|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | TAULOGOblue%201 | |  |
| אוניברסיטת תל-אביב | | **Tel-Aviv University** |
| הפקולטה להנדסה  בי"ס להנדסת חשמל | |  | | Faculty of Engineering  School of Electrical Engineering |
| Indoor Fingerprinting Geolocation | | | | |
| פרויקט מס' 16-1-1-1197  דו"ח סיכום | | | | |
| מבצעים: | | | | |
|  | שם: ברוך מקרו | | ת.ז. 305420291 | |
|  | שם: תומר אמינוף | | ת.ז. 313585069 | |
| מנחים: | | | | |
|  | ד"ר עופר בר שלום | |  | |
| מקום ביצוע הפרויקט: חברת "אינטל" – פתח תקווה | | | | |

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

שירותים מבוססי מיקום הפכו לדבר שבשגרה ומידע אודות המיקום המדויק שלך הפך ללא ספק לנכס חשוב של כל מכשיר אלקטרוני. בעוד שבמרחבים פתוחים מרבית המכשירים מוצאים את המיקום בעזרת ( [global positioning systems](https://en.wikipedia.org/wiki/Global_positioning_system))GPS, בתוך מבנים המכשירים מאבדים את אות הGPS ולמעשה גם את המיקום.

כפתרון לבעיה זו בחברת אינטל מתבצע פרויקט indoor positioning שמטרתו לאפשר ניווט בתוך מבנים (כגון קניונים, משרדים, קמפוסים וכו'). במסגרת פרויקט זה פיתחה החברה ריספונדרים (רכיבי חומרה מבוססי WIFI) אשר מטרתם לתקשר עם מכשיר הלקוח, ובעזרת תשדורות מהם יוכל להסיק את מיקומו במבנה. על מנת לעשות זאת, על הלקוח לבצע חישוביים מתמטיים רבים אשר בין היתר לוקחים זמן ומשאבים יקרים מהלקוח. בפרויקט זה אנו באים להקל על בעיה זו באמצעות בניית מסד נתונים (database) ממדידות מוקדמות של נתונים כגון RSSI ומרחק מהריספונדרים( בעזרת טכנולוגיית Fine Timing Measurements) במבנה הסגור בו אנו רוצים לנווט, נתונים אלה יקראו טביעות אצבע(fingerprints). באמצעות אלגוריתם מתאים נוכל לשערך את מיקום המשתמש באמצעות נתוני התשדורות שהמשתמש מקבל מהריספונדרים וטביעות האצבע ממסד הנתונים. בפרויקט זה נבדוק את הביצועים של שלושה אלגוריתמים שכתבנו אשר מנצלים חלקים אחרים של טביעות האצבע. אלגוריתם 1 משתמש רק בנתוני הRSSI, אלגוריתם 2 משתמש הן בנתוני הRSSI והן בנתוני המרחק מהריספונדרים ואלגוריתם 3 משתמש רק במרחקים מהריספונדרים (Trilateration).

דיאגרמה כללית של הפרויקט:



# הקדמה

ניווט תוך-מבני מדויק הופך חשוב יותר עבור מכשירים מבוססים Wi-Fi בשל השימוש המוגבר של מציאות רבודה, רשתות חברתיות, ניטור בריאות, מעקב אישי, בקרת מלאי ויישומי מיקום-מודע תוך-מבני אחרים.

פריסה בקנה מידה גדול של ניווט תוך-מבני היא הרבה יותר קשה בגלל שני אתגרים טכניים. ראשית, GPS לא ניתן לפריסה לצורך שימוש תוך-מבני מכיוון שאותות GPS לא יכולים להגיע למקלטים בתוך המבנה. שנית וחשוב יותר, בהתאם לסביבות תוך-מבני מסובכות כגון גיאומטריות בניין, תנועת אנשים, וההשפעות האקראיות של התפשטות אותות, הפרעות ורעש ממכשירים אחרים גם יכולים לפגוע בדיוק של המיקום. הפופולריות והמחיר נמוך של כרטיסי ממשק רשת ה- Wi-Fi הוא תמריץ אטרקטיבי להשתמש ב- Wi-Fi כבסיס מערכת מיקום. מכיוון שבמבנים רבים כבר קיימים רכיבי WIFI והמכשירים הסלולריים יכולים לקבל את ערך הRSSI מהם, שיערוך מיקום באמצעות RSSI/טביעות אצבע של RSSI יכול להיות פתרון זמין, רחב היקף וזול מכיוון שלא דורש חומרה מיוחדת שלא מותקנת כבר. הריספונדרים של אינטל מספקים נתונים נוספים כגון מרחק בין הריספונדר למכשיר ולכן ניתן באמצעות חישוב trilateration למצוא את המיקום, אולם חישוב זה עולה בזמן ריצה.

על כן, מטרת הפרויקט היא שיערוך המיקום של המשתמש בתוך המבנה על ידי שלושה אלגוריתמים לשערוך מיקום, והשוואה בין ביצועיהם.

לאחר שהבנו את המוטיבציה לפרויקט התחלנו בסקירה ספרותית אודות מחקרים קיימים בנושא. במהלכה מצאנו דרכים לשערוך מיקום באמצעות טביעות אצבע של RSSI(אלגוריתם 1), ואת אלגוריתם הtrilateration (אלגוריתם 3). את אלגוריתם 2 המשלב הן טביעות אצבע של RSSI והן את נתוני המרחק מהריספונדרים כתבנו בעצמנו.

# רקע תיאורטי

הבעיה של ניווט תוך-מבני מבוסס Wi-Fi מורכבת בקביעת המיקום של מכשיר הלקוח ביחס לריספונדרים. קיימות מגוון שיטות המשיגות זאת, בפרויקט שלנו נדון בשתיים עיקריות ובשילוב שלהן:

* טכניקה למיקום מבוססת על אינדיקציה עוצמת האות הנקלט (RSSI) והוצאת המיקום המתאים ביותר מתוך מסד נתונים של טביעות אצבע שנמדד מראש.
* טכניקה למיקום מבוססת על מרחק/זמן המעוף (TOF), בשיטה זו המיקום נמצא באמצעות trilateration.

## קואורדינטות גאוגרפיות

קואורדינטות גאוגרפיות הן [נקודת ציון](https://he.wikipedia.org/wiki/%D7%A0%D7%A7%D7%95%D7%93%D7%AA_%D7%A6%D7%99%D7%95%D7%9F) לזיהוי מדויק של מיקום על פני [כדור הארץ](https://he.wikipedia.org/wiki/%D7%9B%D7%93%D7%95%D7%A8_%D7%94%D7%90%D7%A8%D7%A5), על פי [קו האורך](https://he.wikipedia.org/wiki/%D7%A7%D7%95_%D7%90%D7%95%D7%A8%D7%9A) ו[קו הרוחב](https://he.wikipedia.org/wiki/%D7%A7%D7%95_%D7%A8%D7%95%D7%97%D7%91), ביחידות של [מעלות](https://he.wikipedia.org/wiki/%D7%9E%D7%A2%D7%9C%D7%94_(%D7%96%D7%95%D7%95%D7%99%D7%AA)).

קואורדינטות גאוגרפיות נכתבות כצמד של [מספרים](https://he.wikipedia.org/wiki/%D7%9E%D7%A1%D7%A4%D7%A8): (lat, lon) כאשרlat מציין את קו הרוחב ו -lon מציין את קו האורך.

קו הרוחב (latitude) נמדד כלפי [קו המשווה](https://he.wikipedia.org/wiki/%D7%A7%D7%95_%D7%94%D7%9E%D7%A9%D7%95%D7%95%D7%94), שמהווה את האפס של קווי הרוחב. צפונה מקו המשווה - קווי הרוחב הם בעלי ערך חיובי, דרומה ממנו - בעלי ערך שלילי. ניתן גם לסמן את קווי הרוחב עם ערכם האבסולוטי (כלומר, ללא סימן פלוס/מינוס) בציון צפון/דרום. ערכי הקצה הם: 90 מעלות (צפון) ב[קוטב הצפוני](https://he.wikipedia.org/wiki/%D7%94%D7%A7%D7%95%D7%98%D7%91_%D7%94%D7%A6%D7%A4%D7%95%D7%A0%D7%99), 90- מעלות (או 90 מעלות דרום) ב[קוטב הדרומי](https://he.wikipedia.org/wiki/%D7%94%D7%A7%D7%95%D7%98%D7%91_%D7%94%D7%93%D7%A8%D7%95%D7%9E%D7%99).

קו האורך (longitude) נמדד כלפי [קו גריניץ'](https://he.wikipedia.org/wiki/%D7%A7%D7%95_%D7%92%D7%A8%D7%99%D7%A0%D7%99%D7%A5%27) שמהווה את האפס של קווי האורך. מזרחה מקו גריניץ' קווי האורך הם בעלי ערך חיובי, מערבה ממנו בעלי ערך שלילי. ערכי הקצה הם: 180 מעלות (מזרח), 180- מעלות (מערב) - שני קווים אלו הם בעצם קו אחד העובר בלב [האוקיינוס השקט](https://he.wikipedia.org/wiki/%D7%94%D7%90%D7%95%D7%A7%D7%99%D7%99%D7%A0%D7%95%D7%A1_%D7%94%D7%A9%D7%A7%D7%98).

בפרויקט אנו נעבוד עם קואורדינטות אלה בצורה עשרונית כך שלדוגמה המיקום של בנין כיתות הוא במיקום (34.8047 , 32.1087).

קיים גם קו גובה הנקרא altitude, והוא מסמן את גובה במטרים של המיקום ביחס לגובה פני הים.

בפרויקט שלנו אנו מניחים גובה קבוע ולכן נדבר על המיקום (lat, lon) ולא (lat, lon, alt).

ניתן לעבור ממערכת קואורדינטות גאוגרפיות למערכת קואורדינטות קרטזית יחסית. אלגוריתם ההמרה ממוש אצלנו במודולutilities.py בפונקציה lla2ecef וההמרה חזרה לקואורדינטות גאוגרפיות מומש בפונקציה ecef2lla.

## RSSI- Received signal strength indication

<http://www.metageek.com/training/resources/understanding-rssi.html>

RSS היא עוצמת האות הנקלט במכשיר בעוד RSSI הוא אינדקטור של עוצמת האות הנקלט במכשיר. RSS נמדד ביחידות [dbm] ונע בין הערכים 0 (עוצמת אות מצוינת) לבין 120- (עוצמת אות מאד חלשה עד כדי לא קיימת) ולכן גם עוצמת האות לא יורדת באופן לינארי ככל שהמרחק מהמשדר גדל.

עוצמת האות מושפעת מגורמים נוספים רבים כגון גודל האנטנות של המכשירים המשדרים והקולטים, מספר הקירות, החפצים, צורת המבנה ומספר האנשים בעת המדידה ולכן השונות שלו גדולה.

## Trilateration

Trilateration הוא תהליך של קביעת מיקום (יחסי או אבסולוטי) בעזרת מדידות של מרחקים ממיקומים הידועים במרחב באמצעות חישובים גיאומטריים של מעגלים וכדורים. תהליך זה נפוץ מאד משום שהוא מרכיב חשוב בתהליך קביעת המיקום במערכות מבוססות GPS (בנוסף לתהליכים דומים כדוגמת  [Triangulation](https://en.wikipedia.org/wiki/Triangulation_(surveying))שבו מחושב המיקום באמצעות מדידות זוויות ממיקומים ידועים).

על מנת לפשט את ההסבר נסביר את התהליך במרחב דו מימדי (נניח כי הגובה נשאר קבוע). כאשר ידוע לנו מרחק ­, מנקודת ייחוס אחת, אנו יודעים שאנו נמצאים על המעגל במרחק ממנה(המעגל הכחול באיור 1-2). כעת בהינתן גם המרחק מהנקודה הירוקה צמצמנו את המיקום שלנו ל2 אפשרויות בלבד. בהינתן גם אנו יודעים בוודאות את מיקומנו.

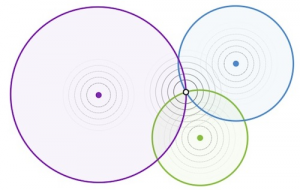
באופן יותר פורמלי: בהינתן אוסף נקודות אנו רוצים למצוא נקודה כך ש:

לכל כאשר המרחק(רדיוס) הנמדד.

הבעיה אינה פתירה כאשר יש פחות מ-3 נקודות, משום שבמקרה זה ישנם יותר נעלמים מאשר משוואות בבעיה. בהינתן 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. בפרויקט הנחנו z קבוע ולכן למעשה ניתן לראות את הבעיה ב-2 מימדים ובעזרת מעגלים ולא כדורים (ספרות).

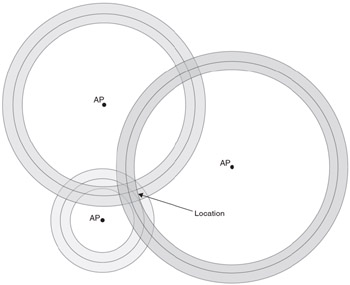


איור ‑ טרילטרציה בסיסית

ניתן לראות באיור מיקום לא ידוע (הנקודה הלבנה), ו-3 מיקומים ידועים במרחב. באמצעות המרחק מ-3 הנקודות ניתן לשרטט מגדל סביב כל נקודה, והנקודה בהתלכדות של 3 המעגלים היא המיקום.

### בעיית ה-Co-Linearity

במציאות בשל שגיאות מדידה המצב יותר דומה לאיור הבא:

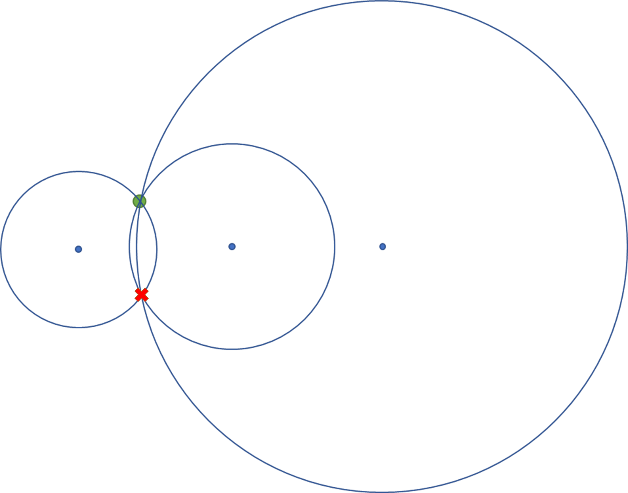


איור ‑ טרילטרציה עם שגיאות מדידה

באיור ‏2.2 ניתן לראות כי לא ניתן לקבוע באופן פשוט, בשל שגיאות המדידה איפה בחתך של 3 הקליפות העבות המיקום.88888888888888

### 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

אלגוריתם 1 מקבל כקלט מסד נתונים של טביעות אצבע ומדידות של המשתמש, עבור כל טביעת אצבע במסד האלגוריתם מחושב המרחק האוקלידי בעזרת הנוסחה הבא:

*כאשר:*

* *: וקטור מדידות ה של המשתמש מכל ריספונדר.*
* *: וקטור נתוני ה של טביעות אצבע הבמסד.*
* *: מספר הריספונדרים.*
* *: הריספונדר הj .*
* *: לחישוב המרחק האוקלידי נבחר .*

*לאחר חישוב המרחקים הנ"ל, האלגוריתם מחזיר את המיקום של הטביעות אצבע עם המרחק המינימלי.*

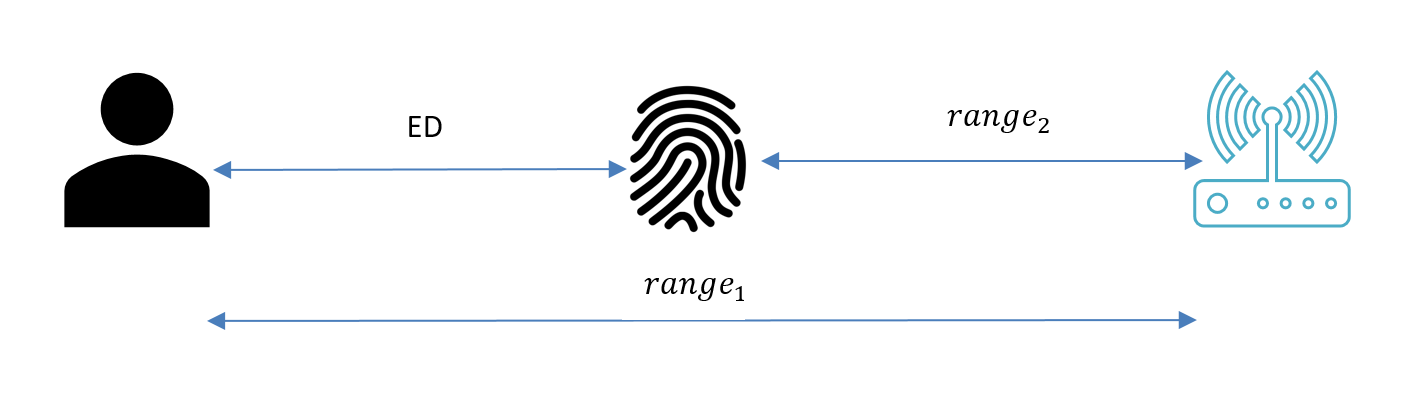
## אלגוריתם 2

אלגוריתם 2 מקבל כקלט מסד נתונים של טביעות אצבע ומדידות של המשתמש ובנוסף מספר מדידות מרחק מהריספונדרים(לרוב 3 מדידות ולכל היותר 2 מדידות מרחק).

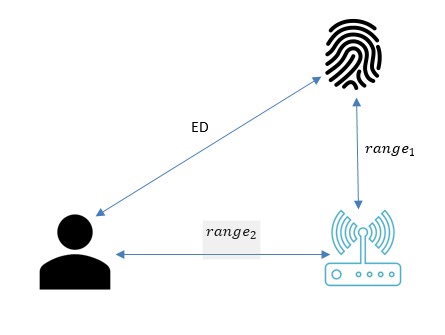
אלגוריתם 2 הוא למעשה שיפור של אלגוריתם 1 המתבסס רק על מדידות הRSSI. בדומה לאלגוריתם 1 אלגוריתם זה מודד את כל המרחקים האוקלידים מטביעות האצבע במסד.

אולם, לאחר קבלת מרחקים אלו, האלגוריתם מתייחס גם למדידות המרחק (הRange) הן במסד הנתונים והן בנתוני המשתמש ובודק שתנאים גיאומטריים אכן מתקיימים עליהם:

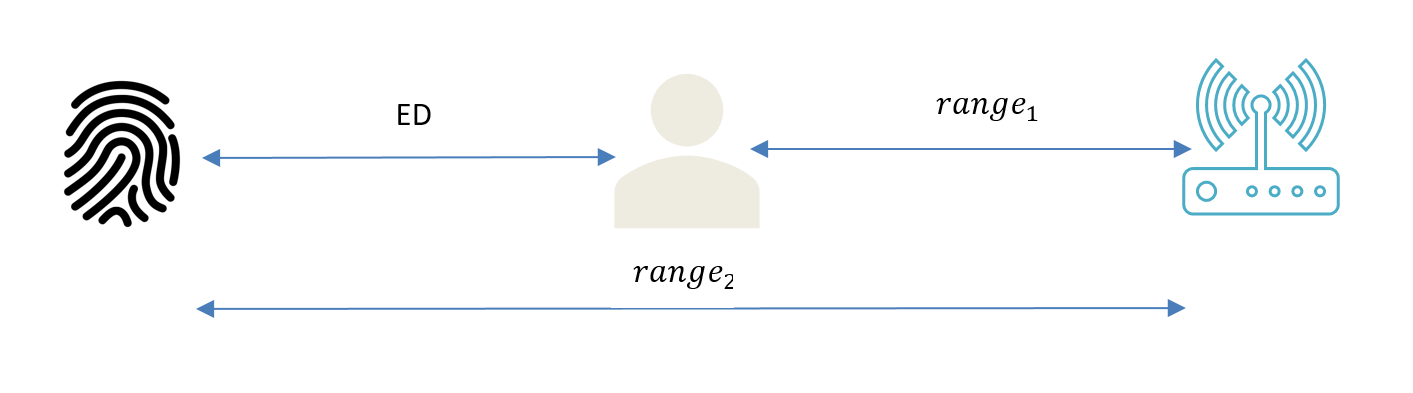
* ED: מרחק האוקלידי שחושב בין המשתמש לטביעת האצבע.
* : מרחק בין המשתמש לריספונדר שניתן כחלק מהקלט לאלגוריתם 2.
* : מרחק בין טביעת האצבע לריספונדר, מרחק זה זמין במסד הנתונים.



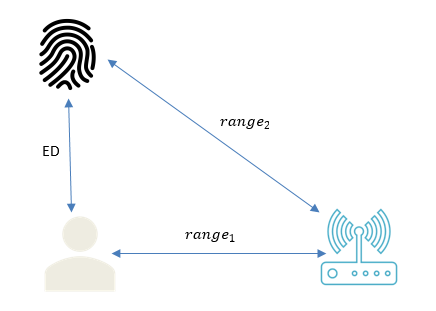
ממקרה זה בנינו את התנאי:



ממשפט בגיאומטריה כי הפרש שתי צלעות במשולש קטן מאורך הצלע השלישית נקבל את התנאי הבא:



ממקרה זה בנינו את התנאי:



ממשפט בגיאומטריה כי סכום שתי צלעות במשולש גדול מאורך הצלע השלישית נקבל את התנאי הבא:

סה"כ מ4 מקרים אלו נקבל את התנאים הבאים:

נסדר את טביעות האצבע במערך לפי המרחק האוקלידי שקבלנו כאשר באינדקס 0 יהיה טביעת האצבע עם הערך המינימלי, בכל איטרציה נבדוק אם התנאים מתקיימים אם כן נחזיר את מיקום טביעת האצבע הרלוונטית אחרת נעבור לטביעת האצבע הבאה.

במקרה ונעבור על כל טביעות האצבע ולא נמצא טביעת אצבע המקיימת את התנאים הנ"ל נחזיר את המיקום של טביעת האצבע באינדקס 0 (עם המרחק המינימלי) כלומר לא יהיה שינוי בין הפלט של אלגוריתם 1 ולפלט של אלגוריתם 2 במקרה זה.

אלגוריתם זה למעשה מסנן בחירה של טביעות אצבע אשר נבחרו בטעות, בגלל מדידת RSSI לא מדויקת או במסד הנתונים או של המשתמש, ומחזיר רק מיקומים אשר עומדים בתנאים גיאומטריים בסיסים.

## אלגוריתם 3

אלגוריתם 3 מקבל כקלט את מיקום כל הריספונדרים ומדידות מרחק של המשתמש מהריספונדרים(3 מדידות ומעלה).

בהינתן מיקומי הריספונדרים ומדידות המרחק מחשבים את מיקום המשתמש ב trilateration(ראה סעיף 2.3), אנו משתמשים בחישוב מטריציוני לפתרון מערכת משוואות.

*אופטימזציה שהכנסנו לאלגוריתם:*

* בדיקת הפלט: מכיוון שה trilaterationיכול לתת מיקומים שגויים(שגיאות מדידות או בעיית Co-Linearity שתיארנו בסעיף 2.3), לאחר שאנו מקבלים את המיקום() אנו מחשבים ידנית את מרחק מיקום זה מהריספונדרים הרלוונטיים(להם קבלנו את המרחק מהמשתמש כקלט) ומשווים עם מדידות המרחק שקבלנו כקלט, בנוסף מחשבים את השגיאה(נסמן שגיאה זו ב).
* במקרה בו אנו מקבלים יותר מ3 מדידות אנו מחלקים אותן לקבוצות שונות בגודל 3 ויותר.

לדוגמא עבור 4 מדידות מרחק אנו מחלקים ל5 קבוצות:

1. .
2. .
3. .
4. .
5. .

שולחים כל קבוצה ל trilateration, לאחר מעבר על כל קבוצות אלו אנו מחזירים את המיקום עם השגיאה, , המינימלית.

# מימוש

למעשה הפרויקט הוא סימולציה של ניווט בlive. לצורך כך התוכנית מקבלת 3 קבצים:

1. קובץ עם הריספונדרים הרלוונטיים ומיקומם בקומה.
2. קובץ הDatabase : מכיל את טביעות האצבע אשר ממפות את הקומה. שורה בטביעת אצבע היא מהצורה (MAC, RSSI[db], Latitude, Longitude, Altitude ,Range[cm]) וטביעת אצבע היא שורה לכל ריספונדר בקובץ ריספונדורים, כאשר:

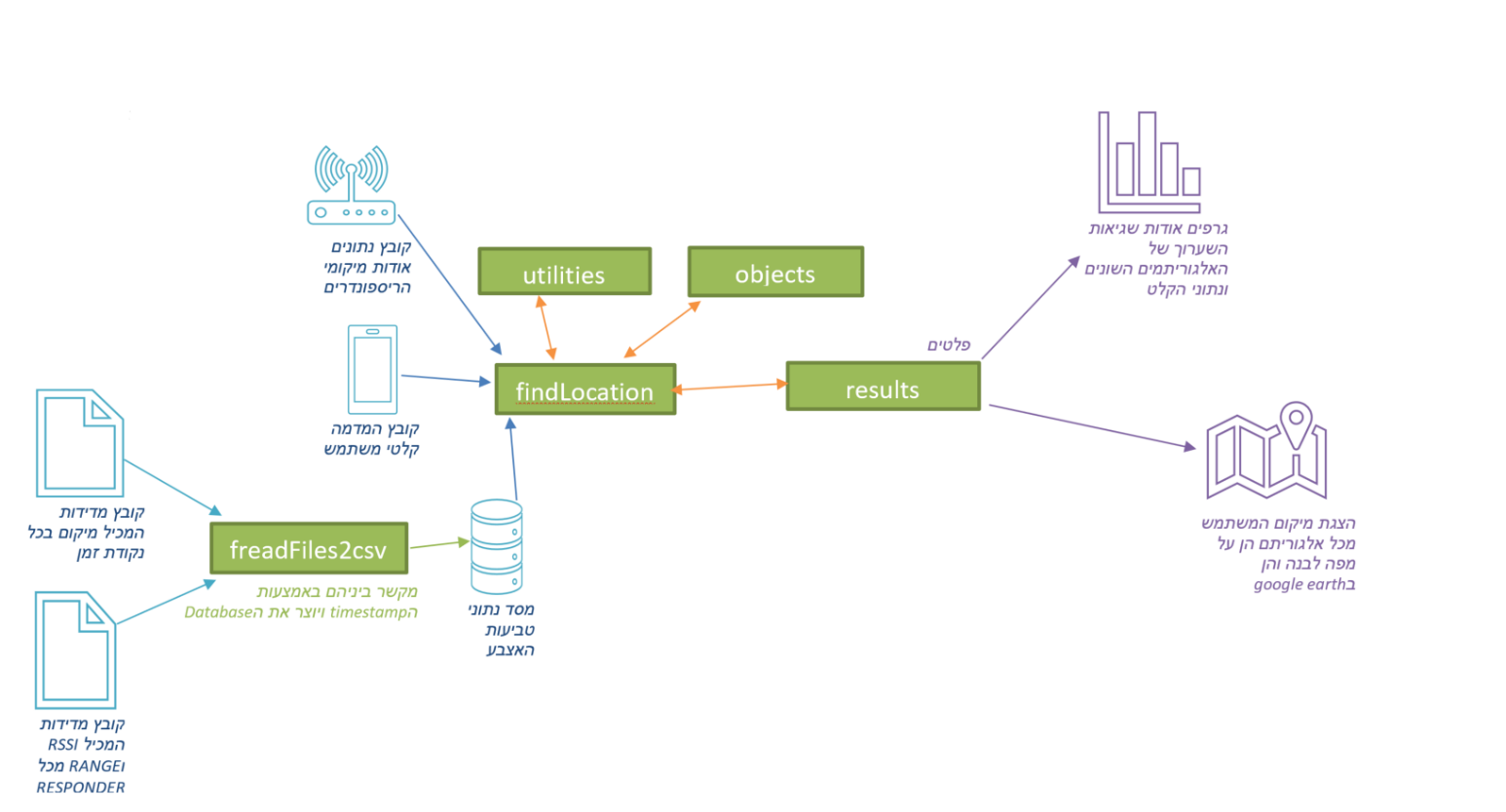
* MAC: Media Access Control address של הריספונדר, זה למעשה המזהה שלו.
* :RSSI זה הוא ערך הRSSI , כאשר המדידה לא הצליחה עבור הריספונדר יהיה ערכו -120[db]
* Latitude: ערך קו האורך במדידה.
* Longitude: ערך קו הרוחב במדידה
* Altitude: קו הגובה במדידה
* :Range המרחק מהריספונדר בסנטימטרים, כאשר המדידה לא הצליחה ערכו יהיה 0.

1. קובץ משתמש: קובץ המדמה את הנתונים שמשתמש מעביר תוך כדי ניווט. הקובץ מכיל נתוני RSSI ו-range לכל ריספונדר בקומה (אם לא הצליח להשיג נתונים אלה עבור ריספונדר מסוים הם יהיו [cm]0 ו- -120[dbm] בהתאמה), ובנוסף גם את המיקום האמיתי שכל מדידה בוצעה בו וזאת על מנת להשוואות בין שיערוך המיקום לבין המיקום האמיתי שהיה בו המשתמש. סך כל "מדידה" של המשתמש בקובץ נראה כמו שורה בקובץ Database.

כל המדידות בוצעו בבניין כיתות להנדסת חשמל, קומה שניה, על ידי צוות הוולידציה של אינטל.

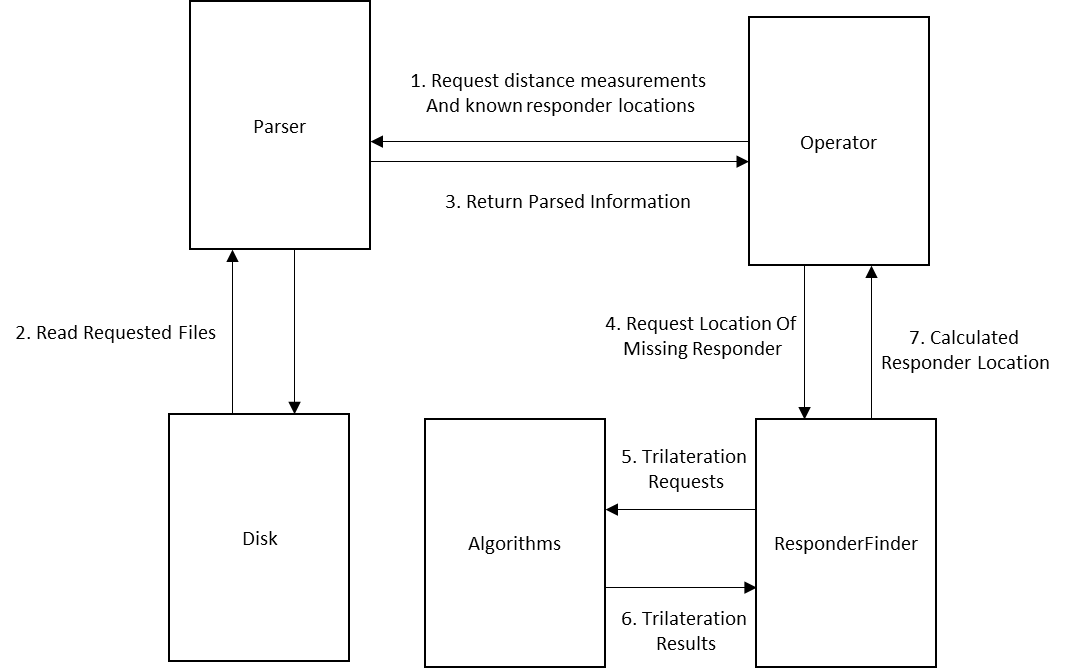
מהמדידות נוצרו שני קבצים- קובץ המכיל את תוצאות המדידות(כמו ערכי RSSI וRange לכל ריספונדר) וקובץ המכיל את המיקומים שבהן בוצעה כל מדידה. לכן תחילה קישרנו בין הקבצים באמצעות שדה הtimestamp ויצרנו קובץ אחד המכיל הן את נתוני המדידות והן את המיקום של כל מדידה.

על מנת ליצור את קבצי המשתמש וה-Database חילקנו את קובץ המדידות לשני קבצים זרים. כך שאחד יהיה הDatabase והשני קובץ המשתמש כמתוארים לעיל. פעולה זו מתבצעת במודול readFiles2csv כפי שניתן לראות באיור \_\_\_\_\_ במסגרת ה\_\_\_\_\_



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:

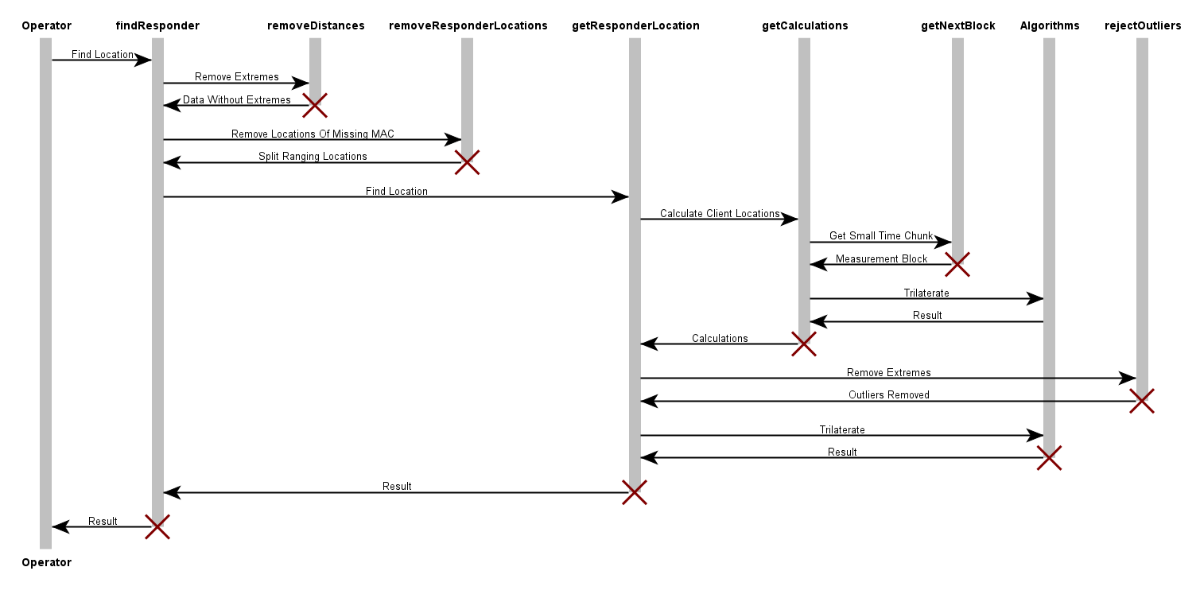
* קובץ ranging – מכיל מרחקים מכל הרספונדרים לאורך זמן
* קובץ responder locations – מכיל מיקומים ידועים של הרספונדרים במרחב (ביחידות lat/long)
* קובץ reference (אופציונלי) – מכיל מיקומים אמיתיים של הלקוח לאורך זמן (ביחידות lat/long)

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

## ResponderFinder

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

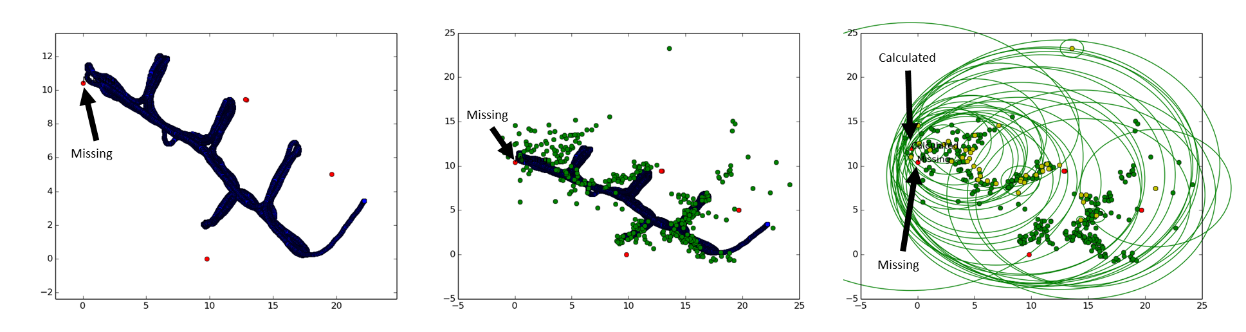
ניתן לראות את רצף ההרצה של המודול באיור הנ"ל:



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- (Anon., n.d.). 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- (Strang, 2006). בשאר מאמר זה נכנה את שיטה זו svd. פירוט על האלגוריתם ניתן למצוא ב- (Anon., n.d.).

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 :....

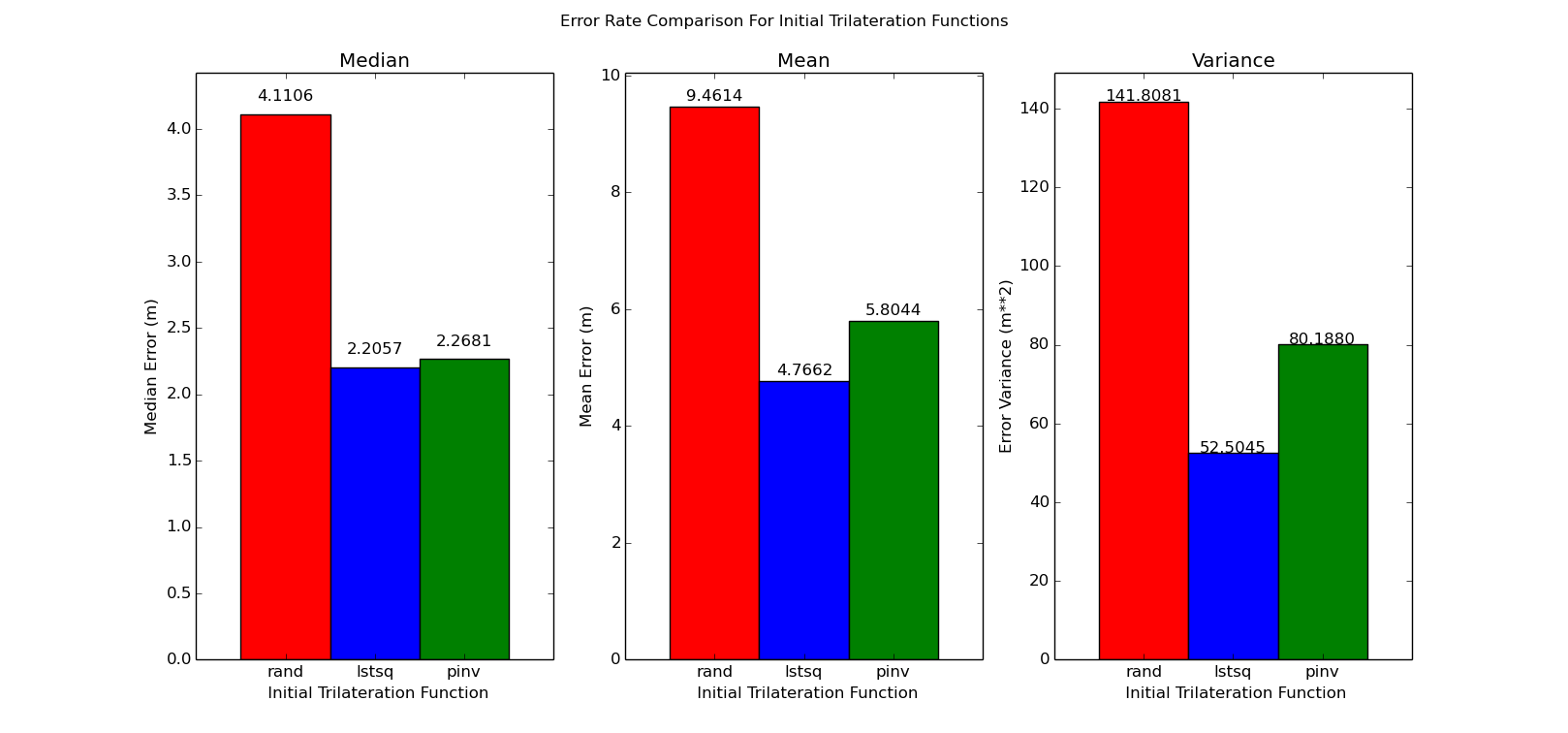
ערכים אלו נבחרו משום שמניסויים בזמן הפיתוח גילינו כי עבור מסלול ארוך, הערכים מתכנסים (??), וכן את המהירות הממוצעת חישבנו ע"י ניתוח קבצי הידע אשר סופקו לנו ע"י חברת אינטל (LOS, NLOS???)... כמוכן, בניסויים הללו לא ביצענו Crowd-Sourcing כלל, על מנל לאמוד את האלגוריתם הבסיסי. את ההשפעה של שיטות ה-Crowdsourcing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

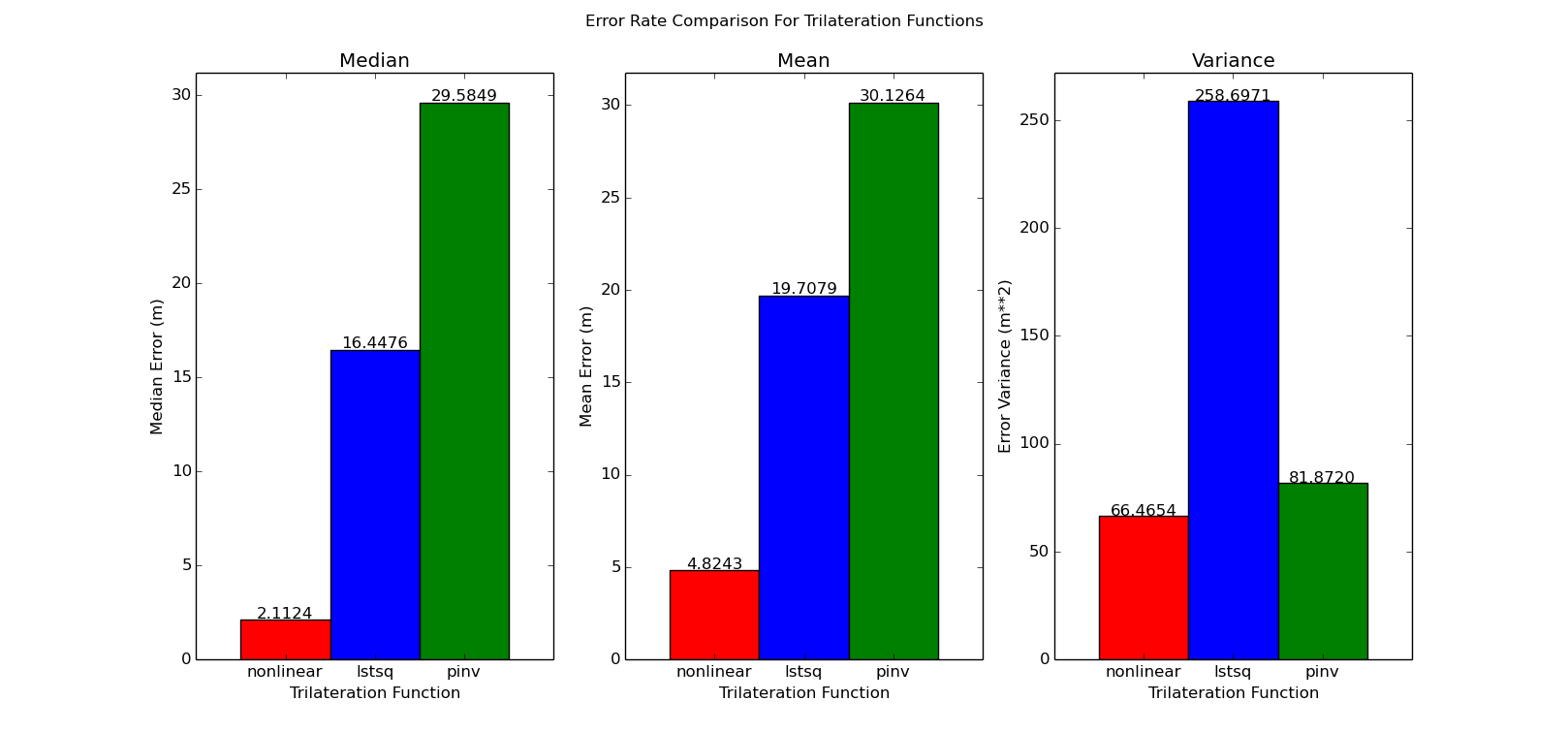
תחילה, נזכיר כי, כפי שפורט בחלק 3, עבור אלגוריתם 2 אנו נדרשים לספק נקודה התחלתית כלשהי. נרצה לאמוד איזו מן השיטות נותת ביצועים טובים יותר.



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.



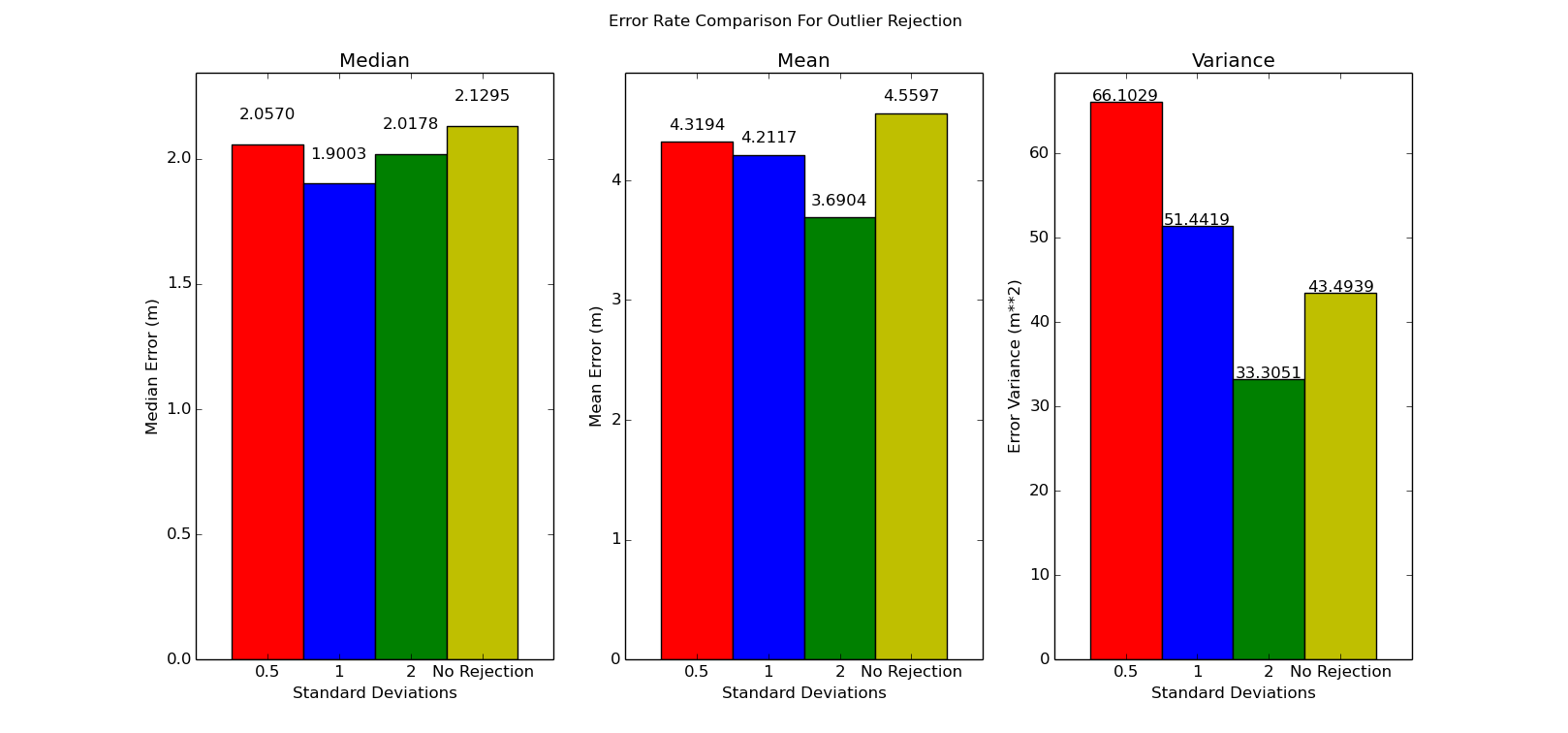
איור ‑ השוואה בין אלגוריתמים

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.

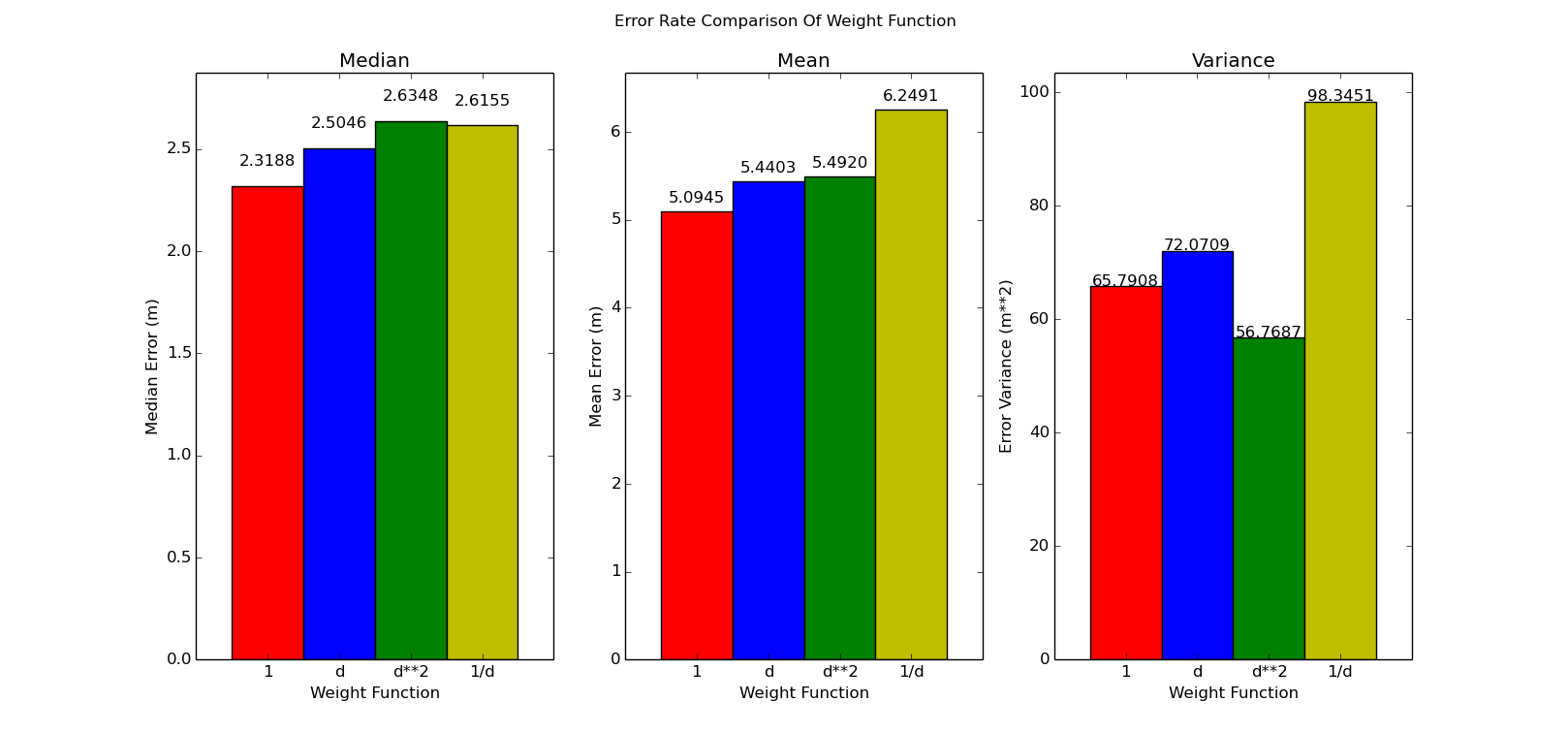
נחל במציאת פרמטר טוב עבור דחיית ערכים קיצוניים.



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.

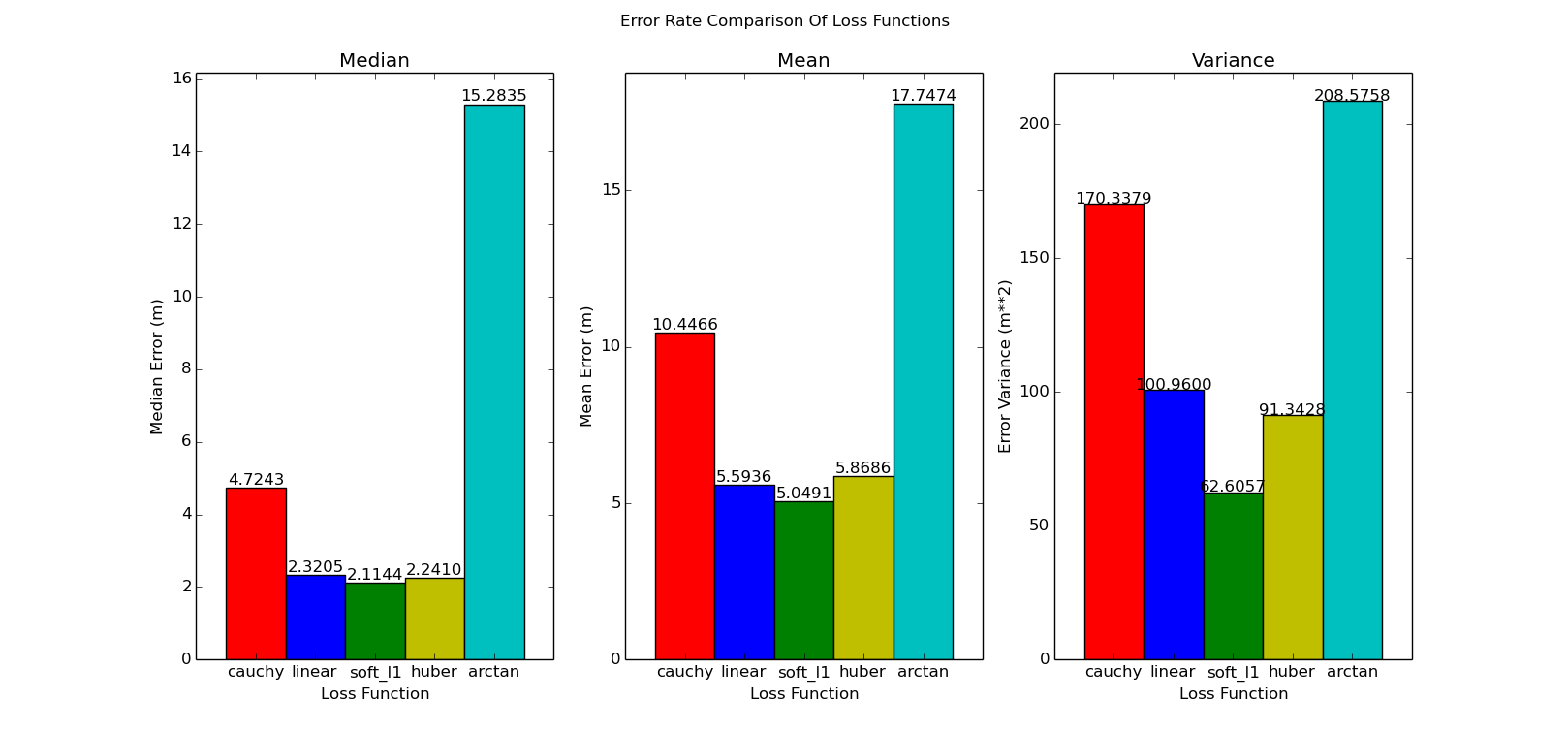
כעת נרצה לבחון האם מתן משקל שונה לכל נקודה כתלות במרחק ממנה ישפר את הביצועים. נזכיר כי משום שהמידע אודות המרחקים מחושב מעוצמת גל אלקטרומגנטי, אשר לו דעיכה בפרופורציה של , נצפה כי השגיאה כתלות במרחק תגדל באופן דומה, ולכן נצפה לשיפור כאשר ניתן לנקודות קרובות משקל גבוה יותר. כמוכן העדפנו פונקציות פשוטות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

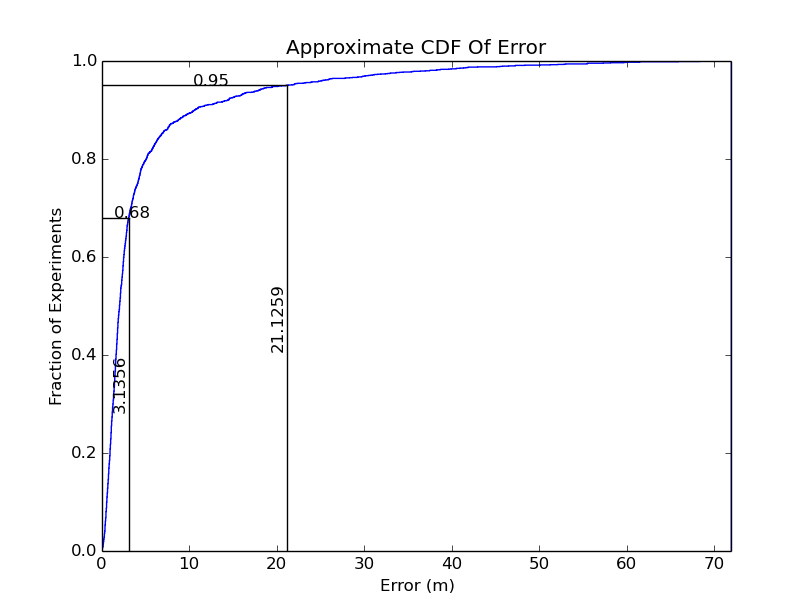
נרצה לבדוק את ההבדל בביצועים של האלגוריתם שלנו כאשר משנים פרמטרים. נחל בלבדוק מה ההשפעה הממוצעת של שוני במספר רספונדרים. נזכור כי כאשר ישנם רק 4 רספונדרים, לא ניתן להשתמש ב-lstsq כאלגוריתם ראשוני. לפיכך נשתמש ב5 רספונדרים כערך מינימלי.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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:



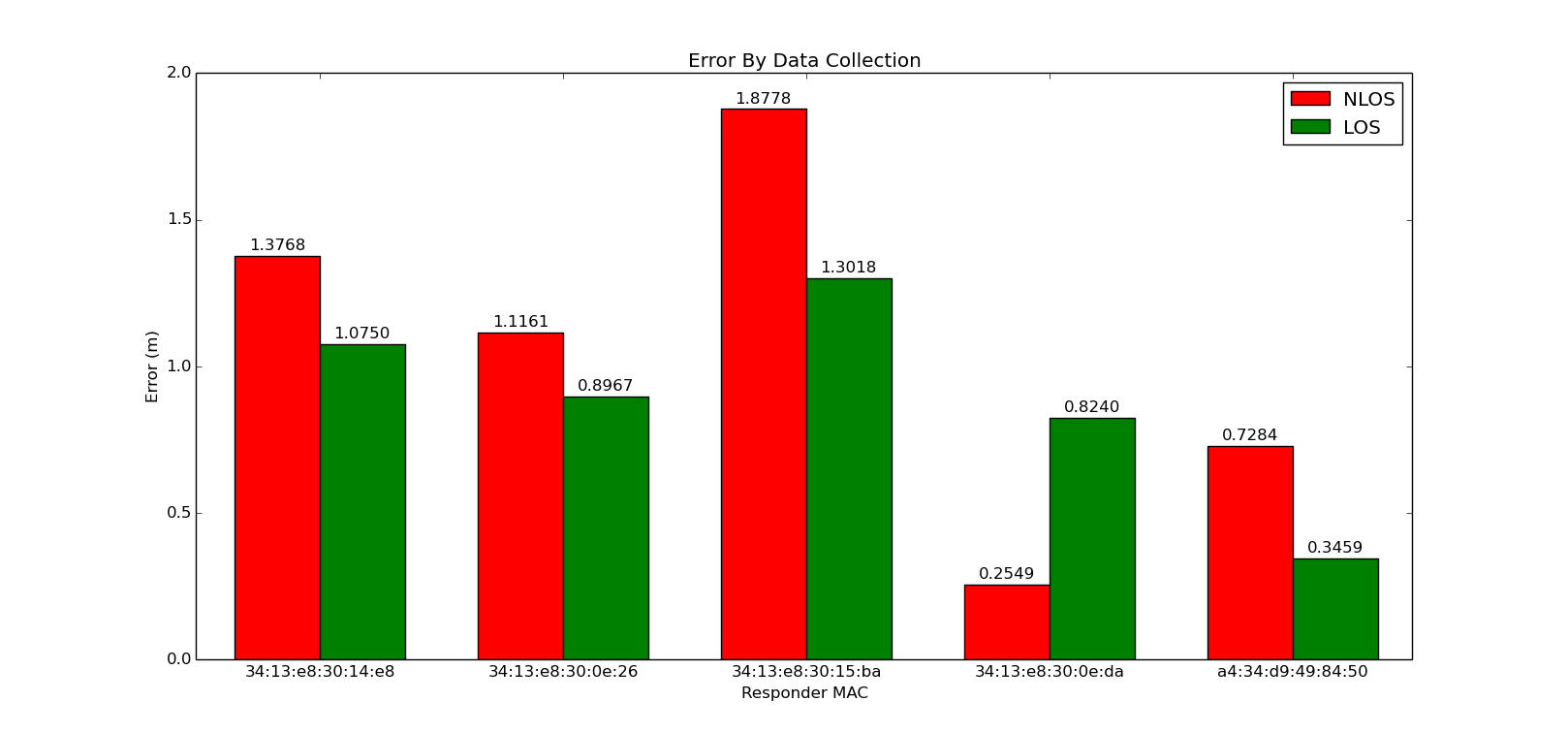
איור ‑ CDF משוער

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

Anon., n.d. [Online]   
Available at: https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html

Anon., n.d. [Online]   
Available at: https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.linalg.svd.html

Hereman, W. & Murphy, W. S., 1995. Determination of a Position in Three Dimentions Using Trilateration and Approximate Distances. *Decision Sciences.*

Strang, G., 2006. *Linear Algebra and Its Applications.* s.l.:Cengage Learning.