reálnými podmínkami byl uvažován testovací prostor o velikosti 10×10 metrů. Do rohů tohoto čtverce jsou umístěny 4 kotvy, tedy na souřadnicích $K_0 = [0; 0]$, $K_1 = [10; 0]$, $K_2 = [0; 10]$ a $K_3 = [10; 10]$. Dále jsou v prostoru umístěny doplňkové kotvy na pozicích $K_4 = [10; 5]$, tedy uprostřed jedné stěny a $K_5 = [5; 7, 5]$, tedy uvnitř vymezeného prostoru. Simulační funkce je navržena tak, aby umožňovala volit, kolik kotev se použije s tím, že pokud se nevyužívají všechny, aktivuje se pouze prvních n kotev. Dále bylo předdefinováno 11 poloh tagu (viz obrázek 2.1) tak, aby byly pokryta většina speciálních i obecných případů. Bylo předpokládáno, že funkce bude vykazovat jiné chování, bude-li tag v polohách obecných a nebo speciálních, jako například poloha $t = [0; 0]$, geometrický střed oblasti, úhlopříčky, spojnice kotev či střed této spojnice. Definované polohy jsou:

- $T_0 = [0; 0]$ - nulový bod oblasti
- $T_1 = [5; 0]$ - spojnice kotev, jedna souřadnice je nulová
- $T_2 = [10; 0]$ - poloha kotvy
- $T_3 = [0; 2, 5]$ - spojnice kotev, mimo střed
- $T_4 = [2, 5; 2, 5]$ - diagonála mimo střed
- $T_5 = [0; 5]$ - spojnice kotev, střed
- $T_6 = [5; 5]$ - geometrický střed pole
- $T_7 = [7, 5; 7, 5]$ - obecná poloha v poli
- $T_8 = [-2, 5; 5]$ - kombinace kladné a záporné souřadnice, mimo pole obecně
- $T_9 = [-2; -2]$ - diagonála mimo pole kotev
- $T_{10} = [20; 30]$ - bod velmi vzdálený
- $T_{-1} = [10; 10]$ - výchozí poloha při špatném zadání, kotva K_3

Použije-li se zaměřování ve 3D prostoru, z-ová souřadnice kotev se nastavuje parametrem dle aktuální potřeby. Program pro generování dat vytváří Comma-separated values (CSV) soubory pojmenované dle nastavení symbolické konstanty `FILENAME` v hlavičce souboru `main.c`. To, jaká data má program vytvářet, je potřebné zvolit odkomentováním správného bloku kódu v hlavičce, čímž se povolí spuštění bloku kódu ve zbytku souboru.



Obr. 2.1: Poloha tagů a kotev v prostoru

Pro tyto simulace byly vytvořeny pomocné funkce, které:

- Generují jednotlivé polohy tagu.
- Simulují měření jednotlivých kotev a generují naměřené vzdálenosti mezi kotvou a tagem, případně vkládají poruchy.
- Ukládají data do souboru pro následné zpracování.

2.1 Reprezentace dat

Protože výstupem simulací je veliké množství dat, bylo potřebné zvolit jejich vhodnou reprezentaci. Data vždy obsahují souřadnice x, y a několik závislých proměnných, proto jedinou vhodnou metodou zakreslení dat byly barevné mapy. Pomocí programu gnuplot byly generovány grafy zachycující hodnoty jednotlivých proměnných v závislosti na souřadnicích x a y. Je-li výstupem simulace soubor obsahující trojrozměrná data, je každá vrstva dat vykreslena do zvláštního grafu. V grafech byly pro snazší orientaci zakresleny polohy kotev bílými kosočtverci, poloha hledaného tagu byla zakreslena bílým trojúhelníkem.

2.2 Simulace vlivu polohy prvotního odhadu na rychlost a přesnost

V první simulaci byla postupně měněna poloha IG pro 11 předem určených poloh tagu, kdy tato data jsou reprezentována 2D mapou hodnot pro každou z poloh (viz graf na obr. 2.1). Tímto vznikne 11 grafů pro jednu simulaci. Cílem simulace bylo zjistit, zda je funkce numericky stabilní a jak reaguje na speciální případy polohy tagu. Proto byly zavedeny symbolické konstanty `DEFAULT` a `DEFAULT1`, které ovlivňují, zda budou eliminovány souřadnice tagu rovny nule.

2.2.1 Podmínky simulace

Testovací funkce byla navržena tak, aby byla univerzální a všechny simulace bylo možné provést pro různá nastavení a různé prostory. Vzhledem k množství generovaných dat bylo následně zvoleno, že kotvy budou rozmístěny do rohů čtverce o velikosti 10×10 metrů s počátkem v $[0; 0]$. Poloha IG byla poté měněna v rozsahu $[-5 \dots 15]$ m pro obě osy, tedy s přesahem 5 metrů mimo čtverec vymezený kotvami. Zvolený krok byl 1 m pro každou osu. Chyba (nejistota měření) naměřených dat byla odhadnuta na 0,15 m, tedy 15 cm.

Simulace byla prováděna pro 3,4,5 a 6 kotev. Pro tuto simulaci je ve zdrojovém kódu relevantní `BLOK1`.

2.3 Vliv šumu v měření na rychlost a přesnost nálezu

Cílem další simulace bylo zjištění, jak vnesení náhodného šumu ovlivní přesnost měření. Tento šum byl specifikován jako přidání náhodně volené délky z intervalu $[-5 \dots 5]$ násobené šumovou konstantou $1/(10^{NOISE})$ pro nastavení $NOISE = [2 \cdots 4]$ ke vzdálenosti reportované každou kotvou.

2.3.1 Podmínky simulace

Při simulaci byla měněna poloha IG v rozsahu $[-5 \dots 15]$ m pro obě osy, zvolený krok byl 1 m a nejistota měření 0,15 m. Vkládání šumu do naměřených dat zajišťuje aktivace symbolické konstanty `NOISE`, její hodnota nastavuje inverzní mocninu šumu.

Simulace byla prováděna pro 3,4,5 a 6 kotev a 3 úrovně šumu. Pro tuto simulaci je ve zdrojovém kódu relevantní taktéž `BLOK1`.

2.4 Vliv chyby jedné z kotev na přesnost nálezu

Následným krokem bylo zjištění, jaký vliv má chyba jedné z kotev. V praktické aplikaci se často stává, že mezi tagem a jednou z kotev je překážka, která zpožďuje příchod signálu a tato kotva poté reportuje nesprávný výsledek s přidaným offsetem. Bylo proto simulováno, jak reaguje systém na zvyšování této chyby, jak se mění přesnost zaměření a jak se mění hodnota χ^2 v závislosti na chybně reportované

vzdálenosti jedné z kotev. Vzdálenost byla měněna jak relativně, tedy přičítáním procent k výsledku, tak absolutně, tedy přičítáním konstantního offsetu. Tento offset byl volen v rozsahu (1/100 až 50/100)m. Poté byl vnášen relativní offset jako zvýšení reportované vzdálenosti o 1-50

2.4.1 Podmínky simulace

Chyba kotvy byla testována vnášením chyby do měření jednotlivých kotev v nastavení pro 3, 4, 5 a 6 kotev. Prvotní odhad (IG) byl nastaven fixně na souřadnice [5;5], výstupem jsou data pro všechny velikosti offsetu a předdefinované polohy tagu. Pro tuto simulaci je ve zdrojovém kódu relevantní `BLOK2`

2.5 Vliv nastavení parametrů mpfit()

Z předchozí simulace bylo vyvozeno, že chybující kotva vnáší značné chyby do zaměření, a proto je potřeba najít postup, jak tuto chybu eliminovat. Protože knihovna *mpfit* má rozsáhlé možnosti nastavení, první realizovanou možností byl test vlivu parametrů funkce na zlepšení přesnosti a rychlosti nálezu. Byla tedy navržena rutina, která pro každou definovanou polohu tagu po krocích zvyšuje chybu měření zvolené kotvy a pro každý krok poté mění hodnotu zvoleného parametru. Následně se provede zaměření s pevně definovanou polohou IG.

2.5.1 Podmínky simulace

Byly simulovány změny parametrů uvedených v tabulce 1.4. Vzhledem k různým výchozím hodnotám velikostí parametrů bylo potřeba přizpůsobit testovací rutinu tak, aby dokázala smysluplně rozmítat hodnoty jednotlivých parametrů. Měnila se vzdálenost reportovaná kotvou K_0. K této kotvě byl přičítán absolutní offset v rozsahu $[0 \cdots 50]$ cm, hodnota parametru byla zvyšována v 15 krocích. Výchozí a testované hodnoty parametrů jsou uvedeny v tabulce 2.1. K této simulaci je ve zdrojovém kódu relevantní `BLOK3`

Tab. 2.1: Výchozí a nastavované hodnoty parametrů

index	`ftol`	`xtol`	`gtol`	`epsfcn`	`stepfactor`	`covtol`
default	10^{-10}	10^{-10}	10^{-10}	$2,22x10^{-16}$	100	10^{-14}
0	10^{-10}	10^{-10}	10^{-10}	10^{-16}	0,0001	10^{-14}
1	10^{-9}	10^{-9}	10^{-9}	10^{-15}	0,0005	10^{-13}
2	10^{-8}	10^{-8}	10^{-8}	10^{-14}	0,001	10^{-12}
3	10^{-7}	10^{-7}	10^{-7}	10^{-13}	0,005	10^{-11}
4	10^{-6}	10^{-6}	10^{-6}	10^{-12}	0,01	10^{-10}
5	10^{-5}	10^{-5}	10^{-5}	10^{-11}	0,05	10^{-9}
6	10^{-4}	10^{-4}	10^{-4}	10^{-10}	0,1	10^{-8}
7	10^{-3}	10^{-3}	10^{-3}	10^{-9}	0,5	10^{-7}
8	10^{-2}	10^{-2}	10^{-2}	10^{-8}	1	10^{-6}
9	10^{-1}	10^{-1}	10^{-1}	10^{-7}	5	10^{-5}
10	10^{0}	10^{0}	10^{0}	10^{-6}	10	10^{-4}
11	10^{1}	10^{1}	10^{1}	10^{-5}	50	10^{-3}
12	10^{2}	10^{2}	10^{2}	10^{-4}	100	10^{-2}
13	10^{3}	10^{3}	10^{3}	10^{-3}	500	10^{-1}
14	10^{4}	10^{4}	10^{4}	10^{-2}	1000	10^{0}

2.6 Vyhledání a vypuštění chybující kotvy z výpočtu

Protože předchozí simulace neposkytla žádnou možnost, jak optimalizovat přesnost měření, byl navržen algoritmus, který po zaměření tagu vyhledává kotvu, jejímž vypuštěním z výpočtu klesne nepřesnost zaměření tagu.

Tento algoritmus (viz obrázek 2.2) vyhledává chybující kotvu tak, že provede měření se všemi kotvami (`N`), uloží χ^2 a jestliže je χ^2 vyšší, než nastavená mez, začne postupně vypouštět jednotlivé kotvy z měření. Jakmile objeví kotvu, která chybuje, hodnota χ^2 prudce poklesne, protože model lépe odpovídá naměřeným datům.

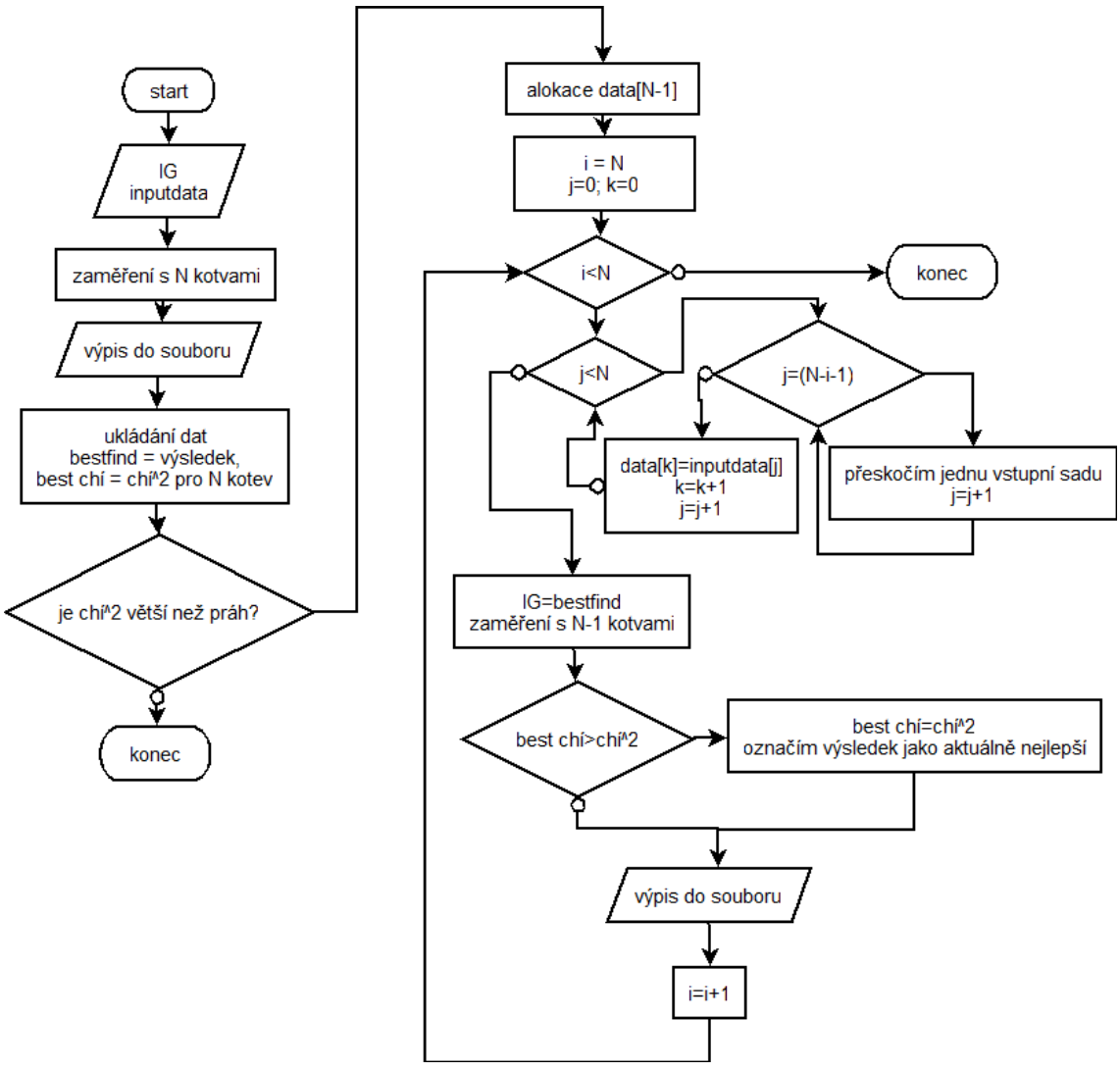
2.7 Zaměřování ve 3D prostoru

Protože v reálných podmínkách se tag pohybuje téměř vždy mimo rovinu, ve které leží kotvy, bylo simulováno, jaký vliv má vzdálenost tagu od roviny kotev, jestliže funkce zaměřuje pouze dvojrozměrně. Pro tuto simulaci byl zvolen jiný způsob reprezentace dat, kdy je poloha tagu plynule posouvána po rovině, která je rovnoběžná

s rovinou kotev, ale je od ní vzdálená 0 až 2 metry. Prvotní odhad IG byl zvolen [5,5] a [3,6] pro znázornění vlivu odlišného IG na výsledek výpočtu.

2.7.1 Podmínky simulace

Funkce *mpfit()* byla přepnuta do 3D režimu, avšak bylo zablokováno, aby měnila hodnotu parameteru z. Pro vykreslování byla měněna poloha tagu v rozsahu $[-5 \cdots 15]$ m pro obě osy. Bylo sledováno, jaký vliv má vzdálenost tagu od roviny kotev. Data byla vygenerována pro 3, 4, 5 a 6 kotev. Relevantní blok zdrojového kódu je `BLOK5`.



Obr. 2.2: Algoritmus pro vynechání kotvy, která vnáší největší chybu měření

2.8 Trojrozměrné zaměření tagu

Protože lokalizace může být nasazena i ve vysokých prostorech, bylo v dalším kroku simulováno, jak dobře funkce hledá polohu tagu ve všech třech souřadnicích. Bylo testováno, jaký vliv má poloha IG na rychlost a přesnost nálezu, zejména na správnost určení znaménka u osy Z, neboť numericky jsou správná řešení jak nad rovinou kotev, tak pod rovinou kotev.

2.8.1 Podmínky simulace

Bylo testováno, jak se funkce chová zaměřování pro 4 kotvy s umístěním tagu pod rovinu kotev, dále posun roviny kotev ze Z=0 do kladných souřadnic a vliv nastavení limitu pro výškový parametr tagu (`.limited[0]=1` v 1.3). Při simulaci byla měněna poloha tagu v rozsahu $[-5 \cdots 15]$ m v obou osách, IG byl nastaven do obecné polohy [3,6] s tím, že souřadnice Z byla buď mezi kotvami a tagem, v rovině tagu nebo v rovině kotev. Ve zdrojovém kódu tuto simulaci obsluhuje `BLOK6`

2.9 Vyhledávání chybující kotvy pomocí funkce, zpřesnění nálezu

Jednou z klíčových činností bylo vytvoření funkce, která sama vyhledá chybující kotvu, označí ji, vyloučí z měření a dodá co možná nejlepší souřadnice tagu. Proto byl algoritmus **??** přepsán do samostatné funkce a byla provedena simulace jeho úspěšnosti.

2.9.1 Podmínky simulace

Pro tuto simulaci byl IG stanoven fixně na souřadnice $[5; 5]$, souřadnice tagu byly měněny v rozsahu $[-5 \cdots 15]$ m v obou osách a bylo testováno jak iterační řešení přímo v kódu (s podrobným výpisem), tak řešení pomocí vyhledávací funkce. Zaměřování bylo testováno pro 4 kotvy, chyba byla vnášena postupně do měření všech kotev. K této simulaci je ve zdrojovém kódu relevantní `BLOK7`.

2.10 Aplikace vyřazení špatného měření pro trojrozměrná data

Jednou z možností, jak odstranit největší chyby při zaměřování dat, kdy funkce předpokládá pohyb tagu v rovině kotev, zatímco tag se pohybuje mimo tuto rovinu, je aplikovat funkci, která vyřazuje nejhorší měření i na tento případ. Proto bylo v této simulaci testováno, jaký vliv má tento postup na úspěšnost nálezu.

2.10.1 Podmínky simulace

Simulace probíhala ve 3D, kdy byla měněna poloha tagu v rozsahu $[-5 \cdots 15]$ m pro osy X a Y a $[-2 \cdots 0]$ m pro osu Z. Data byla zaměřována ve 2D, s použitím 4 kotev. Ve výpisu je zachycen průběh řešení přímo v kódu funkce main, tedy testování, kterou kotvu je vhodné vyřadit a následně výsledek zaměřovací funkce *find2D()*.

3 VÝSLEDKY SIMULACÍ

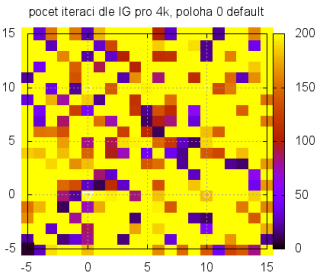
V této části je proveden rozbor dat získaných simulacemi popsanými v kapitole 2. Data byla konvertována do grafické podoby programem Gnuplot, případně byla statisticky zpracována pomocí MS Excel.

3.1 Vliv polohy prvotního odhadu na přesnost a rychlost nálezu

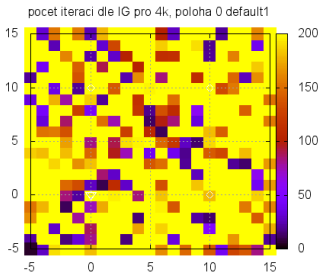
Úkolem simulace 2.2 bylo zjistit, jaký vliv má poloha IG na rychlost a přesnost nálezu tagu. Při této simulaci bylo zjištěno, že je-li x-ová i y-ová souřadnice tagu rovna nule, funkce hledá polohu tagu dlouho, často skončí až omezením maximálního počtu iterací pro 3, 4, 5 i 6 kotev. Po posunu polohy tagu o 0,01 metru mimo nulové souřadnice skokově poklesl počet iterací z maxima na průměrně 10. Tato modifikace byla v kódu ponechána, neboť v reálném prostředí není možné, aby tag přesně kopíroval polohu kotvy a vždy bude existovat odchylka, která zaručí, že algoritmus poběží korektně. Potvrdila se i obava o numerickou stabilitu výpočtu, kdy v případě násobení nulových délek vychází nulová směrnice a funkce nemůže správně rozhodnout, kterým směrem má postupovat. Při zkoumání vlivu počtu kotev na rychlost výpočtu bylo zjištěno, že tag je nalezen po průměrně 8,8 iteracích pro 2 kotvy, po 8,3 iteracích pro 3 kotvy, po 7,5 iteracích pro 4 kotvy, po 7,9 iteracích pro 5 kotev a po 8,1 iteracích pro 6 kotev.

V tabulce 3.1 jsou vypsány nejvyšší počty iterací pro jednotlivá nastavení algoritmu a jednotlivé počty kotev, tato data byla následně zpracována do grafu 3.4. Řádky označené *OK* reprezentují výsledek výpočtu po posunutí tagu mimo nulové souřadnice a posunutí IG mimo nulové souřadnice, řádky označené *default* reprezentují základní nastavení výpočtu a řádky označené *Tag mimo 0* reprezentují posun tagu mimo souřadné osy bez posunu IG.

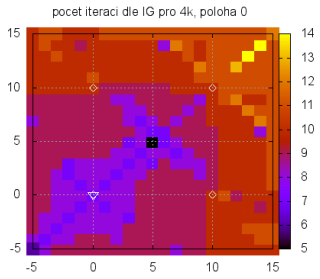
Následně byly do tabulky 3.2 vypočteny a do grafu 3.5vyneseny průměrné hodnoty počtu iterací pro jednotlivé počty kotev a polohy, již pouze pro finální nastavení algoritmu, které se vyhýbá problematickým nulovým bodům. Byly vykresleny grafy závislosti počtu iterací na poloze prvotního odhadu (IG) a závislosti χ^2 na poloze IG. V grafech závislosti počtu iterací (graf 3.6) je dobře patrné, že funkce hledá nejrychleji, je-li jako IG zvolena souřadnice blízká skutečné poloze tagu, ale i při suboptimální volbě IG dojde k nálezu tagu v relativně nízkém počtu iterací a nárůst iterací je přibližně o 50%. Pouze při volbě nejhoršího možného IG, který je však pro



Obr. 3.1: Zaměření v pozici 0 bez posunu z nuly

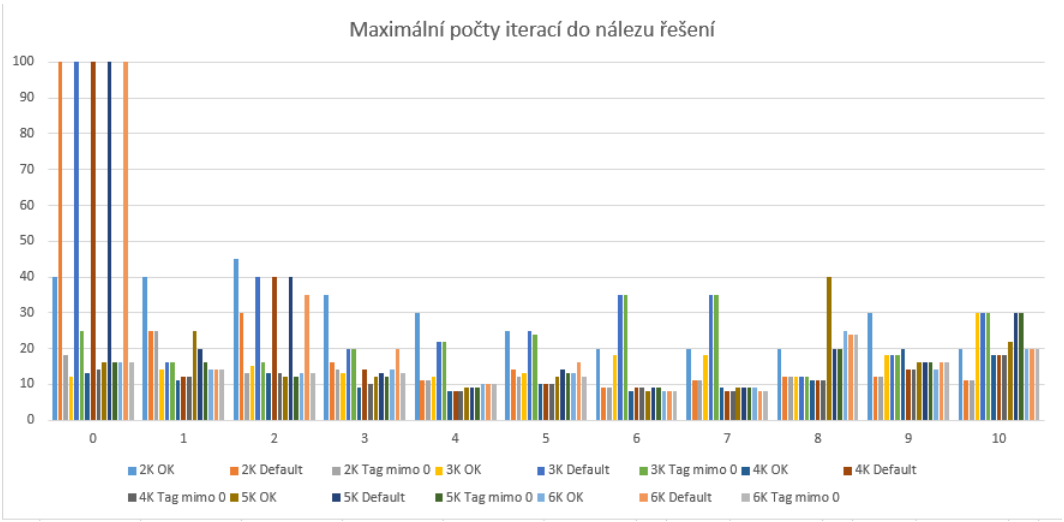


Obr. 3.2: Zaměření v pozici 0 s posunem tagu mimo nulu



Obr. 3.3: Zaměření v pozici 0 s posunem tagu i IG mimo nulu

každou polohu tagu jiný, dojde ke zvýšení počtu iterací až o 100%.



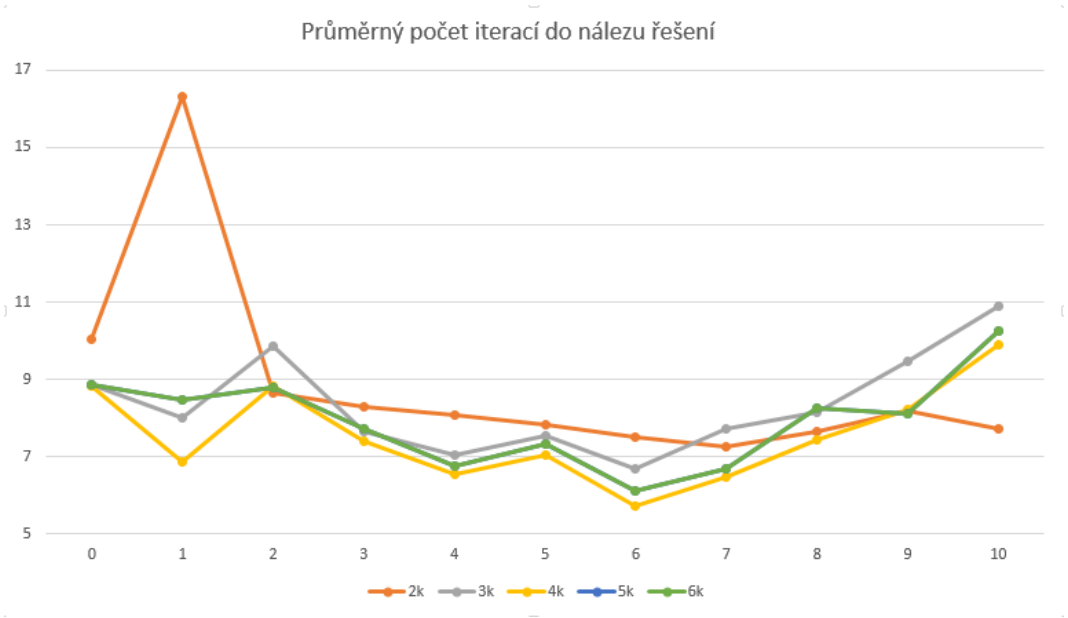
Obr. 3.4: Maximální počet iterací do náleu řešení pro jednotlivé polohy

3.2 Vliv šumu v měření na rychlost a přesnost nálezu

Další simulací bylo zjištěno, jak vnesení náhodného šumu ovlivní přesnost měření. Tento šum byl specifikován jako přidání náhodně volené délky z intervalu $[-5 \cdots 5]$ násobené šumovou konstantou $1/(10^{NOISE})$ pro nastavení $NOISE = [2 \cdots 4]$ ke vzdálenosti reportované kotvou. Z těchto měření byly vygenerovány grafy zobrazující závislost maxima, průměru a mediánu chyby zaměření tagu pro jednotlivá nastavení *NOISE* a pro jednotlivé počty kotev. Dále byly vytvořeny grafy závislostí 2D

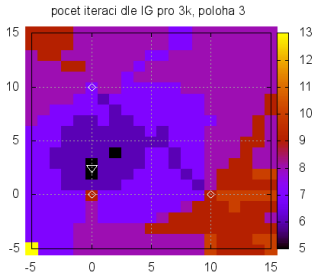
Tab. 3.1: Srovnání maximálních počtů iterací do nalezení polohy pro první simulaci

Poloha		0	1	2	3	4	5	6	7	8	9	10
2K	Default	200	25	30	16	11	14	9	11	12	12	11
	Tag mimo 0	18	25	13	14	11	12	9	11	12	12	11
	OK	40	40	45	35	30	25	20	20	20	30	20
3K	Default	200	16	40	20	22	25	35	35	12	18	30
	Tag mimo 0	25	16	16	20	22	24	35	35	12	18	30
	OK	12	14	15	13	12	13	18	18	12	18	30
4K	Default	200	12	40	14	8	10	9	8	11	14	18
	Tag mimo 0	14	12	13	10	8	10	9	8	11	14	18
	OK	13	11	13	9	8	10	8	9	11	20	18
5K	Default	200	20	40	13	9	14	9	9	20	16	30
	Tag mimo 0	16	16	12	12	9	13	9	9	20	16	30
	OK	16	25	12	12	9	12	8	9	40	16	22
6K	Default	200	14	35	20	10	16	8	8	24	16	20
	Tag mimo 0	16	14	13	13	10	12	8	8	24	16	20
	OK	16	14	13	14	10	13	8	9	25	14	20

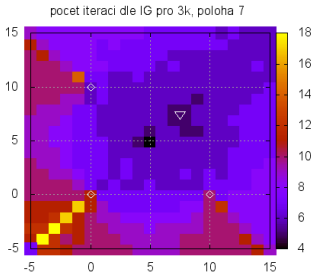


Obr. 3.5: Průměrný počet iterací pro jednotlivé polohy

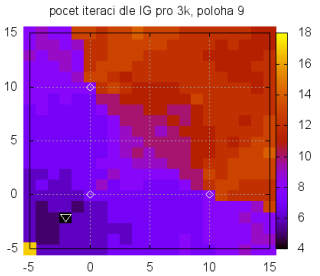
chyby, počtu iterací a χ^2 pro každou sestavu (poloha tagu, počet kotev, úroveň šumu). Vzhledem k množství těchto sad a výsledných grafů je uveden pouze reprezentativní výběr několika sad. Z grafů znázorňujících χ^2 a 2D chybu zaměření lze



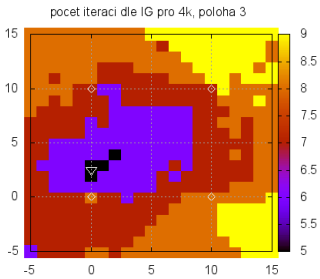
Obr. 3.6: Vliv IG na rychlost nálezu pro polohu 3 a 3 Kotvy



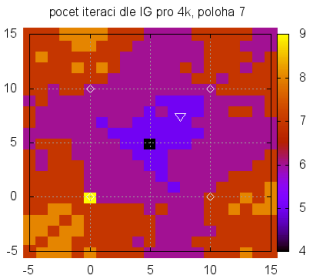
Obr. 3.7: Vliv IG na rychlost nálezu pro polohu 7 a 3 Kotvy



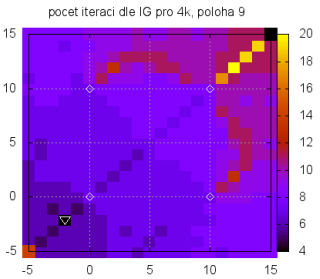
Obr. 3.8: Vliv IG na rychlost nálezu pro polohu 9 a 3 Kotvy



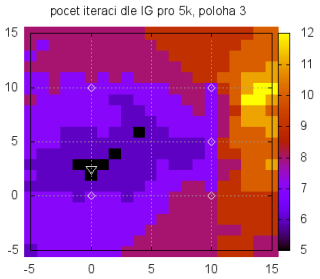
Obr. 3.9: Vliv IG na rychlost nálezu pro polohu 3 a 4 Kotvy



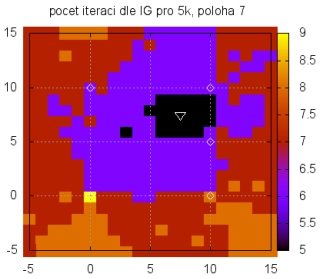
Obr. 3.10: Vliv IG na rychlost nálezu pro polohu 7 a 4 Kotvy



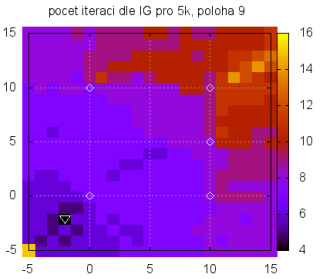
Obr. 3.11: Vliv IG na rychlost nálezu pro polohu 9 a 4 Kotvy



Obr. 3.12: Vliv IG na rychlost nálezu pro polohu 3 a 5 Kotev



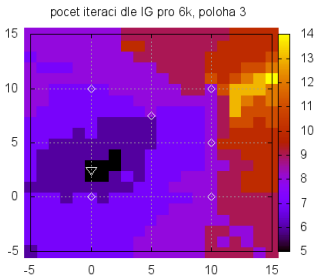
Obr. 3.13: Vliv IG na rychlost nálezu pro polohu 7 a 5 Kotev



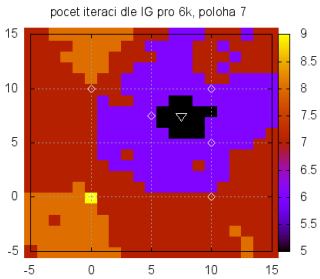
Obr. 3.14: Vliv IG na rychlost nálezu pro polohu 9 a 5 Kotev

Tab. 3.2: Průměrný počet iterací do nálezu tagu pro různé polohy

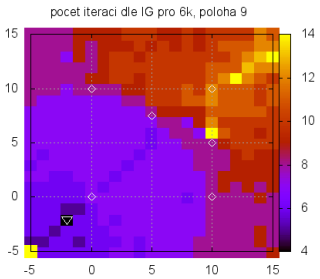
poloha	2k	3k	4k	5k	6k
0	10,05	8,85	8,81	8,86	8,86
1	16,30	7,98	6,86	8,48	8,48
2	8,63	9,86	8,84	8,80	8,80
3	8,29	7,66	7,40	7,73	7,73
4	8,09	7,02	6,55	6,73	6,73
5	7,82	7,52	7,03	7,31	7,31
6	7,50	6,67	5,72	6,10	6,10
7	7,25	7,70	6,46	6,68	6,68
8	7,63	8,15	7,43	8,25	8,25
9	8,19	9,48	8,20	8,10	8,10
10	7,70	10,90	9,91	10,25	10,25
průměr	8,87	8,34	7,56	7,93	8,09



Obr. 3.15: Vliv IG na rychlost nálezu pro polohu 3 a 6 Kotev



Obr. 3.16: Vliv IG na rychlost nálezu pro polohu 7 a 6 Kotev



Obr. 3.17: Vliv IG na rychlost nálezu pro polohu 9 a 6 Kotev

usoudit, že náhodný šum náhodně poškozuje výsledek a pozice prvotního odhadu nemá na přesné zaměření vliv, respektive není viditelná žádná oblast, ze které by funkce vyhledávala s menší chybou. Obdobně, jako při zaměřování bez zašumění, je dobře viditelné, že rušení nemá velký vliv na rychlost konvergence funkce.

3.3 Vliv chyby jedné z kotev na přesnost nálezu

V simulaci 2.4 bylo úkolem zjistit, jaký vliv na přesnost algoritmu má offset vnesený do měření jedné z kotev. Bylo zjištěno, že chyba zaměření je na offsetu kotvy závislá hodně a je-li offset absolutní, v blízkosti kotev je zaměření ovlivňováno přibližně o polovinu velikosti této chyby, jak je vidět v tabulce 3.3 ve sloupcích „absolutní offset, chyba“. Polohy „pos“ v tabulce odpovídají definovaným polohám na obrázku

2.1.

Tab. 3.3: Závislost chyby zaměření a chyby modelu na poloze tagu

	rel. offset, χ^2		rel. offset, chyba		abs. offset, χ^2		abs. offset, chyba	
pos	20%	50%	20%	50%	0,2	0,5	0,2	0,5
0	0,02	0,14	0,01	0,02	0,45	3,63	0,15	0,29
1	25,9	155,5	0,41	1,02	1,03	6,22	0,08	0,2
2	103,1	622	0,85	2,09	0,91	5,39	0,1	0,24
3	6,44	38,6	0,21	0,52	0,61	3,76	0,13	0,31
4	12,8	76,2	0,29	0,75	0,73	4,33	0,11	0,29
5	25,9	155,5	0,41	1,02	0,74	4,47	0,11	0,28
6	44,5	270	0,7	1,68	0,88	5,33	0,09	0,24
7	119,2	734	0,84	1,95	0,73	4,33	0,11	0,29
8	26,2	153	0,59	1,51	0,92	5,6	0,09	0,23
9	9,9	59,8	0,16	0,41	0,55	3,46	0,17	0,41
10	1684	10177	2,92	6,05	0,62	3,77	0,37	0,91

Vnášení absolutního offsetu měření

Byla vynesena chyba zaměření a χ^2 do grafů, následně bylo pro $0,20$ a $0,50$ m vyčteno, jaký vliv tyto chyby mají na zaměření pro jednotlivé polohy tagu. Bylo zjištěno, že při přičítání absolutního offsetu funkce reaguje na chybu tak, že:

- v blízkosti kotvy se zhoršení přesnosti zaměření zvyšuje o zhruba 75 % chyby (poloha 0),
- je-li tag blíže jiné kotvě, chyba zaměření je zhruba 55 % vneseného offsetu (poloha 7),
- je-li tag přesně na kotvě, chyba zaměření klesá na 40 % offsetu (poloha 2)
- je-li tag stejně vzdálený od několika kotev, chyba zaměření je nejnižší, klesá ke 40 % (poloha 1 a 6).
- Závislost velikosti chyby zaměření na velikosti vneseného offsetu je obvykle téměř lineární, kvadraticky roste pouze, je-li tag v bezprostřední blízkosti kotvy, která vnáší offset.
- Chyba zaměření χ^2 roste kvadraticky s nárůstem chyby zaměření.
- Stejná hodnota χ^2 neznamená stejnou chybu zaměření. Některá zaměření s vyšším χ^2 dosahují nižší chyby zaměření (poloha 2 vs. poloha 8).
- při chybě $0,20$ m se odchylka zaměření pohybuje mezi $0,08$ a $0,15$, jestliže se tag pohybuje mezi kotvami, a mezi $0,09$ a $0,37$ m, je-li tag mimo oblast ohraničenou kotvami.

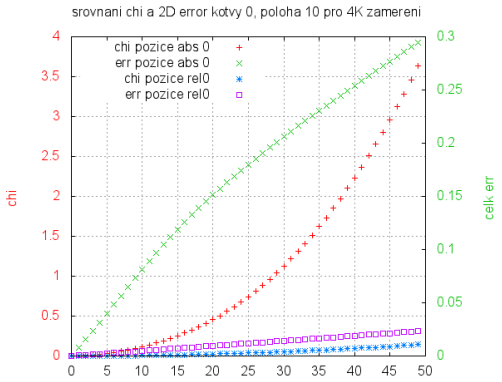
- při chybě $0,50$ m se odchylka pohybuje mezi $0,20$ a $0,31$ m pro tag mezi kotvami a $0,23$ a $0,91$ m pro tag mimo oblast ohraničenou kotvami.
- je-li tag velmi vzdálen od kotev(například jako v poloze 10, na souřadnicích [20,30]), chyba zaměření je vyšší, než offset kotvy. Z toho lze usoudit, že není vhodné umístit kotvy blízko sebe a měřit s nimi na prostoru řádově větším, než kotvy ohraničují.
- při vnesení chyby o velikosti 2 % či 5 % vzdálenosti mezi kotvami je chyba zaměření zhruba $1,5\%$ či 3%, což pro testovací prostor o velikosti 10m znamená asi 15cm či 30cm.

Vnášení relativního offsetu měření

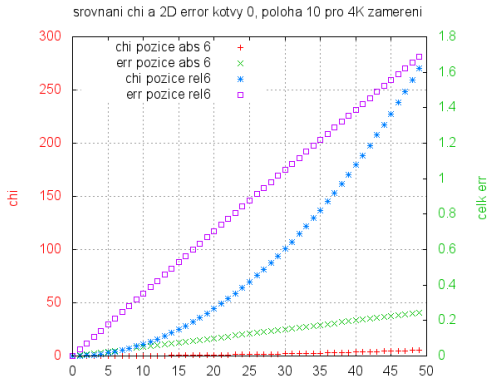
V tomto kroku byla vzdálenost reportovaná kotvou K_0 relativně navyšována o 0 % až 50 % pro každou polohu tagu. Tyto výsledky byly vyneseny do grafů vždy společně s grafy absolutní chyby a následně bylo pro 20 % a 50 % Při simulaci chování systému při relativním navýšení reportované vzdálenosti bylo zjištěno, že

- čím dále je tag od kotvy, tím vyšší je změřená chyba zaměření. To odpovídá zjištění ze simulace přičítání absolutního offsetu, tedy je-li přičítán zlomek vzdálenosti ke kotvě, jeho velikost tohoto zlomku má lineární vliv na chybu.
- Jestliže je tag vzdálen od kotvy a překážka způsobí rušení charakteru relativní chyby (například když přímý směr šíření signálu zablokuje překážka, v níž se signál šíří znatelně pomaleji, nežli ve vzduchu, velice výrazně se zvyšuje chybovost zaměření tagu.
- Chyba zaměření pro chybu vzdálenosti kotvy 20 % se pohybovala od $0,01$ do $0,91$ m pro tag umístěný mezi kotvami a mezi $0,59$ a $2,92$ m, jestliže se tag nacházel mimo prostor ohraničený kotvami.
- Pro chybu vzdálenosti kotvy 50 % se chyba zaměření pohybovala od $0,02$ do $2,09$ m pro tag umístěný mezi kotvami a mezi $0,41$ a $6,05$ m, jestliže se tag nacházel mimo prostor ohraničený kotvami.

V tabulce 3.3 jsou uvedeny výsledné chyby zaměření pro definovaných 11 poloh tagu, v grafech (3.19, 3.18) jsou znázorněny chyby zaměření a χ^2 pro rostoucí hodnotu vstupní chyby, tedy 0 až 50 % pro relativní offset a 0 až 50 cm absolutní offset. Pro názornost byl vybrán graf, kdy tag leží přímo na kotvě, která chybuje (poloha 0) a graf pro situaci, kdy je tag uprostřed oblasti vymezené kotvami (poloha 6).



Obr. 3.18: Srovnání absolutní a relativní vložené chyby pro polohu 0



Obr. 3.19: Srovnání absolutní a relativní vložené chyby pro polohu 6

3.4 Vliv nastavení parametrů mpfit()

Pomocí simulace 2.5 byla zjištěna závislost rychlosti a přesnosti hledání na hodnotách parametrů fuknce *mpfit*.

Byly otestovány všechny parametry uvedené v tabulce 2.1. Data získaná z těchto simulací byla vynesena do grafů jako barevné mapy, kdy na ose X je vynesena velikost chyby zvolené kotvy v centimetrech, na ose Y je vynesena velikost parametru a barva reprezentuje naměřenou hodnotu, kterou je buď chyba zaměření, χ^2 nebo počet iterací. Tento způsob zápisu byl zvolen proto, že výchozí nastavení parametrů je 10^{-10} u parametrů `ftol`, `xtol`, `gtol`, 10^{-14} pro parametr `covtol` a. Parametr `stepfactor`(λ) má výchozí hodnotu 100 a bylo testováno jeho snižování i zvyšování. Parametr `epsfcn` má výchozí hodnotu $2.22 \cdot 10^{-16}$, jeho testované hodnoty nebudou násobit tuto hodnotu, ale bude postupováno po krocích definovaných v tabulce 2.1. Byly vygenerovány grafy pro každý parametr a polohu tagu tak, že na měření je aplikován absolutní offset 0-0,50 jednotky, poté je v 15 krocích zvyšována hodnota zkoumaného parametru dle tabulky 2.1 a z těchto grafů bylo zjištěno následující:

- Parametr `ftol`

  -rostoucí hodnota parametru v rozsahu 10^{-10} až 10^{-1} neovlivňuje přesnost nálezu, při dalším zvýšení již přesnost klesá, ale funkce stále nachází relevantní řešení

  -v rozsahu 10^{-10} až 10^{-1} snižuje počet iterací potřebných k úspěšnému nálezu bez vlivu na přesnost, snížení počtu iterací je často o více, než 50 %.

  -hodnota prahu parametru, kdy funkce začne zvyšovat chybu výpočtu se pro různé polohy tagu nemění.
- Parametr `xtol`

-obdobně, jako u `ftol`, rostoucí hodnota v rozsahu 10^{-10} až 10^{0} neovlivňuje přesnost nálezu, při dalším zvýšení přesnost klesá.

-v rozsahu 10^{-10} až 10^{0} rostoucí hodnota snižuje počet iterací potřebných k úspěšnému nálezu bez vlivu na přesnost.

-hodnota prahu parametru, kdy funkce začne zvyšovat chybu výpočtu, se mění v rozsahu 10^{0} až 10^{1}

- Parametr `gtol`

  -zvyšování parametru v rozsahu 10^{-10} až 10^{-1} neovlivňuje chybu, při dalším zvýšení již funkce naprosto selhává v hledání a vrátí polohu IG.

  -v rozsahu 10^{-10} až 10^{0} rostoucí hodnota snižuje počet iterací potřebných k úspěšnému nálezu bez vlivu na přesnost.

  -hodnota prahu parametru, kdy funkce začne zvyšovat chybu výpočtu, se nemění.

- Parametr `epsfcn`

  -zvyšování parametru má negativní vliv na rychlost nálezu řešení i přesnost. Vhodná hodnota parametru, kdy ještě nedochází k zvýšení chyb je 10^{-16} až 10^{-13}

  -i při malém zvýšení dochází ke zvýšení počtu potřebných iterací pro nález řešení.

  -tento parametr nemá tedy význam měnit a může být ponechán na výchozí hodnotě.

- Parametr `stepfactor`

  -žádná manipulace s tímto faktorem neovlivňuje přesnost výpočtu polohy

  -je-li tento parametr nastaven na příliš nízkou hodnotu, funkce konverguje pomalu. Pro hodnoty vyšší než 1 funkce *mpfit()* hledá řešení již vždy stejně rychle a zvyšování nemá tedy smysl.

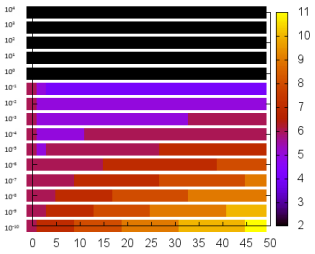
  -je-li hodnota tohoto parametru ponechána na výchozím nastavení, funkce konverguje dostatečně rychle, změnou parametru nelze dosáhnout zlepšení.

- Parametr `covtol`

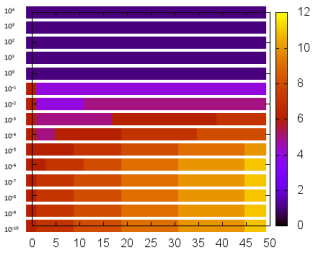
  -jakákoli změna tohoto parametru v rozsahu 10^{-14} až 10^{0} se nijak neprojevila na přesnosti nálezu ani na počtu iterací potřebném k nálezu řešení.

Srovnání vlivu jednotlivých parametrů

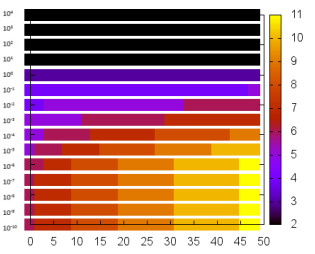
Z vynesených grafů bylo zjištěno, že dokud není překročena „prahová hodnota“ každého z parametrů `ftol`, `gtol` a `xtol`, zvyšování jejich hodnot má pozitivní vliv na rychlost nálezu řešení, tedy snižuje počet iterací potřebných k nálezu. Nejvyšší vliv na zrychlení výpočtu má parametr `ftol` a nejnižší vliv parametr `xtol`. Zvyšování



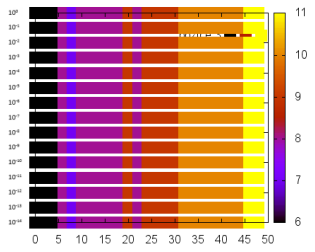
Obr. 3.20: Vliv změny parametru ftol na počet iterací



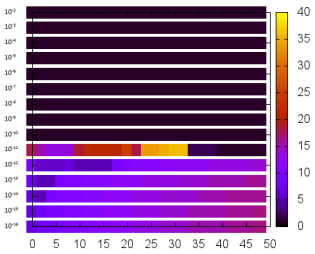
Obr. 3.21: Vliv změny parametru gtol na počet iterací



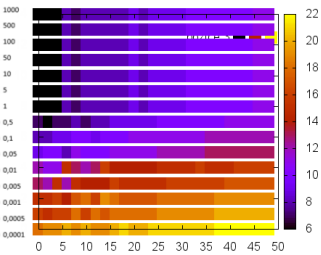
Obr. 3.22: Vliv změny parametru xtol na počet iterací



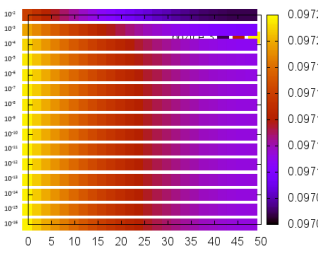
Obr. 3.23: Vliv změny parametru covtol na počet iterací



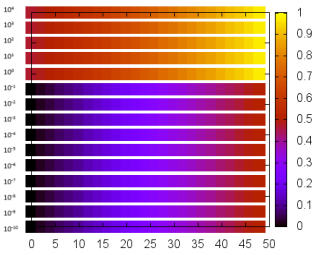
Obr. 3.24: Vliv změny parametru epsfcn na počet iterací



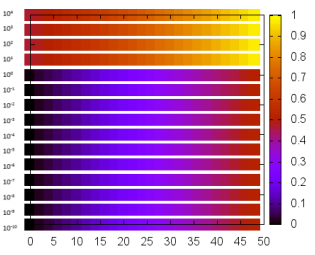
Obr. 3.25: Vliv změny parametru stepfactor na počet iterací



Obr. 3.26: Vliv změny parametru epsfcn na počet chybu výpočtu



Obr. 3.27: Vliv změny parametru ftol na počet chybu výpočtu



Obr. 3.28: Vliv změny parametru xtol na počet chybu výpočtu

parametru `gtol` má sice vliv na rychlost, ale vnáší riziko, že funkce přestane vyhledávat řešení. Výsledkem této simulace tedy je, že neexistuje vhodné nastavení parametrů, které by snížilo chybu nálezu řešení.

3.5 Vyhledání a vypuštění chybující kotvy z výpočtu

Vzhledem k neúspěšnosti pokusu o eliminaci chyby měření nastavením parametrů funkce *mpfit()* byl navržen a aplikován algoritmus (obr. 2.2) vyhledávání chybující kotvy. V této simulaci byl algoritmus pouze ověřován, zda správně vyhledává, proto vypisuje všechny pokusy o nález do souboru. V pozdějších aplikacích (v kapitole 2.9 a 2.10) již je tento algoritmus upraven tak, aby nevypisoval všechny informace, ale pouze optimalizovaný nález polohy.

Jako příklad je v tabulce 3.4 uvedeno zaměření tagu v poloze 3 (10;0, 1), kdy kotva K_2 vnáší do výpočtu chybu 34 %.

Tab. 3.4: Výpis zaměřování tagu s vypouštěním jednotlivých kotev z výpočtu

X_g	Y_g	X_ig	Y_ig	N_{iter}	χ^2	celkErr	nález	vypušt. kotva
11,082	-1,077	5,000	5,000	6	693,02	1,5335	0	99
10,876	-1,907	11,082	-1,077	9	594,18	2,1078	1	3
10,000	0,010	10,876	-1,907	7	0,00	0,0000	1	2
11,547	-1,535	10,000	0,010	6	546,52	2,1865	-1	1
11,908	-0,871	10,000	0,010	12	595,11	2,1019	-1	0

Hodnoty s indexem „_g“ jsou konečný odhad polohy, index „_ig“ značí počáteční odhad výpočtu, N_{iter} je počet iterací, *celkErr* je chyba zaměření, nález kladným číslem oznamuje zlepšení, záporným zhoršení proti dosavadnímu nejlepšímu stavu. Poslední sloupeček signalizuje, která kotva byla ve výpočtu vypuštěna.

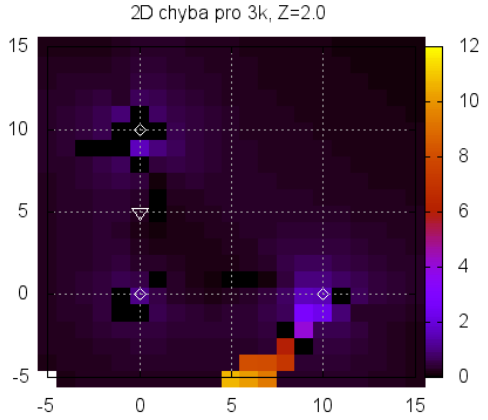
Z výsledků simulace vyplývá, že tento algoritmus správně hledá kotvu, která do měření vnáší chybu a proto byl zpracován tak, aby ukládal výstupní data všech těchto měření do paměti a poté vypsal měření, které vykazuje nejmenší chybu.

3.6 Zaměřování ve 3D prostoru

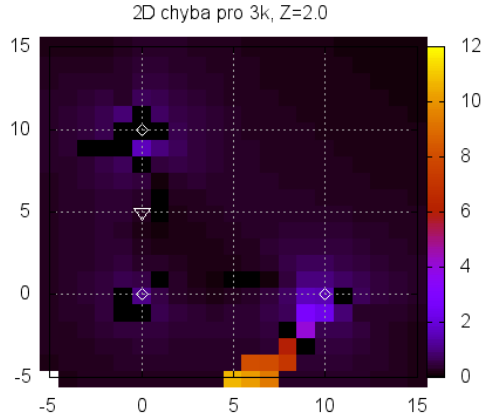
Simulace 2.7 měla za cíl testovat vzdálení tagu od roviny vyhledávání.

3.6.1 Vliv vzdálenosti tagu od roviny kotev

Bylo zjištěno, že je-li funkce *mpfit()* omezena na výpočet pouze 2 souřadnic (vlastnost `par[2].fixed` (viz tab. 1.3)parametru reprezentujícího z-ovou souřadnici je nastavena na 1), může být snadno simulován vliv vzdálení tagu od roviny kotev. Proto byla zablokována z-ová souřadnice na 0 a funkce počítala polohy tagu, který byl vzdálen 0 až 2 metry od roviny kotev. Tato data byla vygenerována pro 3, 4, 5 a 6 kotev a byl posuzován vliv polohy tagu na chybu zaměření a počet iterací výpočtu. Z grafů (3.29 až 3.32) lze odečíst, že poloha IG, mimo několik výjimek, nemá vliv na přesnost výpočtu, ovlivňuje však počet iterací. Z grafů (3.36 až 3.38) lze naopak odečíst, že chyba zaměření roste se vzdáleností tagu od roviny kotev, ale roste také, nachází-li se tag blízko kotvy.



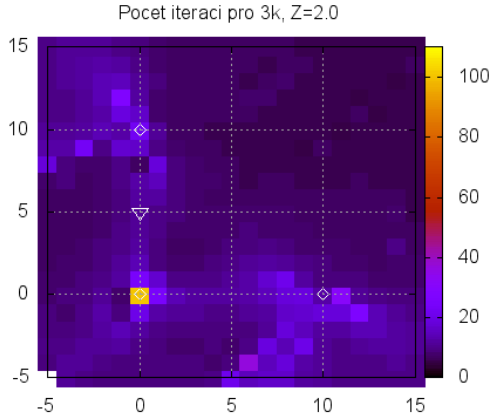
Obr. 3.29: 2D chyba pro z=2m, IG [3,6]



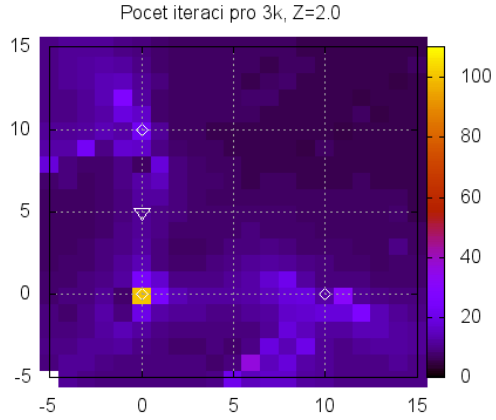
Obr. 3.30: 2D chyba pro z=2m, IG [5,5]

3.6.2 Simulace zaměřování ve 3D prostoru-vliv prostorového omezení

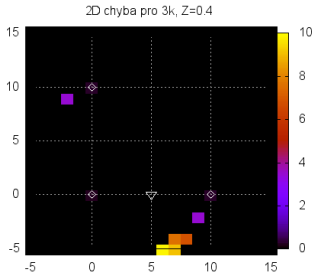
Simulací 2.8 bylo zjišťováno, jak se funkce *mpfit()* chová, má-li zpracovávat trojrozměrná data. Byly vykresleny grafy zobrazující 2D chybu zaměření i 3D chybu zaměření a bylo zjištěno, že není-li z-ová souřadnice tagu omezena parametrem `par[2].limited[]`, funkce *mpfit()*, zejména, je-li IG umístěn v rovině kotev nebo na opačné straně, než se nachází tag, selhává. Jakmile je však IG umístěn do výšky tagu



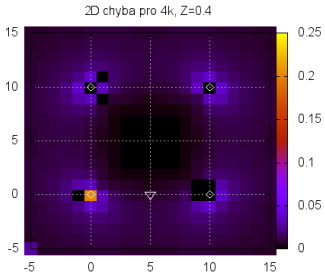
Obr. 3.31: počet iterací pro z=2m, IG [3,6]



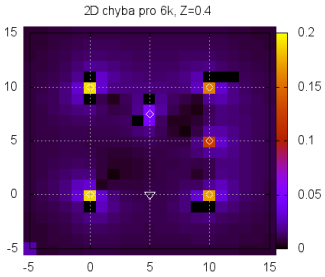
Obr. 3.32: počet iterací pro z=2m, IG [5,5]



Obr. 3.33: 2D chyba pro 3kotvy, z=0,4m



Obr. 3.34: 2D chyba pro 4kotvy, z=0,4m



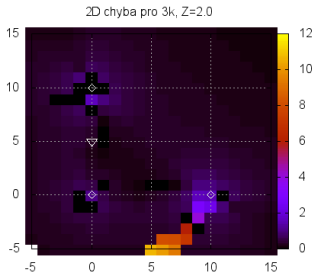
Obr. 3.35: 2D chyba pro 6kotev, z=0,4m

nebo alespoň mezi tag a rovinu kotev, funkce hledá velmi spolehlivě. Při umístění IG do roviny kotev funkce obvykle nesprávně určí hodnotu parametru Z a nedojde ke správnému řešení, což dobře zachycují grafy 2D chyby pro tento případ. V případě, že je IG umístěn mimo rovinu kotev, hledá funkce řešení velmi spolehlivě a bezchybně. Při pozorování počtu iterací vyšlo najevo, že funkce potřebuje méně iterací k nálezu, je-li IG ve výšce tagu.

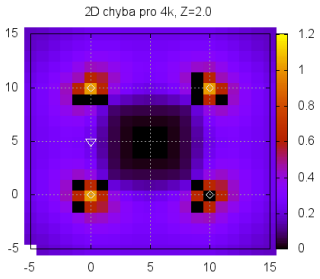
3.6.3 Implementace funkce pro vyhledávání chybující kotvy

Algoritmus 2.2 byl přepracován v samostatnou funkci, která vyhledává chybující kotvu a po svém volání vrátí nejlepší možný nález. Volání této funkce je:

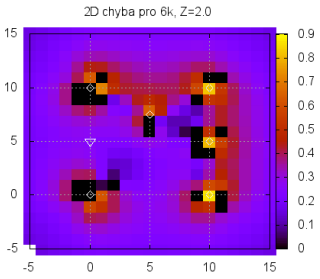
```
int find2D(struct vars_struct_xy vstup, int nKotev, double *guess, mp_result *res,
    int *kotva)
```



Obr. 3.36: 2D chyba pro 3kotvy, z=2m



Obr. 3.37: 2D chyba pro 4kotvy, z=2m



Obr. 3.38: 2D chyba pro 6kotev, z=2m

A funkce po svém průběhu vrací celočíselný chybový kód stejný, jako MPFIT a v poli `double guess` jsou výsledné souřadnice. Fuknce přijímá i ukazatel na strukturu `mp_result`, do které poté uloží výsledek nejlepšího zaměření. Dále na pozici pointeru `int kotva` funkce uloží číslo kotvy, která byla pro nejlepší zaměření vypuštěna.

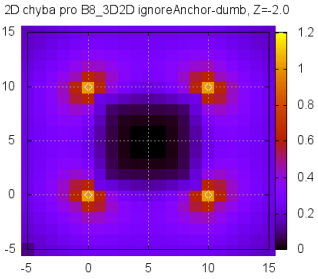
Ve výpisu v souboru B7_data.xlsx jsou v předposledním sloupci uvedeny číselně tyto stavy:

- -20 - Řešení neposkytuje zlepšení.
- 15 - Řešení poskytlo zlepšení zaměření.
- 1 - Řešení pomocí funkce *find2D()*.
- 0 - Řešení vypočtené algoritmem ve funkci *main()*.

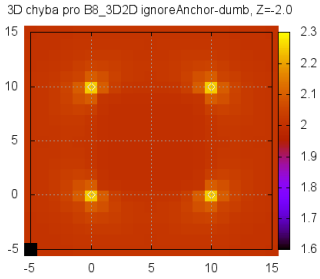
Při pohledu na průběh zaměřování se potvrdilo, že funkce zaměřuje správně a vrací všechny výsledky stejně, jako původní algoritmus ve funkci *main()*.

3.7 Aplikace vypouštění chybující kotvy při 2D zaměřování ve 3D prostoru

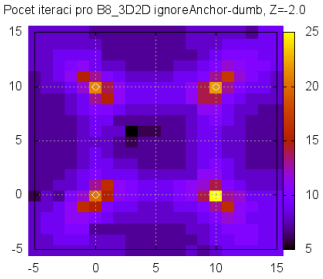
Poslední simulací, popsanou v 2.10 je zaměření tagu, který je mimo rovinu kotev, pomocí 2D zaměřovací funkce *mpfit()* a následná aplikace algoritmu na vyřazení chybné kotvy. Pomocí přepínačů v hlavičce funkce *main()* jsou vygenerovány dva datasety, kdy první obsahuje data prostým zaměřením pro všechny kotvy a druhý obsahuje pouze výstup funkce *find2D()*. Srovnáním grafů i nahlédnutím do souboru B8_srovnanihledani.xlsx se lze přesvědčit, že ve většině případů tento postup zvýší přesnost lokalizace i o více, než 50 %.Červeně podbarvené buňky v tomto souboru označují polohy tagu, kde se zaměření nezlepšilo, ale zhoršilo. Na grafech níže (graf 3.39 a další) lze srovnat velikost chyb a počet iterací k nálezu.



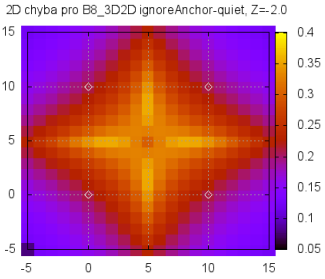
Obr. 3.39: 3D chyba bez vylučování kotvy, z=2m



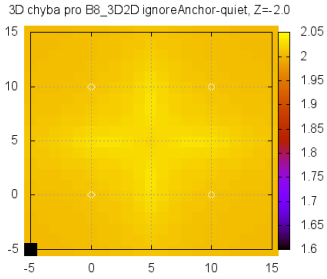
Obr. 3.40: 2D chyba bez vylučování kotvy, z=2m



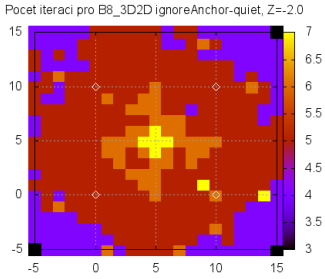
Obr. 3.41: počet iterací bez vylučování kotvy, z=2m



Obr. 3.42: 3D chyba s vylučováním kotvy, z=2m



Obr. 3.43: 2D chyba s vylučováním kotvy, z=2m



Obr. 3.44: počet iterací s vylučováním kotvy, z=2m

4 ZÁVĚR

Cílem této práce bylo zjistit, jaké jsou možnosti optimalizace algoritmů pro zaměřování v lokalizačních systémech. V první části práce je vysvětlen princip lokalizace s využitím senzorových sítí, princip měření vzdálenosti v těchto sítích a jak se provádí trilaterace. V práci je popsán Levenberg-Markvardtův algoritmus, který se používá k hledání řešení přeurčených soustav nelineárních rovnic a detailně popsány možnosti knihovny MPFIT, která tento algoritmus implementuje, její schopnosti, možnosti nastavení a způsob použití.

Dále práce navrhuje simulace, které ověřují chování Levenberg-Markvardtova algoritmu v různých situacích a hledá řešení problému, kdy v senzorové síti jedna jednotka reportuje nesprávnou vzdálenost a vnáší tím chybu do zaměřovacího procesu.

V poslední části práce shrnuje výsledky těchto simulací, nastiňuje možnosti změn parametrů a představuje řešení pro omezení vlivu chyby měření vzdálenosti mezi bezdrátovými jednotkami. Toto řešení je vyzkoušeno na systému, který vykazuje chybné měření vzdálenosti, protože se zaměřovaná jednotka nenachází v rovině výpočtu a řešení se prokáže jako účinné.

LITERATURA

[1] HASÍK, Karel. *Numerické metody* [online]. Olomouc, 2008, 119 s. [cit. 2015-05-26].
Dostupné z: http://www.slu.cz/math/cz/knihovna/ucebni-texty/Numericke-metody/Numericke-metody.pdf

[2] KOLÁČKOVÁ, Adéla. *Produkční funkce a odhad jejich parametrů: Bakalářská práce* [online]. Brno, 2013 [cit. 2015-05-24]. Dostupné z: http://is.mendelu.cz/zp/portal_zp.pl?prehled=vyhledavani;podrobnosti=49945;zp=38576;download_prace=1.

[3] MARKWARDT, Craig B. *MPFIT: A MINPACK-1 Least Squares Fitting Library in C*[online]. 2010 [cit. 2015-05-26].
Dostupné z: https://www.physics.wisc.edu/~craigm/idl/cmpfit.html

[4] STŘELEC, Martin. *Využití metody nelineárních nejmenších čtverců pro rekonstrukci přechodové charakteristiky.* [online]. 4 s. 2006 [cit. 2015-05-26].
Dostupné z: http://www.strelda.wz.cz/Clanky/StepRecon_CZ.pdf

[5] ŠIMEK, Milan. *Bezdrátové senzorové sítě* [online]. Vydání první. Brno, 2012, 169 s. [cit. 2015-05-26].
ISBN 978-80-214-4638-0.

SEZNAM OBRÁZKŮ

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

GPS Global Positioning system - globální poziční systém

WSN Wireless sensor network - bezdrátová senzorová síť

ToA Time of Arrival - čas příchodu packetu

RTT Round trip time - doba odezvy

χ^2 chí-kvadrát - suma druhých mocnin rozdílu modelu a původních dat

GN Gauss-Newtonova metoda

LM Levenberg-Markvardtův algoritmus

UF Uživatelská funkce

IG Initial guess - prvotní odhad souřadnic

CSV Comma-separated values - hodnoty oddělené čárkami