|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | TAULOGOblue%201 | |  |
| אוניברסיטת תל-אביב | | **Tel-Aviv University** |
| הפקולטה להנדסה  בי"ס להנדסת חשמל | |  | | Faculty of Engineering  School of Electrical Engineering |
| Crowd sourcing for responder positioning | | | | |
| פרויקט מס' 16-1-1-1199  דו"ח סיכום | | | | |
| מבצעים: | | | | |
|  | רון קינוליק | | 204093686 | |
|  | גל ארנון | | 203922687 | |
| מנחים: | | | | |
|  | בר-שלום עופר | | חברת "אינטל" | |
| מקום ביצוע הפרויקט:  אינטל – פ"ת | | | | |

תוכן עניינים

[תוכן עניינים 2](#_Toc485141332)

[איורים 3](#_Toc485141333)

[טבלאות 3](#_Toc485141334)

[תקציר 4](#_Toc485141335)

[1 הקדמה 4](#_Toc485141336)

[2 רקע תיאורטי 5](#_Toc485141337)

[2.1 Trilateration 5](#_Toc485141338)

[2.2 בעיית ה-Co-Linearity 5](#_Toc485141339)

[2.3 אלגוריתם 1 6](#_Toc485141340)

[2.4 אלגוריתם 2 7](#_Toc485141341)

[2.5 משקול לפי מרחקים 7](#_Toc485141342)

[3 סימולציה 7](#_Toc485141343)

[3.1 מבנה 7](#_Toc485141344)

[3.2 מידול רעש מדידה 9](#_Toc485141345)

[4 מימוש 10](#_Toc485141346)

[4.1 Parser 11](#_Toc485141347)

[4.2 ResponderFinder 11](#_Toc485141348)

[4.3 Algorithms 12](#_Toc485141349)

[5 ניתוח תוצאות 13](#_Toc485141350)

[5.1 מציאת פרמטרים אידאלים 13](#_Toc485141351)

[5.2 השוואות בין תוצאות הסימולציה לנתוני אמת 17](#_Toc485141352)

[6 סיכום, מסקנות והצעות להמשך 18](#_Toc485141353)

[מקורות 19](#_Toc485141354)

איורים

[איור ‏2.1 טרילטרציה בסיסית 5](#_Toc484783665)

[איור ‏2.2 טרילטרציה עם שגיאות מדידה 5](#_Toc484783666)

[איור ‏2.3 בעיית הקו-לינאריות 5](#_Toc484783667)

[איור ‏3.1 התפלגות שגיאת מדידה עבור מדידות האמת 9](#_Toc484783668)

[איור ‏3.2 התפלגות שגיאה אמתית לעומת מסומלצת 9](#_Toc484783669)

[איור ‏4.1 דיאגרמת זרימת מידע במערכת 10](#_Toc484783670)

[איור ‏4.2 אופן הריצה של ResponderFinder 11](#_Toc484783671)

[איור ‏4.3 הדגמה גרפית של הרצת האלגוריתם על נתוני אמת 12](#_Toc484783672)

[איור ‏5.1 CDF משוער 14](#_Toc484783673)

טבלאות

[טבלה ‏5.1 תוצאות השוואה בין אלגוריתמים ראשוניים 13](#_Toc484783674)

[טבלה ‏5.2 השוואה בין אלגוריתמי Trilateration 13](#_Toc484783675)

[טבלה ‏5.3 תוצאות למספר רספונדרים משתנה 13](#_Toc484783676)

[טבלה ‏5.4 השוואה בין פונקציות משקל 14](#_Toc484783677)

[טבלה ‏5.5 פירוט אחוזוני CDF משוער 14](#_Toc484783678)

[טבלה ‏5.6 תוצאות האלגוריתם על נתוני NLOS 15](#_Toc484783679)

[טבלה ‏5.7 תוצאות האלגוריתם על נתוני LOS 15](#_Toc484783680)

תקציר

הבעיה המתוארת הינה מציאת מיקום של מכשירים (סלולריים, מחשבים ניידים וכו') בתוך מבנים. בעבר, לא ניתן היה להשתמש בטכנולוגיות מיקום על מנת למצוא מיקום תוך-מבני מכיוון שהטכנולוגיות התבססו על לוויינים שחגים בשמיים ושמיקומם ידוע מראש.

כעת, בעזרת טכנולוגיית FTM (Fine Timing Measurements) שמתבססת על טכנולוגית Wi-Fi, ניתן למדוד מרחקים מרספונדרים שמיקומם ידוע מראש בעזרת אלגוריתמים דומים ששומשו בעבר לניווט חוץ מבני ניתן למצוא את מיקום הלקוח בתוך המבנה.

בניגוד ללווינים בחלל שמשוגרים על ידי גופים גדולים שמנהלים מסד נתונים שמתעדכן של מיקום הלווינים השונים בשמיים, הגופים שפורסים את הרספונדרים למציאת מיקום קטנים ועצמאיים (בנייני משרדים, קניונים וכו') ולא יכולים להתחייב לתחזק ולעדכן מסד נתונים אחוד שישמש כמסד נתונים עבור כל הרספונדרים הפרוסים בשטח גיאוגרפי כלשהו.

כפתרון לבעיה, הוצע לפתח אלגוריתם שמסוגל למצוא רספונדרים שמיקומם אינו ידוע תוך כדי ידיעה של מיקום של מספר רספונדרים קיימים (שאותם יעדכן מנהל המקום ברגע ההצבה שלהם). האלגוריתם ישתמש באלגוריתם מיקום תוך מבני דומה לאלגוריתם למציאת מיקום של מכשיר במרחב הבניין.

האלגוריתם ירוץ אצל הגוף שאחראי לעדכן את מסד הנתונים ובעזרתו ניתן יהיה לוודא את מיקומם של רספונדרים ידועים (ולשנות אותו במידה והוזזו) או למצוא מיקום של רספונדרים חדשים שהוצבו במרחב ללא עדכון ידני של מסד הנתונים.

מכיוון שמדידות מיקום תוך מבני בכל אופן יתבצעו על ידי לקוחות שירצו לקבל מידע על מיקומם במבנה נוכל לקבל הרבה מידע איתו נקבע מיקום מדויק של הרספונדר ונעדכן את השרת במיקומו.

לצורך פיתוח האלגוריתם פיתחנו סימולטור שמאפשר לדמות מדידות תנועה של מכשיר ניווט שנע בתוך מרחב עם פריסת רספונדרים כלשהי.

האלגוריתם מומש בסביבת Python ומאפשר למשתמש לשנות את פרמטרי האלגוריתם השונים ולהריץ על נתוני אמת ונתונים שנוצרו מסימולציה.

# הקדמה

מטרת הפרויקט הינה לאפשר מציאת מיקומם של רספונדרים שמיקומם אינו ידוע תוך קבלת מדידות מרחקים מהם ומרספונדרים שאת מיקומם אנו יודעים.

פרויקט זה נועד על מנת למנוע צורך לעדכן מסד נתונים כלשהו כאשר מזיזים/מוסיפים רספונדר מפריסה קיימת (האלגוריתם לא יעבוד על פריסה חדשה לחלוטין של רספונדרים).

לצורך פתרון הבעיה, אנו משתמשים באלגוריתמים קיימים של trilateration. תחילה, מהמדידות השונות אנו מוצאים את המיקום של המשתמש לאורך זמן תנועתו במרחב. בנוסף, יש לנו גם את המרחקים של המשתמש מהרספונדר שלא ידוע מיקומו. אם נשלב את המידע הזה נוכל שוב פעם להשתמש באלגוריתם trilateration ולמצוא את מיקום הרספונדר.

מכיוון שאלגוריתמי trilateration משומשים רבות בתעשייה (עקב שכיחות הניווט החוץ-מבני) ישנה עבודה קיימת על מימושם בצורה יעילה, אך מכיוון שטכנולוגיית הניווט התוך-מבני חדשה וטכנולוגיית הניווט החוץ-מבני מבוססת על תנועת לוויינים שמעודכנת בצורה ידנית, לא נמצאה עבודה קיימת שנועדה למצוא מיקום רספונדרים שמיקומם אינו ידוע, ולכן האלגוריתם פותח על ידינו בלבד.

# רקע תיאורטי

במהלך המחקר אחר חומר תיאורטי בנושא הפרוייקט, השיטה היחידה אשר אנו מצאנו על-מנת לבצע את המשימה הנדרשת היא להשתמש בנתונים אשר המשתמש אסף כדי למצוא את מיקומו, ומתוך מיקומו במספר זמנים שונים, למצוא את מיקום הרספונדר שאין לנו מידע מדויק עליו. לכן הבעיה העיקרית בפרוייקט הינה מציאת מיקום מדויק מתוך מרחק משוער מנקודות ידועות במרחב. החלק המרכזי של החומר התיאורטי אשר אספנו, הינו מ- [1].

## Trilateration

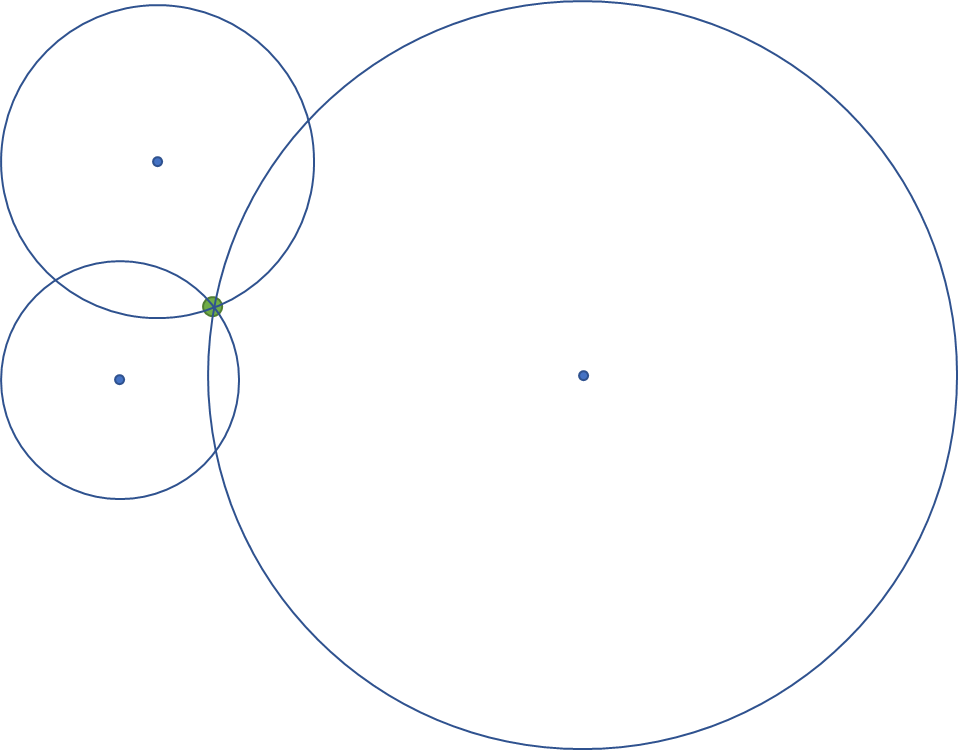
בהנתן אוסף נקודות אנו רוצים למצוא נקודה כך ש:

לכל כאשר המרחק הנמדד. בעיה זו נקראת בעיית ה-Trilateration.

הבעיה אינה פתירה כאשר יש פחות מ-3 נקודות, משום שבמקרה זה ישנם יותר נעלמים מאשר משוואות בבעיה. בהנתן 3 נקודות, תמיד ניתן לסובב ולמתוח את הצירים על-מנת ש-3 הנקודות יהיו על אותו ציר , ונקודה אחת היא בראשית. סה"כ המשוואות יהיו:

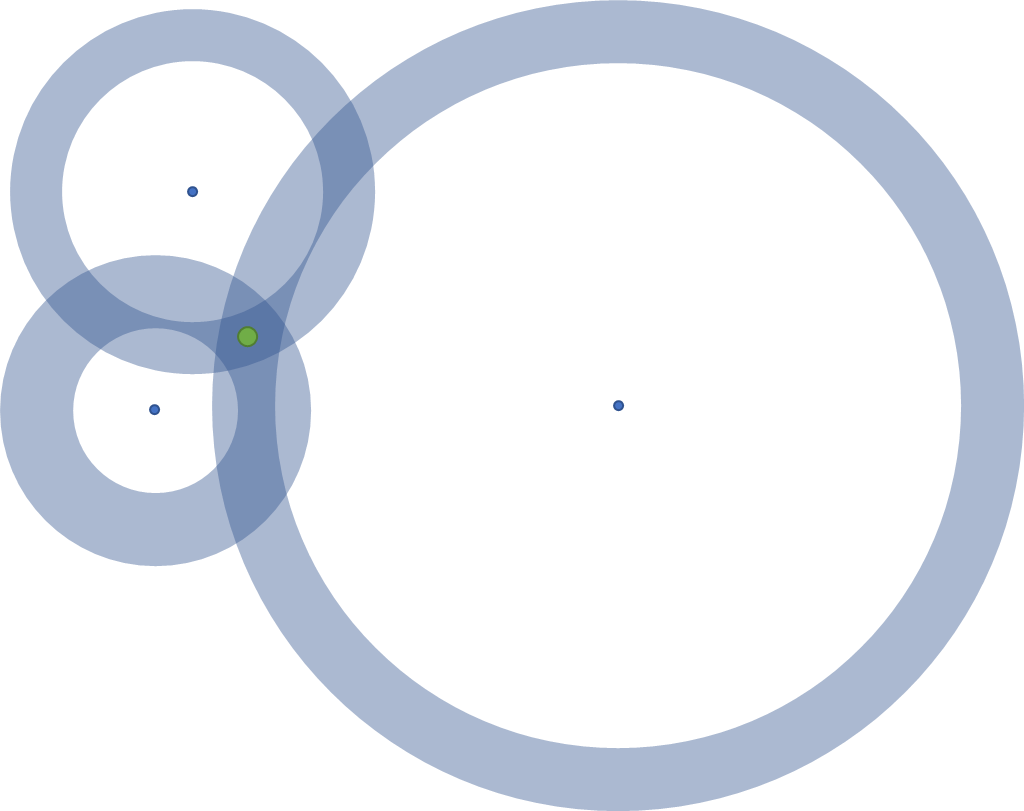
כעט, באמצעות פתרון משוואות ריבועיות ניתן להגיע לפתרון

נשים לב כי ל- ייתכנו 2 פתרונות. על-מנת לוודא את ציר נדרשות נקודות נוספות. לאורך פרוייקט זה הנחנו כי ציר הינו ידוע. לכן, למעשה ניתן לראות את הבעיה ב-2 מימדים, והפתרון שלה כאשר אין שגיאות באיור הבא:



איור . טרילטרציה בסיסית

ניתן לראות באיור מיקום לא ידוע, הנקודה הירוקה, ו-3 מיקומים ידועים במרחב, הנקודות הכחולות. באמצעות המרחק מ-3 הנקודות ניתן לשרטט מגדל סביב כל נקודה, והנקודה בהתלכדות של 3 המעגלים היא המיקום. במציאות בשל שגיאות מדידה המצב יותר דומה לאיור הבא:

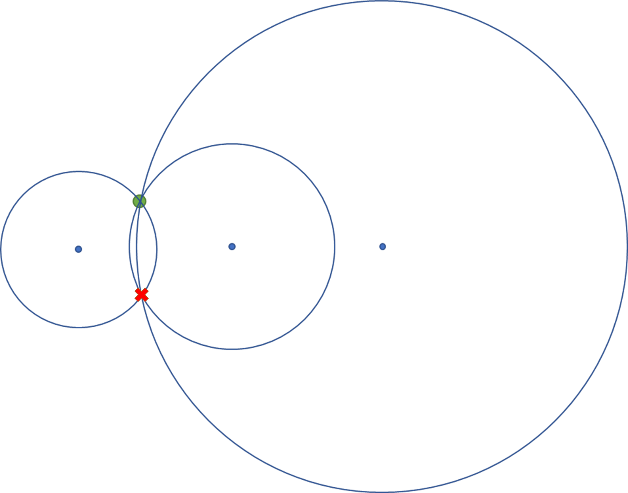


איור . טרילטרציה עם שגיאות מדידה

באיור ‏2.2 ניתן לראות כי לא ניתן לקבוע באופן פשוט, בשל שגיאות המדידה איפה בחתך של 3 הקליפות העבות הנקודה הירוקה. במשוואות מצב זה מתבטא בחוסר קיום פתרון למערכת המשוואות. לכן נדרש לבצע חישוב משוואות לינאריות כאשר קימת שגיאה.

## בעיית ה-Co-Linearity

בעיה תיאורתית משמעותית, אשר פוגעת ביכולת של כל אלגוריתם Trilateration לבצע איכון על-פי מרחקים הינה co-linearity של הנקודות אשר מיקומן ידוע. נסביר בעיה זו בהדגמה:



איור . בעיית ה-Co-Linearity

באיור ‏2.3 ניתן לראות בכחול 3 נקודות אשר מיקומן ידוע. 3 נקודות אלו יושבות על קו ישר. הנקודה הירוקה מייצגת את הנקודה שאת מיקומה במרחב אנו רוצים למצוא. מצויר מעגל סביב כל נקודה כחולה, אשר מייצג את המיקומים האפשריים של הנקוה הירוקה ביחס לנקודות אלו. כפי שניתן לראות, משום ש-3 הנקודות יושבות על ציר אחד, חיתוך 3 המעגלים המושרים על ידם מתלקדים הן בנקודה הירוקה, הפתרון האמתי, והן במיקום של האיקס האדום. ללא מידע נוסף אין דרך לדעת איזו נקודה היא הנכונה. מבחינה מתמטית למעשה נוצרה מערכת משוואות ללא פתרון יחיד. כאשר מוסיפים שגיאות במדידה החישוב נהיה מדויק אף פחות.

## אלגוריתם 1

כפי שצויין בחלק 2.1, אנו נדרשים לפתור בעיה לא לינארית עם שגיאות שיטה אחת על-מנת לפתור בעיות כאלו, מתוך הנחה שברוב המקרים יהיה מספר סביר של רספונדרים, היא לינאריזציה. נשתמש בנקודה אחת של רספונדר כרפרנס ללינאריזציה. משום שזהו רק כלי מתמטי נקודה זו שרירותית, ולכן לנוחות נבחר בנקודה הראשונה, .

לכן עבור מ:

נגיע למשוואה:

באמצעות הוספה והחסרה של נתונים מ-.

נפתח את המשוואה ונגיע ל:

כאשר הינו המרחק בין הרספונדר הראשון לרספונדר ה-. לכן סה"כ נגיע למערכת המשוואות הבאה:

זוהי מערכת לינארית (נזכור כי כל הערכים בצד הימני של המשוואה הינם מספרים). אותה נוכל לפתור באמצעות שיטות לינאריות, לדוגמא SVD כפי שמתואר ב- [1]. יש להעיר שצריך שלמטריצה מצד שמאל יהיו לפחות 3 עמודות. לכן אם משכללים את הלינאריזציה זו חובה שיהיו מדידות מלפחות 4 רספונדרים שאת מיקומם אנו יודעים.

בפועל אלגוריתם זה נותן תוצאות לא טובות, ורגיש לבעיית ה-co-linearity.

## אלגוריתם 2

נרצה למצוא פתרון אשר פועל עבור 3 רספונדרים ידועים, ופתרון אשר פחות רגיש לבעיית הco-linearity.

לכן נתבונן שוב על הבעיה המקורית שלנו.

נשים לב כי למעשה במונחים של least squares, זו בעיית האופטימיזציה הבאה:

כאשר הינה פונקציית loss שקובעת כמה משקל לשים על כל תוצאה. פונקציה זו מאפשרת מתן משקל לא שווה לנקודות אשר חורגות משמעותית מהשאר. במקרים בהם החריגות אינן גדולות, נותן לקבוע כי תהיה פונקציית הזהות, אך ישנן loss functions מקובלות אחרות, לדוגמא פונקציית ה-arctan. על בעיות מסוג זה נעשה מחקר רב. פתרון טוב לבעיה הינו שיטת non-linear least squares שיש מספר אלגוריתמים אשר ממשמים אותו, לדוגמא Newton Iteration. שיטות אלו הן איטרטיביות, ומחפשות נקודת מינימום לוקאלית לבעיה. על-מנת לממש אלגוריתם איטרטיבי אשר משתמש בגרדיאנט כדי למצוא נקודת מינימום, יש לספק גם את היעקוביאן של הבעיה. במקרה שלנו:

הבעיה עם אלגוריתמים אשר מוצאים מינימום לוקאלי הינה שיש להם סיכוי לא למצוא את המינימום הגלובלי של הפונקציה. ניתן במצב כזה לבחור נקודת התחלה אקראית, אך לכן נרצה לשאוף לתת לאלגוריתם נקודת התחלה יותר מושכלת. לכן נשתמש באלגוריתם 1 על-מנת למצוא נקודה ראשונית, ורק לאחר מכן נפעיל את אלגוריתם 2. כאשר אין לנו מספיק נקודות על-מנת להפעיל את אלגוריתם 1, נבחר נקודה אקראית כנקודה התחלתית.

## משקול לפי מרחקים

מרחק הרספונדרים מהנקודה נמדדת באמצעות גלים אלקטרומגנטיים. משום שלגלים אלו יש דעיכה פורפורציונית לאחד חלקי המרחק בריבוע, ניתן לשער כי השגיאה תגדל ביחס דומה. לכן נרצה לבחון מתן משקל גבוהה יותר למדידות של מרחקים קצרים. לפיכך נצטרך לשנות את המשוואה הלא לינארית שלנו. כעת היא נראית כך:

משום ש אינו תלוי ב, נקבל כי היעקוביאן זהה, עד כדי מכפלה במשקל:

## Outlier Rejection

במהלך בניית הפרוייקט התגבש הצורך לסנן מתוך מידע ערכים קיצוניים,כתלות בפרמטר כלשהו. על מנת לבצע זאת נאלצנו למצוא שיטה לאמוד מהם הערכים החורגים טווח מסוים. עקב השונות הגדולה אשר יש לאלגוריתמי Trilateration, רצינו אומדן אשר תלוי בחציון, ולא בתוחלת. לכן השתמשנו ב-Median Absolute Deviation (MAD), במקום בסטיית תקן. עבור סט מדידות :

ז"א, אנו נחשב את החציון של הסט. לאחר מכן נחשב את חציון הערך המוחלט של הפרשי הערכים מהחציון של הסט. REFERENC….

כך ניתן לדחות כל ערך אשר חורג במעבר מספר שרירותי של MAD-ים, בדומה לדחייה על-פי סטיות תקן אשר שימושו נפוץ.

# סימולציה

## מבנה

על-מנת לבדוק את האלגוריתמים שלנו בנינו מערכת סימולציה. אנו נדרשים ליצור מרחב בעל מספר כלשהו של רספונדרים, ליצור מסלול הליכה של משתמש במרחב זה, ולסמלץ את השגיאה במדידה של המשתמש.

המערכת מקבלת מספר פרמטרים אשר נועדו לאפשר מספר רב של סימולציות שמתאימים למספר רב של תרחישים שונים.

תחילה, הסימולטור יצור סביבה (Enviornment) שבה ירוצו תרחישים (Scenarios) שונים. ניתן לייצר הרבה תרחישים לסביבה יחידה.

הסביבה מכילה את תיחום השטח ופריסת הרספונדרים בו.

הפרמטרים ההכרחיים ליצירת הסביבה הם:

* - נקודה שמאלית תחתונה ונקודה ימנית עליונה של המתחם – על מנת לדעת היכן ניתן לפרוס את הרספונדרים השונים.
* – מספר רספונדרים שצריך לפרוס במרחב
* – במידה והמשתמש ירצה סט רספונדרים ידוע, הוא יכניס מערך של הנקודות שלהם במרחב והסימולטור לא יגריל אותם.

לאחר שהסביבה מוכנה, ניתן ליצור תרחיש. ליצירת תרחיש הפרמטרים הבאים נחוצים:

* - נקודה שמאלית תחתונה ונקודה ימנית עליונה של המתחם – על מנת לדעת היכן להגביל את מסלול התנועה.
* – מערך של מידע על רספונדרים שפרוסים במרחב.
* – זמן ריצה של הסימולציה ().
* – מהירות ממוצעת של התנועה במסלול והשונות של המהירות מסביב לממוצע ()
* – שונות של שינוי זווית במסלול בכל נקודת זמן. התוחלת של שינוי הזווית הוא .
* – נועדו לאפשר לשנות את תדירות מדידת המיקום של הלקוח לקובץ הרפרנס.
* – נועדו לאפשר לשנות את תדירות מדידת המרחקים בין הלקוח לרספונדרים השונים (בכל מרווח זמן תתבצע מדידה יחידה).

בעזרת הפרמטרים השונים, המערכת מייצרת מסלול הליכה תוך הכנסת רעש למדידות (זמנים ומרחקים) תוך הנחה שהשגיאות מתפלגות נורמלית סביב התוחלת הנתונה עם שונות נתונה.

לכל אורך מסלול ההלכים, המערכת מחשבת לאורך המסלול את מרחק הנקודה מכל הרספונדרים בסביבה ומייצרת שלושה קבצים:

* Responders\_Locations.txt – מכיל את מיקומי הרספונדרים בסביבה.
* – מכיל את מדידות ה- מהרספונדרים הידועים.
* – מכיל את מדידות המיקום האמיתי של המסלול שבוצע.

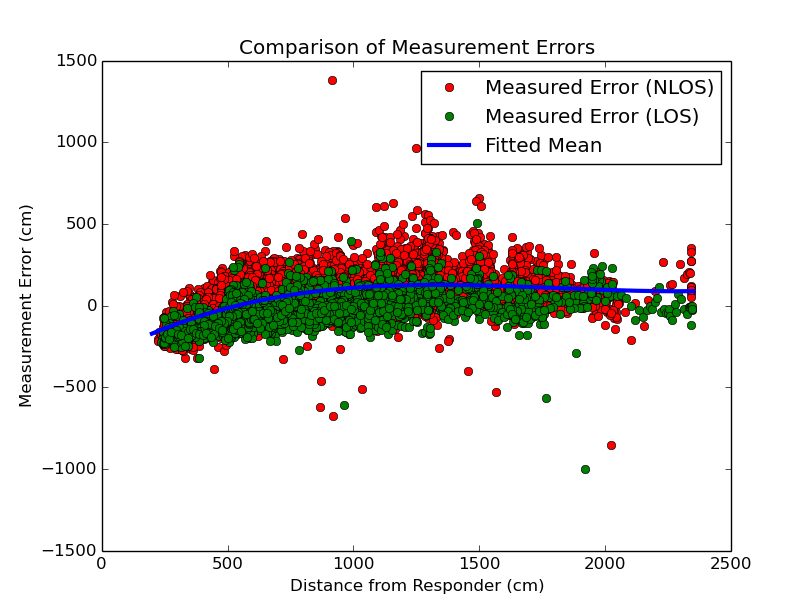
שני הקבצים הראשונים ישומשו לצורך הרצת האלגוריתמים השונים והקובץ השלישי ישמש כרפרנס לצורך בדיקות שונות (בתרחיש אמיתי הקובץ הזה לא יהיה).

כל הקבצים יוצרו בפורמט ע"פי קבצים שסופקו על ידי קבוצת ה-Location באינטל כך שניתן להריץ תחת אותה תשתית קבצים אמיתיים וקבצי סימולציה.

הסימולטור נועד להרצת Stand Alone. בנוסף, הסימולטור תומך בהרצה ישירה דרך ה- ללא כתיבה לקבצים על מנת לחסוך בזמן עבור ריצות ארוכות.

## מידול רעש מדידה

על מנת ליצור סימולציה אמינה של המציאות, נאלצנו למדל את שגיאת המדידה של מרחק מרספונדר. אנו הנחנו כי התפלגות השגיאה היא נורמאלית. כדי לבצע משימה זו, השתמשנו בקבצי המיקום המדויק של המדידות שבוצעו במשרדי אינטל. עבור כל מדידת מרחק, מצאנו את המיקום האמיתי, והחסרנו ביניהם על-מנת ליצור גרף התפלגות שגיאה לפי מרחק. כאשר התבוננו בגרף, הן עבור מדידות LOS והן עבור מדידות NLOS התבהר שיש תלות חזקה במרחק בשגיאות כפי שניתן לראות באיור ‏3.1:



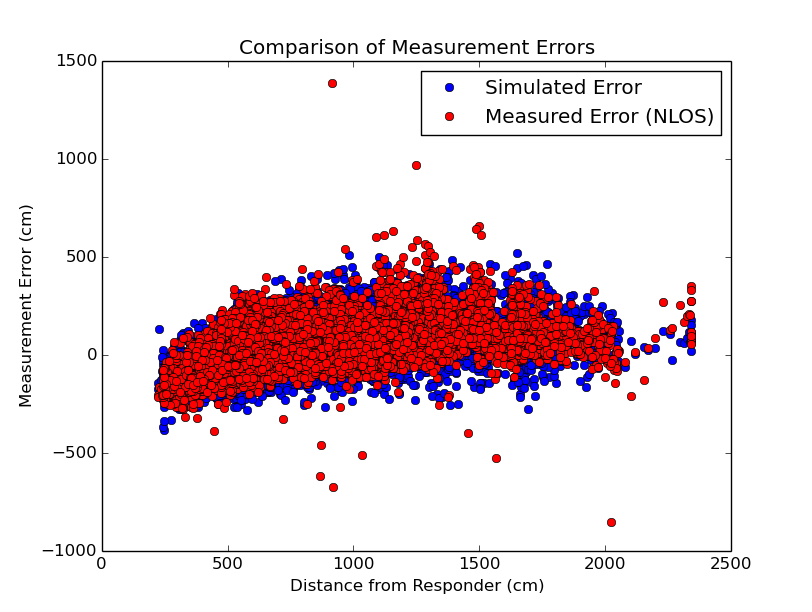
איור . התפלגות שגיאת מדידה עבור מדידות האמת

לפיכך ביצענו התאמה של פולינום לתוחלת השגיאה של כל מרחק. קיבלנו החלטה למדל את השגיאה של NLOS הן משום שזהו מקרה יותר מציאותי, והן משום שבדגימה זו היו לנו נתונים רבים יותר, ולכן האומדנים צפויים להיות אמינים יותר. בהתבוננות, גילינו כי הפולינום מדרגה הכי נמוכה אשר נתן התאמה סבירה בין תוחלת השגיאה למרחק, היה פולינום מסדר שלישי:

כאשר מרחק הרספונדר,, נמדד בסנטימטרים.

כמוכן השתמשנו באומדן חסר ההטיה המקובל כאשר מדידת השגיאה ה-, מספר הדגימות, ו- התוחלת באותה נקודה. כך מצאנו עבור כל מרחק d את ע"פ המדגם. גם עבור השונות ביצענו התאמה לפולינום מסדר שלישי, כאשר שוב ביחידות של סנטימטרים:

כך לבסוף השגיאה המסומלצת במדידה של מרחק מרספונדר אשר המרחק האמתי ממנו הוא , הינו: .



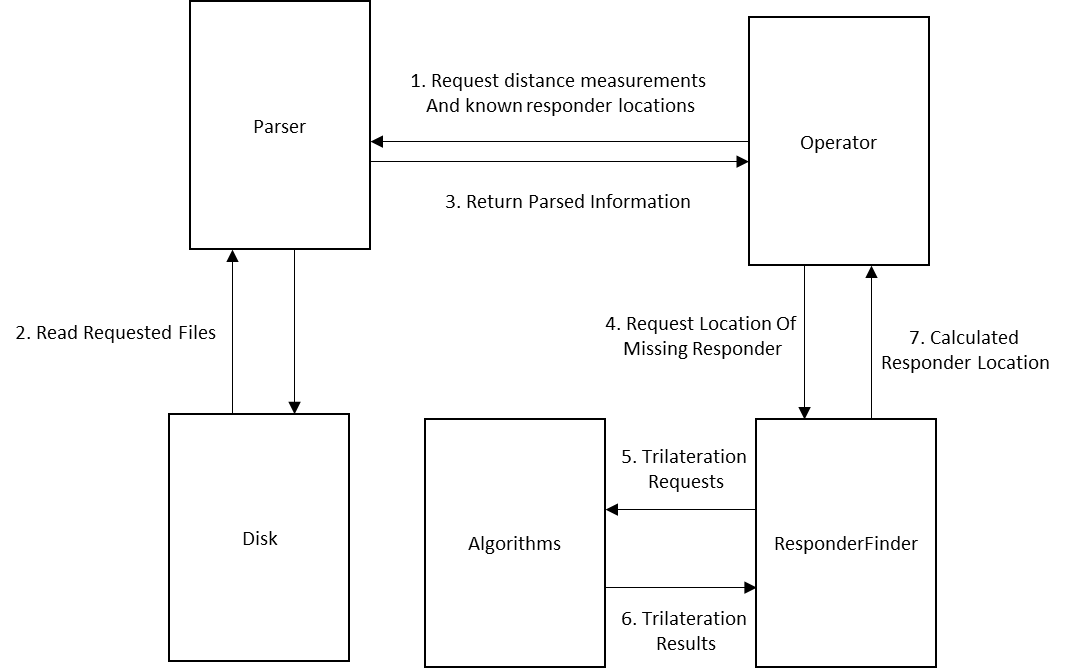
איור . התפלגות שגיאה אמתית לעומת מסומלצת

כפי שניתן לראות ב-איור ‏3.2 הסימולטור אכן יוצר תוצאות דומות לאלו שנמדדו במציאות. הסיבה שהשונות שלו מעט גדולה משל האמיצי היא מספר הנקודות הקיצוניות אשר יש במדידה האמיתית.

# מימוש

המערכת בנויה מ-3 מודולים אשר ממומשים בשפת פייתון: Parser, ResponderFinder, Algorithms.

1. Parser: תפקידו לקרוא את קבצי מדידות המרחקים, ומיקומי הרספונדרים הידועים אשר התקבלו מהמערכת, לסנן חלקים אשר אינם תקינים בקבצים, לבצע המרות יחידות ומיקומים כיישור קו לקראת שלבי עיבוד, ולפלוט את המידע בצורה של אובייקטים מסודרים.
2. ResponderFinder: משתמש במידע בפורמט אשר פולט ה-Parser על-מנת למצוא רספונדר אשר את מיקומו במרחב אנו לא יודעים.
3. Algorithms: מכיל מספר אלגוריתמים ל-Trilateration אשר משמשים כספריית עזר ל-ResponderFinder.



איור . דיאגרמת זרימת מידע במערכת

באיור ניתן לראות את מעבר המידע בין המודולים. ה-Operator, אשר רוצה להשתמש במידע אשר נאסף על-מנת למצוא את מיקומו של טרנספונדר כלשהו מבקש מה-Parser (1) שיקרא את הקבצים הרלוונטיים. ה-Parser קורא אותם (2) ומכינם לעיבוד, ומחזיר את המידע ל-Operator (3). לאחר מכן ה-Operator שולח את המידע, פרמטרים לאופטימיזציה, ואת ה-MAC של הרספונדר אשר הוא חפץ במיקומו ל-ResponderFinder (4). ResponderFinder מבצע עיבוד וחישוב, ומשתמש ב-Algorithms כספריית עזר (5) ו-(6) ולבסוף, פולט את המיקום המשוער של הרספונדר (7).

## Parser

המודול מקבל כקלט נתיבים לקבצים:

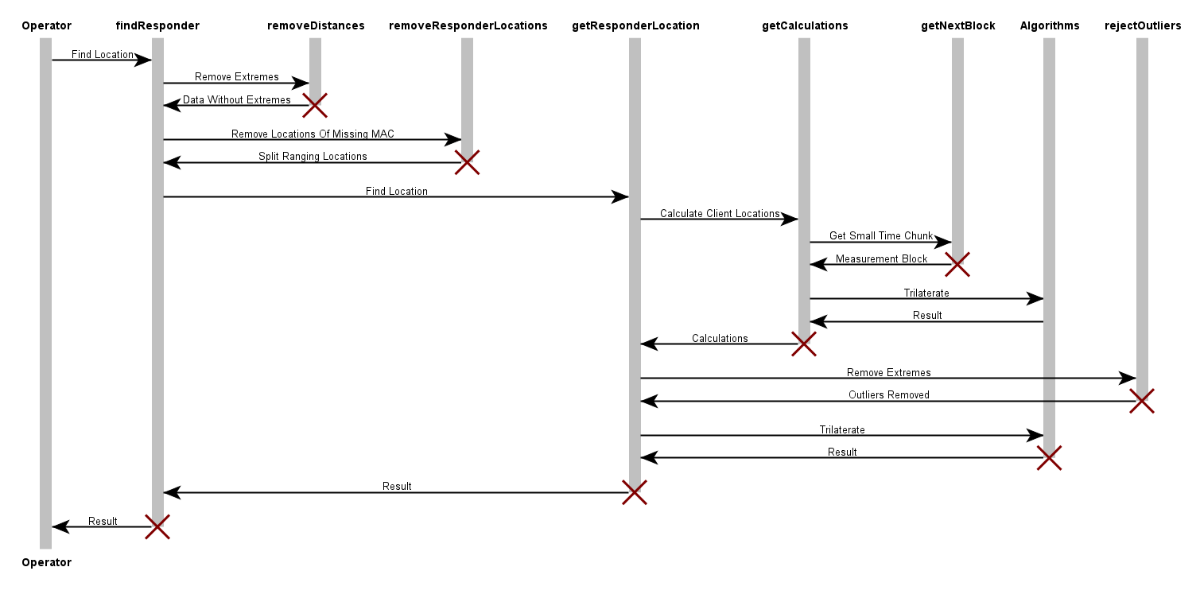
* קובץ ranging – מכיל מרחקים מכל הרספונדרים לאורך זמן
* קובץ responder locations – מכיל מיקומים ידועים של הרספונדרים במרחב (ביחידות lat/long)
* קובץ reference (אופציונלי) – מכיל מיקומים אמיתיים של הלקוח לאורך זמן (ביחידות lat/long)

המודול מפרסר את הקבצים, ממיר יחידות רלוונטיות ומחזיר עצמים ל-Opearator.

## ResponderFinder

המודול אחראי על ניתוח המידע אשר התקבל והחזרת מיקום משוער של הרספונדר.

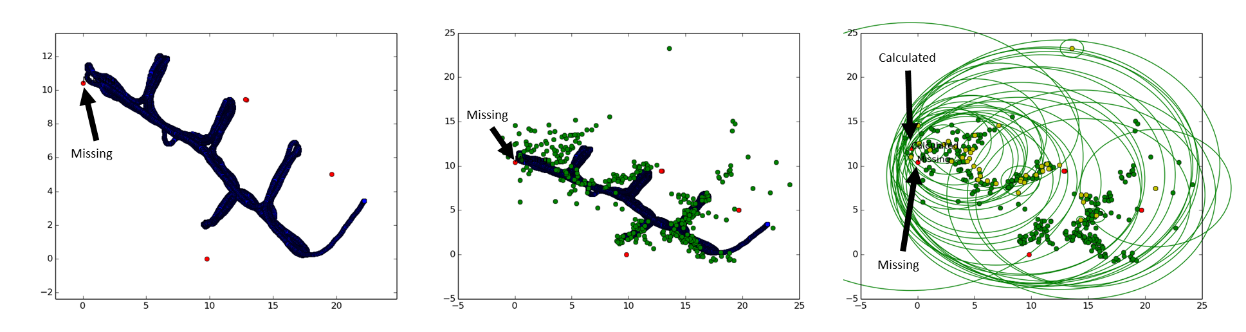
ניתן לראות את רצף ההרצה של המודול באיור הנ"ל:



איור . אופן הריצה של ResponderFinder

בקצרה, רצף ההרצה הינו:

1. בתחילת ההרצה המודול מוריד מהנתונים כל מדידת מרחק של מעל 100 מטרים (Rough Outlier Rejection). זאת משום שלאחר דיבור עם עובדי אינטל התברר כי טווח הקליטה של הרספונדרים הוא כ-40 מטרים, וערכים קיצוניים הם שגיאות מדידה.
2. לשם פשטת הניתוח בהמשך, מפצלים את המדידות למדידות שאת מיקום הרספונדר אנו יודעים, ומדידות של מרחקים מהרספונדר את מיקומו אנו מחפשים.
3. פונקציית getCalculations נקראת, כאשר מהותה לקחת מרחקים אשר נמדדו ע"י המשתמש, לפצלם לבלוקים לפי טווח זמן, אשר הנחת היסוד היא שהמשתמש לא זז בטווח זמן זה, ולחשב את המיקום המשוער של המשתמש בכל בלוק. כעט יש בידינו מיקומים משוערים של המשתמש בזמנים מסוימים, ומדידות מרחק מהרספונדר שאת מיקומו אנו מחפשים בזמנים מסוימים.
4. אם השתמשנו באלגוריתם 2, אנו מקבלים בחזרה לכל בלוק "מחיר" שמהווה מאין מדד לביטחון האלגוריתם בחישוב. כדי להמנע משגיאות מתווספות, אנו מאפשרים הסרת ערכים עם מחיר כבד במיוחד ביחס לחישובים האחרים ע"י פרמטר אשר האופרטור קובע.
5. כעת מוצאים אילו בלוקים חופפים את המרחקים אשר נמדדו מהרספונדר אשר את מיקומו אנו מחפשים. מתייחסים למיקום המחושב בכל בלוק כנקודה, והמרחק הנמדד כמרחק מהנקודה לרספונדר.
6. משתמשים במידע שחושב לעיל, ולמידע שהתקבל מהרצות עבר (והועבר לאלגוריתם ע"י המפעיל) על-מנת לבצע טרילטרציה מהמיקומים של המשתמשים למציאת המיקום של הרספונדר.



איור . הדגמה גרפית של הרצת האלגוריתם על נתוני אמת

ניתן לראות ב-[איור ‏4.3] את שלבי המערכת העיקריים באופן גרפי. בכל הגרפים הנקודות האדומות הינן הרספונדרים. הרספונדר אשר את מיקומו איננו יודע מסומד בחץ. בגרף השמאלי ביותר המסלול האמיתי של המשתמש. בגרף האמצעי ניתן לראות בירוק את המיקומים המשוערים של המשתמש (הבלוקים השונים) אשר המערכת חישבה. כפי שניתן להבחין באופן כללי המערכת מוצאת נקודות אשר דומות למסלול האמיתי. בגרף הימני ביותן ניתן לראות את המרחק אשר המערכת מייחסת מהבלוקים אל הרספונדר. משום שיש מספר גדול של נקודות, בגרף רואים את המעגל המושרה רק של כ-30 נקודות במרחב אשר לנוחות צבועות בצהוב. כמוכן ניתן לראות כי המערכת אכן מניחה את המיקום המשוער של הרספונדר בהתלקדות של מעגלים רבים, וכי הוא במרחק קרוב יחסית לאמתי (כ2.5 מטרים).

### Crowdsourcing

משום שהנחת היסוד של הפרוייקט הינה שישנם מספר רב של אנשים אשר ינועו במרחב, וביחד יאספו באופן עקיף מידע אשר עוזר למצוא את הרספונדר אשר את מידומו איננו יודעים, נדרשנו לאשפר שיטה שבה ניתן לקבל מידע, לחשב מיקום משוער של רספונדר, ובשלב כלשהו בעתיד, להשתמש במידע חדש על-מנת לשפר את החישובים. אנו מציעים 2 גישות על-מנת לבצע זאת:

1. שמירת חישובי ביניים (calcs): במהלך תהליך מציאת מיקום רספונדר, אנו מחשבים נקודות משוערות במרחב בהן המשתמש עבר. על מנת לשפר את תהליך החישוב, ניתן לשמור ב-database את חישובי ביניים אלו, ואת המרחקים המשוערים מנקודות אלו לרספנדר אשר את מיקומו איננו יודעים. כאשר מגיעים נתונים חדשים אנו נפצל נתונים אלו לבלוקים, ונמצא את המיקום המשוער של המשתמש. כעט בידינו נקודות מהמידע החדש, ונקודות מהמידע הישן. נשתמש בשניהם בשביל למצוא את הנקודה החדשה.
2. שמירת נקודה ומשקל (pts): לשיטה הקודמת 2 בעיות משמעותיות. הראשונה היא שיש לשמור ב-database את כל החישובים אשר נעשו עד כה למציאת הרספונדר. השנייה היא שישנה תלות בזמן החישוב למספר הנתונים, כך שזמן הריה הולך וגדל עם איסוף יותר מידע. לכן הגינו פתרון נוסף, שבו אנו שומרים את המיקום הסופי אשר חושר לרספונדר, ומשקל אשר פונקציה של מספר החישובים אשר הובילו לחישוב נקודה זו (מעין רמת בטחון של האלגוריתם). כאשר נקבל נקודה ומשקל, נוסיף את הנקודה ל-block-ים אשר חושבו, כנקודה אשר קרובה מאוד (כתלות במשקל) לרספונדר.

## Algorithms

במודול זה אנו מימשנו את האלגוריתמים הבסיסיים ל-Trilateration אשר פורטו בפרק 2. למימוש אלגוריתם 1 היו לרשותינו 2 פותרי מערכות לינאריות (אחד של ספריית numpy והשני של ספריית scipy) אשר פותרים ברקע את הבעיה בצורה מעט שונה. האלגוריתמים:

1. lstsq: פתרון של ספריית numpy אשר משתמש בפתרון SVD עם מספר אופטימיזציות הן למהירות והן לקירוב. פירוט על האלגוריתם ניתן למצוא ב- [2]. בשאר מאמר זה נכנה את שיטה זו lstsq.
2. pinverse: פתרון של ספריית scipy אשר למעשה מוצא בשיטת SVD את ה-pseudoinverse של המטריצה. את ה-pseudoinverse ניתן להכפיל בחלק הימני של המשוואות כדי לפתור את הבעיה, כפי שמתואר ב- [1]. בשאר מאמר זה נכנה את שיטה זו svd. פירוט על האלגוריתם ניתן למצוא ב- [3].

למעשה, 2 הפתרונות הנ"ל, בעולם ללא שגיאות, אמורים לתת פתרונות זהים, אך הן עקב שגיאות קירוב מכונה, והן משום שלעיתים למערכות הלינאריות ייתכנו מספר פתרונות שונים מקורבים, לעיתים אנו מצפים לקבל תוצאות שונות.

עבור אלגוריתם 2, הלא-לינארי, השתמשנו בפתרות least\_squares של ספריית scipy אשר הוכיח עצמו כמהיר ואמין. כמוכן, כפי שפורט בחלק 2 על-מנת לבל נקודה התחלתית סבירה האלגוריתם יכול להשתמש ב2 הפתרונות של אלגוריתם 1, או בנקודה אקראית במרחק סביר סביב אוסף שאר הרספונדרים. לאלגוריתם פונקציות עזר אשר מחשבות עבורה את ה-residuals בכל נקודה, ואת ה-Jacobian Matrix. בהמשך המאמר נכנה פתרון זה nonlinear.

# ניתוח תוצאות

## מציאת פרמטרים אידאלים

על מנת למצוא את הפרמטרים הטובים ביותר עבור האלגוריתם, נאלצנו לקבוע ערכי ברירת מחדל עבור הסימולטור. הערכים אשר נבחרו הם :....

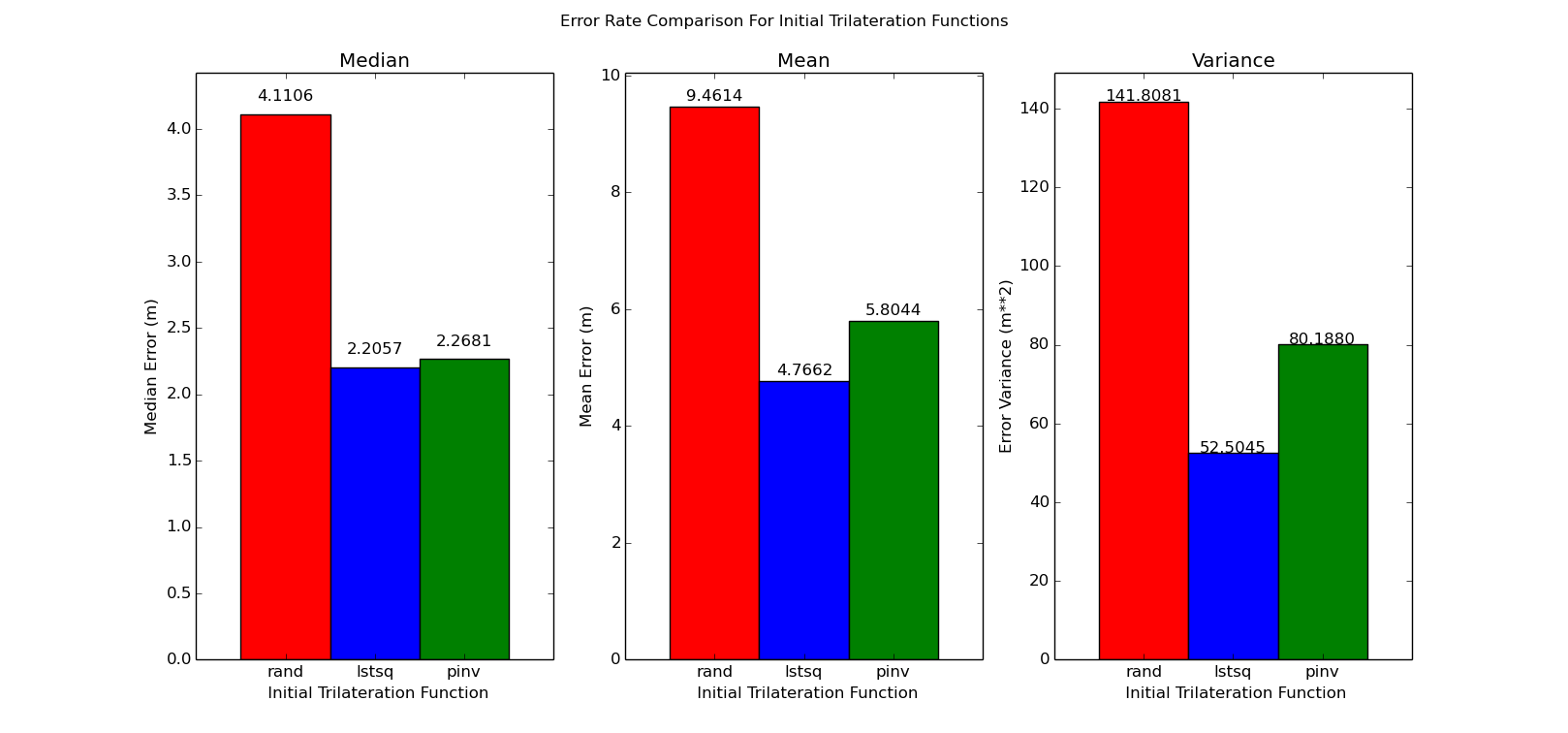
ערכים אלו נבחרו משום שמניסויים בזמן הפיתוח גילינו כי עבור מסלול ארוך, הערכים מתכנסים (??), וכן את המהירות הממוצעת חישבנו ע"י ניתוח קבצי הידע אשר סופקו לנו ע"י חברת אינטל (LOS, NLOS???)... כמוכן, בניסויים הללו לא ביצענו Crowd-Sourcing כלל, על מנל לאמוד את האלגוריתם הבסיסי. את ההשפעה של שיטות ה-Crowdsourcing נבדוק בחלק ???.

בנוסף, קבענו ערכי ברירת מחדל עבור האלגוריתם. אלא אם כן נאמר אחרת, ערכי כל ניסוי הינם:

* Crop: 1
* TimeSpan: 2
* Multilat\_func: nonlinear
* Initial: lstsq
* Weight\_funciton:1

### השוואה בין אלגוריתמים

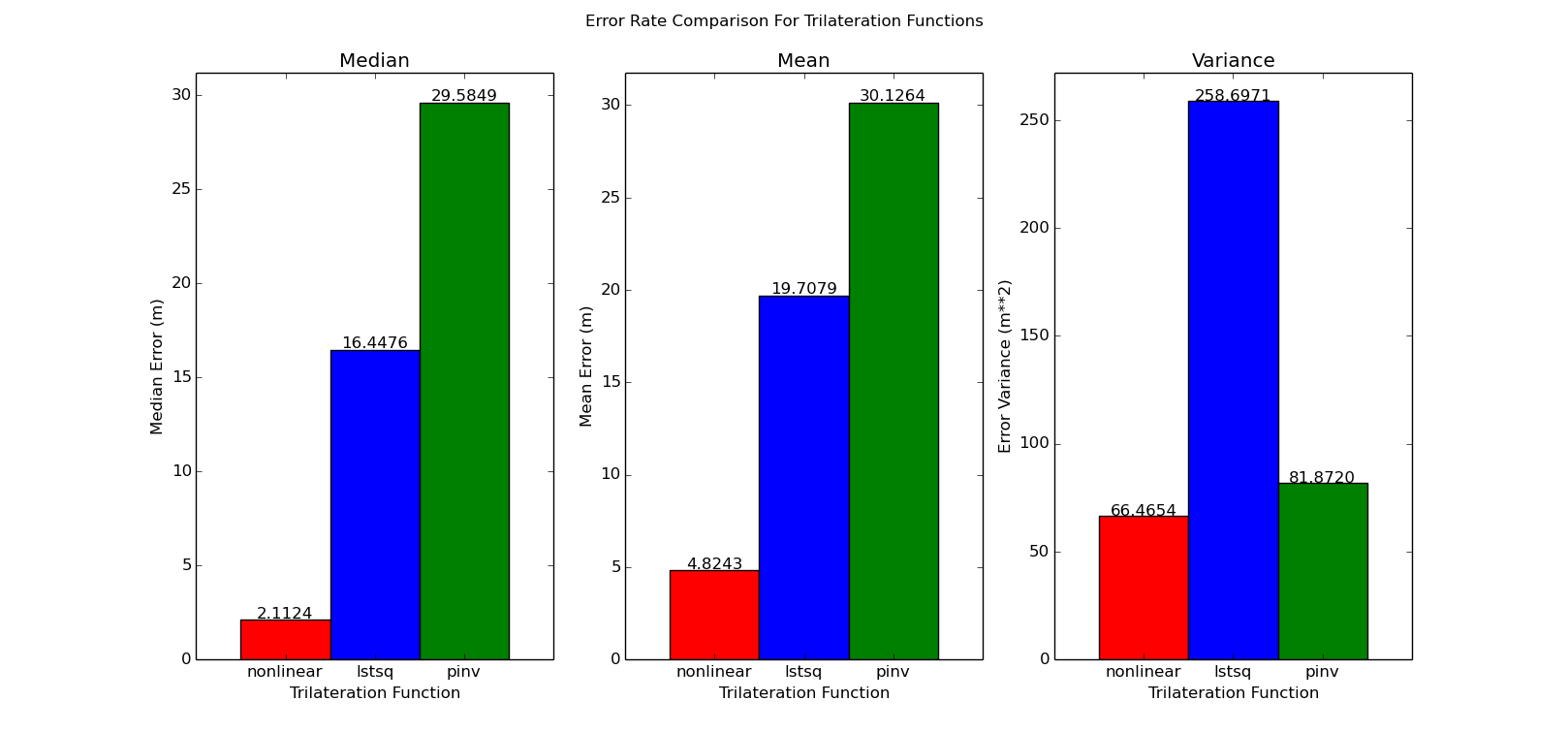
תחילה, נזכיר כי, כפי שפורט בחלק 3, עבור אלגוריתם 2 אנו נדרשים לספק נקודה התחלתית כלשהי. נרצה לאמוד איזו מן השיטות נותת ביצועים טובים יותר.



איור . השוואה בין פונקציות התחלתיות

כפי שניתן לראות באיור ‏5.1, שימוש ב-2 האלגוריתמים לפתרון של בעיה לינארית למציאת נקודה התחלתית טובה עבור האלגוריתם נותנות תוצאה סופית טובה משמעותית ביחס לבחירת נקודה באקראי. בין האלגוריתמים ניתן להבחין כי החציון שלהם דומה, אך ה-Solver המכונה lstsq מביא לממוצע ושונות משמעותית טובים יותר מאשר של ה-Solver אשר כיננו pinv. לכן נסיק מהניסוי כי lstsq נותן את הביצועים הטובים ביותר.

כעט נרצה להשוות בין הרצות של 2 ה-Solver-ים עבור אלגוריתם 1, לפתרון עבור אלגוריתם 2, על-מנת לדעת על איזה אלגוריתם להתמקד בפרוייקט.



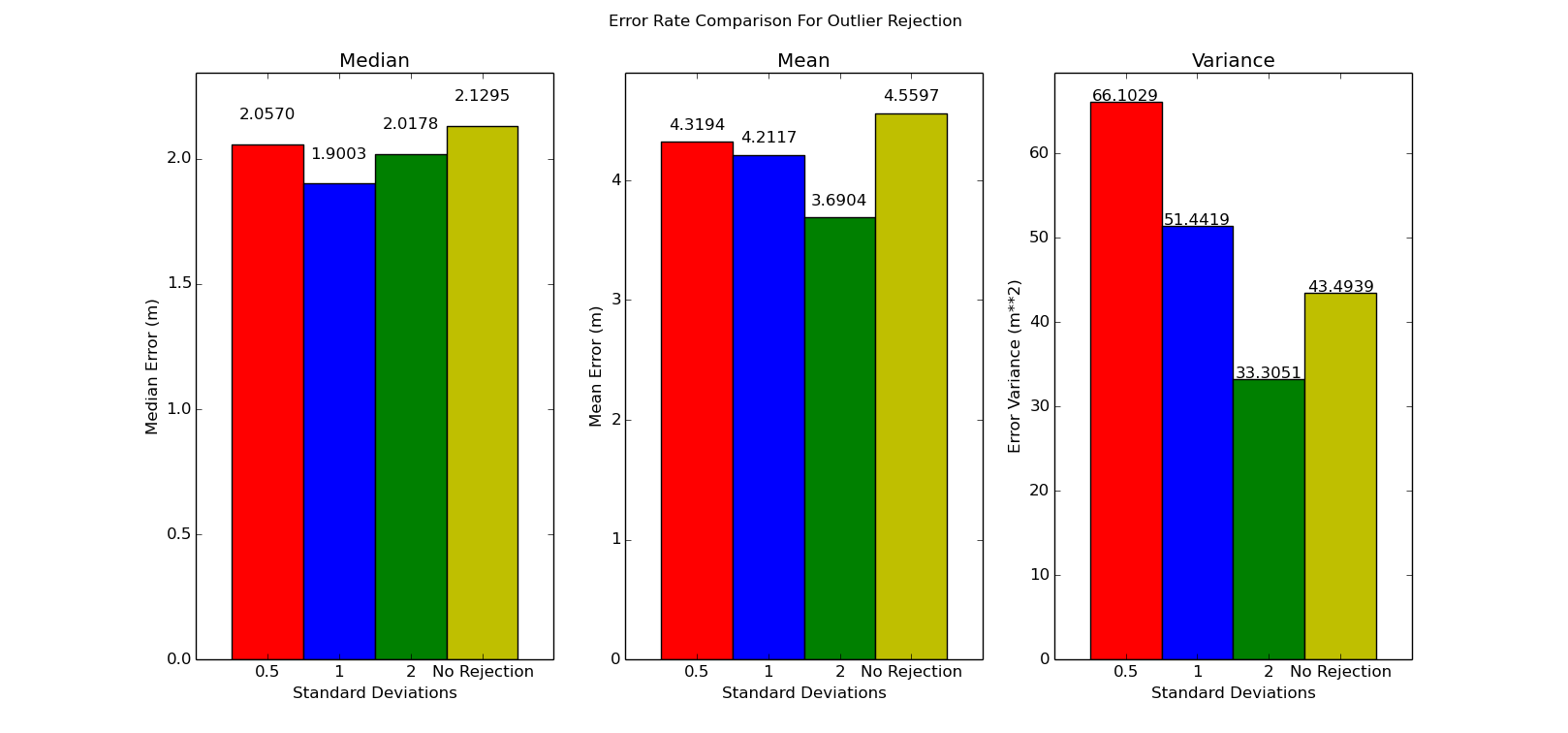
איור . השוואה בין אלגוריתמים

באיור ‏5.2 ניתן לראות כי 2 המימושים של אלגוריתם 1 נותנים תוצאות לא טובות. אנו משערים כי חוסר דיוק זה נובע מהעובדה שהשגיאות, ומספר רב של מדידות יוצר מערכת משוואות בעלת מספר פתרונות, והאלגוריתמים הלינאריים אינם מתמודדים עם בעיה זו היטב. למרות זאת, ניתן לראות כי מימוש ה-lstsq טוב משמעותית ממימוש ה-pinv, אם כי עם שונות גדולה בהרבה. אנו משארים כי זו הסיבה בגללה קיבלנו תוצאות טובות יותר באיור ‏5.1 עבור lstsq מאשר pinv. אלגוריתם 2 מביא לתוצאות הטובות ביותר באופן משמעותי.

### השוואת האלגוריתם במספר פרמטרים

לאחר מציאת העובדה כי אלגוריתם 2 מניב תוצאות טובות משמעותית מאלגוריתם 1, שאפנו להגיע לפרמטרים אידאיליים עבור אלגוריתם זה, ולבסוף לבדוק את התוצאות על מדידות אמתיות אשר נמדדו בשטח.

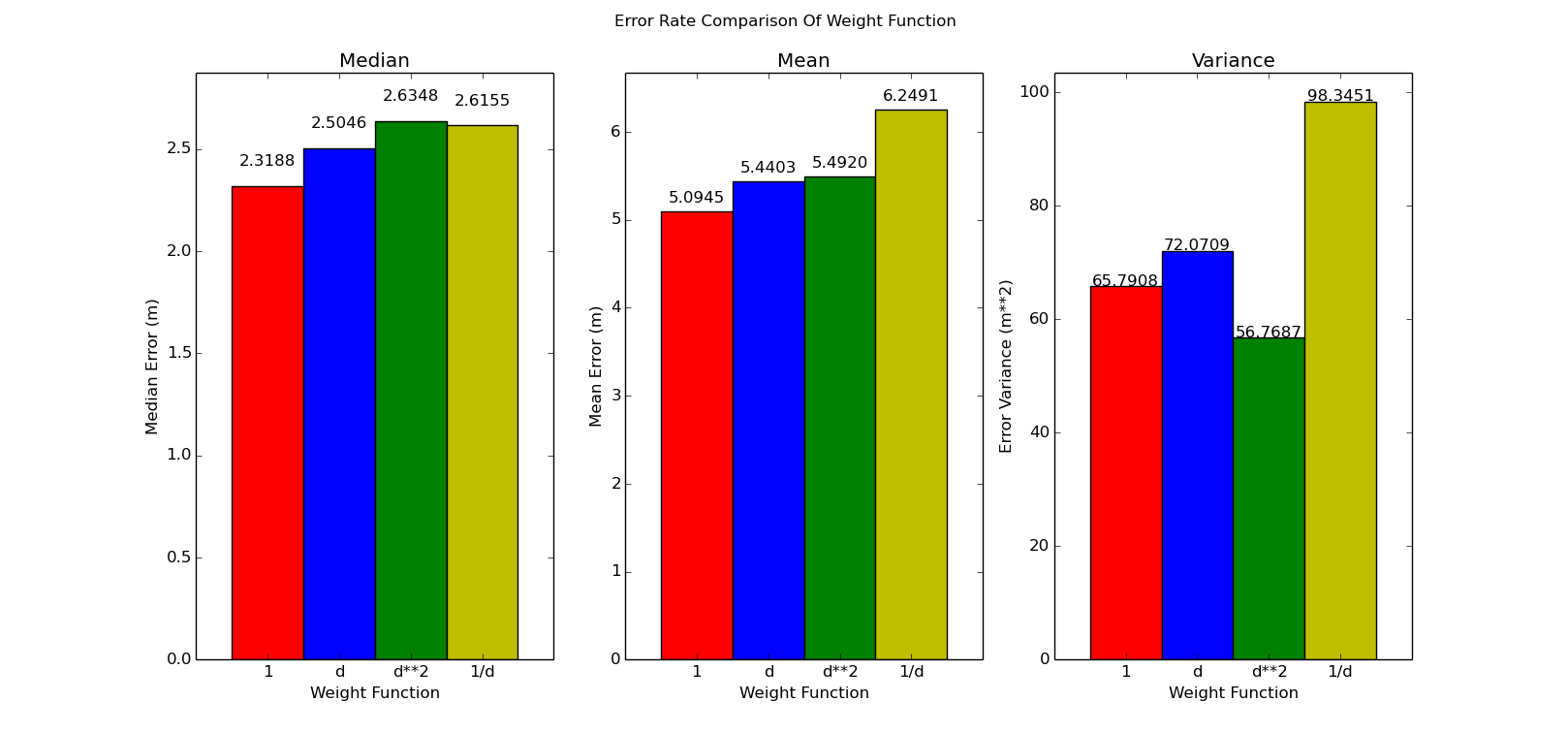
נחל במציאת פרמטר טוב עבור דחיית ערכים קיצוניים.



איור . השוואת Crop

ניתן לראות כי אכן יש השפעה ניכרת בהשמטת נתונים אשר להם ניתן cost גבוהה יותר, כאשר אם מורידים מידע רב מדי, כפי שניתן לראות בעמודות (בלוקים??) אשר משויכים להסרה לערכים אשר חורגים ביותר מחצי סטיית תקן, השגיאה עולה. תופעה זו קוראת משום שאנו מאבדים מידע שהוא חיוני. מנתונים אלו קשה להבחין האם הסרה של מה שמעברת סטיית תקן אחת, או הסרה של מה שמעבר ל-2 היא הטובה ביותר. אנו החלטנו לאמץ מצב של סטיית תקן אחת, שכן בצורה זו אנו מסירים יותר מידע, ואם נרצה לשמור את חישובי הביניים, כך נדרש לפחות זכרון.

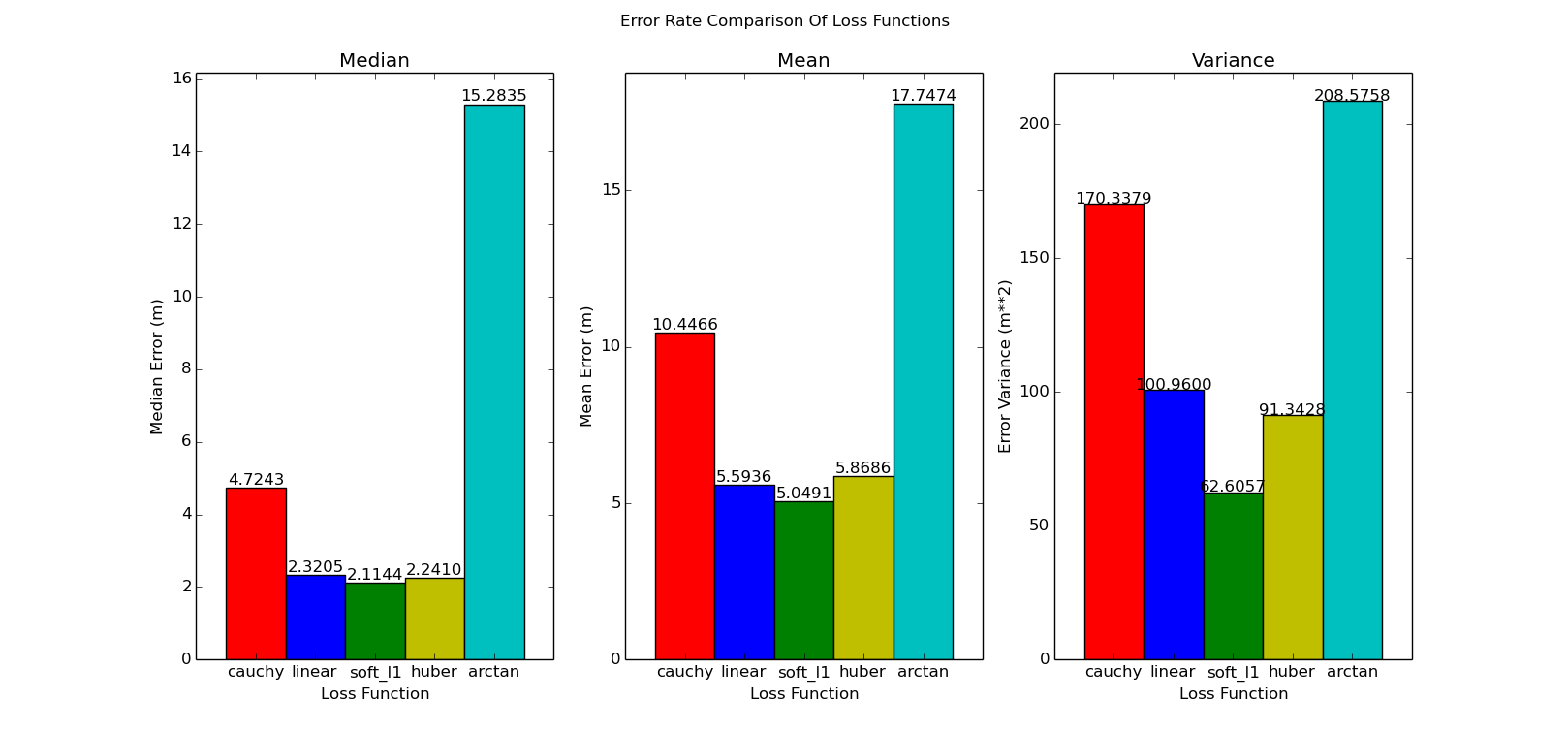
כעת נרצה לבחון האם מתן משקל שונה לכל נקודה כתלות במרחק ממנה ישפר את הביצועים. נזכיר כי משום שהמידע אודות המרחקים מחושב מעוצמת גל אלקטרומגנטי, אשר לו דעיכה בפרופורציה של , נצפה כי השגיאה כתלות במרחק תגדל באופן דומה, ולכן נצפה לשיפור כאשר ניתן לנקודות קרובות משקל גבוה יותר. כמוכן העדפנו פונקציות פשוטות על מנת לא להעלות משמעותית את זמן הריצה.



איור . השוואות פונקציית משקל

בתוצאות אנו רואים בברור כי מתן משקלים, עבור פונקציות המשקל אשר בדקנו, פוגעת בתוצאות, בניגוד לציפייה שלנו. אם זאת, ניתן לראות כי לפונקציית משקל השונות קטנה יותר משל כל פונקציה אחרת – דבר אשר כן מסתדר עם ההנחה שלנו. אנו משארים כי השיפור כאשר לא ניתן משקל נובע מכך שהשגיאה אינה תלויה באופן פשוט מהמרחק, כפי שניתן היה לראות בניתוח הרעש לשם בניית הסימולטור.

כעט נזכרו כי ה-Solver עבור בעיית non-linear least squares חשף לנו מספר פונקציות loss אפשריות, אשר מטרתן, לתת משקלים לתוצאות ביניים חריגות באופנים שונים. נרצה להשוות בין הפונקציות הללו....



איור . Loss

כפי שניתן לראות, הפתרונות Cauchy ו-arctan אשר משפיעים באופן חזק על המידע נתנו תוצאות מאוד לא טובות. פונקציית ה-loss הטובה ביותר התגלתה כ-soft\_l1.

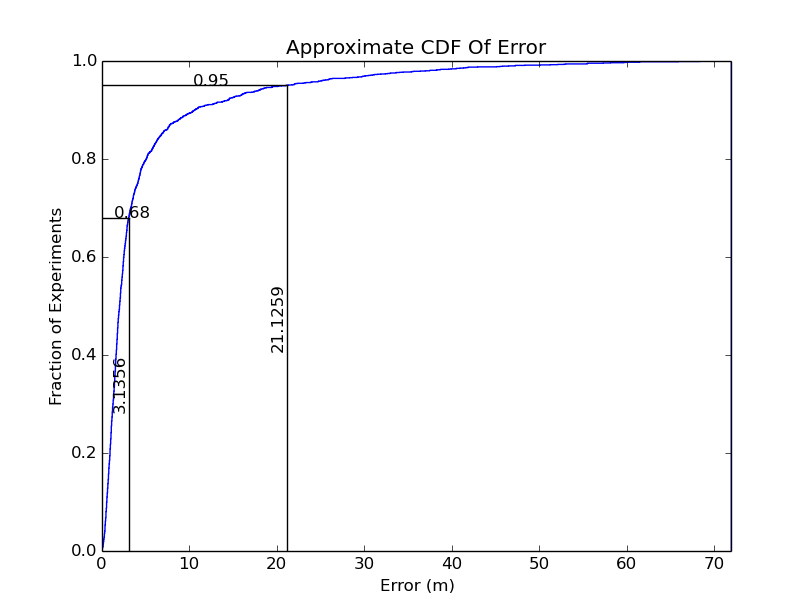
נרצה לבדוק את ההבדל בביצועים של האלגוריתם שלנו כאשר משנים פרמטרים. נחל בלבדוק מה ההשפעה הממוצעת של שוני במספר רספונדרים. נזכור כי כאשר ישנם רק 4 רספונדרים, לא ניתן להשתמש ב-lstsq כאלגוריתם ראשוני. לפיכך נשתמש ב5 רספונדרים כערך מינימלי.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מס' רספונדרים | חציון שגיאה (m) | ממוצע שגיאה (m) | מינימום שגיאה (m) | מקסימום שגיאה (m) |
| 5 | 2.26349215072 | 4.43915738774 | 0.0629513359133 | 29.858850149 |
| 7 | 2.06911039519 | 3.87349272389 | 0.373301339564 | 66.2279799046 |
| 9 | 1.70777282081 | 2.66192439132 | 0.297728939241 | 13.6342285323 |
| 11 | 1.95035023183 | 2.55589170921 | 0.0861221999582 | 17.4693812425 |
| 13 | 1.81532028198 | 2.31208109096 | 0.1097104026 | 14.520456122 |
| 15 | 1.66846366452 | 2.31427022149 | 0.137580912466 | 12.0053183404 |
| 17 | 1.67779175087 | 2.38497266588 | 0.183864562119 | 10.3954157738 |
| 19 | 1.91739197863 | 2.8213802708 | 0.188862354411 | 19.484571731 |
| 21 | 2.04041659374 | 2.63575122942 | 0.092249122304 | 17.44903197 |

טבלה . תוצאות למספר רספונדרים משתנה

לפי תוצאות הסימולציות אשר מוצגות בטבלה ‏5.3, ניתן לראות כי אכן ישנו שיפור בכל הפרמטרים כאשר מעלים את מספר הרספונדרים. יש לציין כי הוספה של פי 4 רספונדרים, מ-5 ל-20, הובילה להורדה של החציון בכחצי, ואת הממוצע הוריד אף יותר.

לבסוף כדי לאמוד את ביצועי האלגוריתם, ביצענו 200 סימולציות בעלי הפרמטרים אשר מצאנו עבור האלגוריתם, כאשר ישנם 5 רספונדרים, ובנינו CDF משוער:



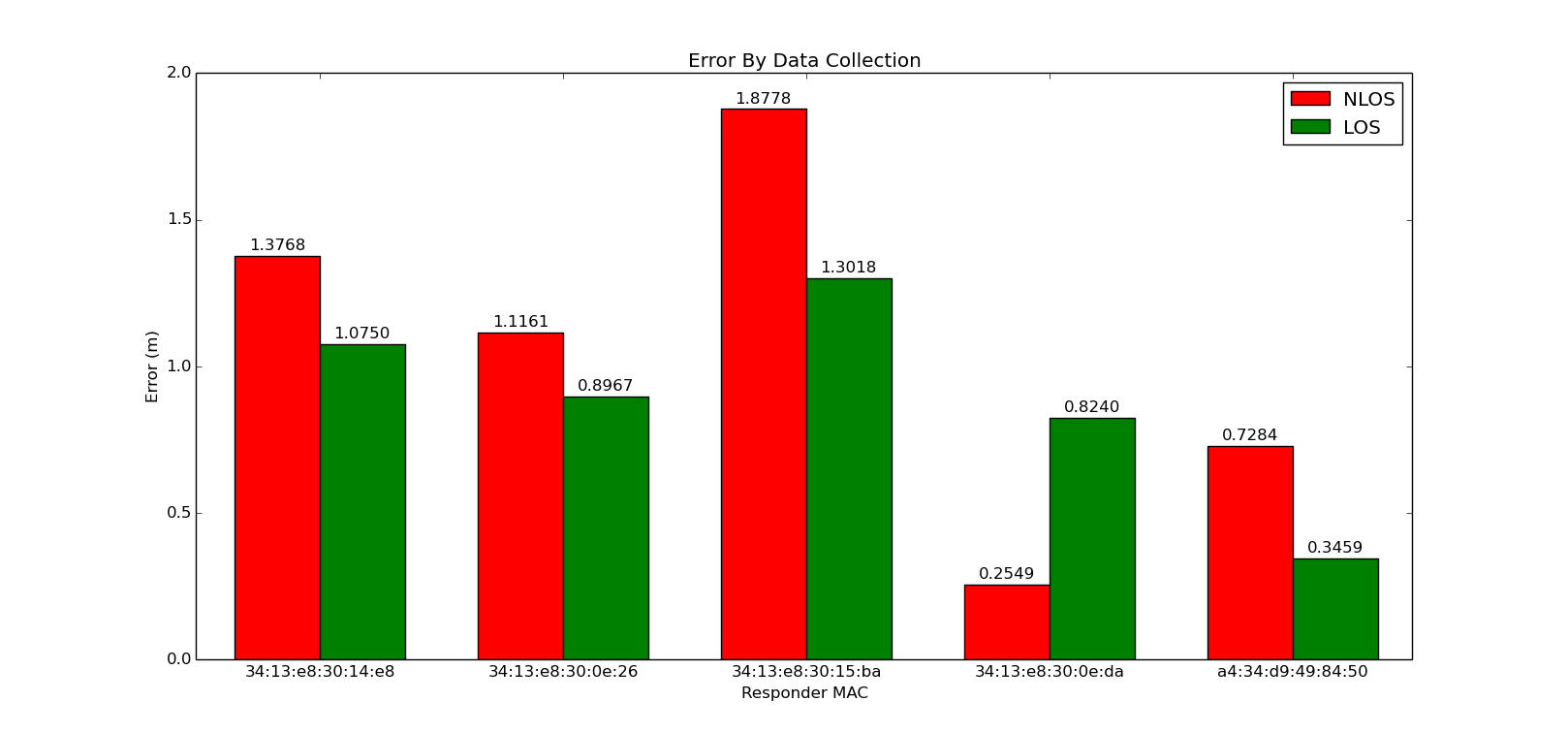
איור . CDF משוער

באיור ‏5.6 ניתן ה-CDF האמפירי מתוך הסימולציות. נקודת הסטיית תקן אחת (כ-68%) היא כ3.136 מטרים- תוצאה טובה, ונקודת 2 סטיות התקן (כ-95%) היא כ-21.1259 מטרים, ערך גבוה.

## השוואות בין תוצאות הסימולציה לנתוני אמת

### אינטל

כפי שתואר בחלק 3, קיבלנו 2 סטים של מידע מחברת אינטל אשר נמדדו במשרדי אינטל בפתח תקווה. הסט הראשון, NLOS, בוצע כאשר לא היה קו ראייה בין המשתמש המודד לרספונדרים, ובסט השני, LOS, היה קו ראייה.



איור . השוואת תוצאות אמת – אינטל

כפי שניתן לראות, כל הרספונדרים בבניין נמצאו עם שגיאה קטנה יחסית. כמצופה, ברוב המקרים התוצאה ב-NLOS מעט פחות טובה מהתוצאה ב-LOS, שכן השגיאות גדולות יותר בסט זה.

### בניין כיתות

# סיכום, מסקנות והצעות להמשך

מקורות

|  |  |
| --- | --- |
| [1] | W. Hereman and W. S. Murphy, "Determination of a Position in Three Dimentions Using Trilateration and Approximate Distances," *Decision Sciences,* 1995. |
| [2] | G. Strang, Linear Algebra and Its Applications, Cengage Learning, 2006. |
| [3] | [Online]. Available: https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html. |
| [4] | [Online]. Available: https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.linalg.svd.html. |