The Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

מערכת SLAM לרכב אוטונומי

פרויקט מס' 24-1-1-3049

דו"ח סיכום

מבצעים:

אורית ציקינובסקי 314617572

תובל אליהו 208606202

מנחים:

רועי רייך אוניברסיטת ת"א

מקום ביצוע הפרויקט:

אוניברסיטת ת"א

תוכן עניינים

רשימת איורים

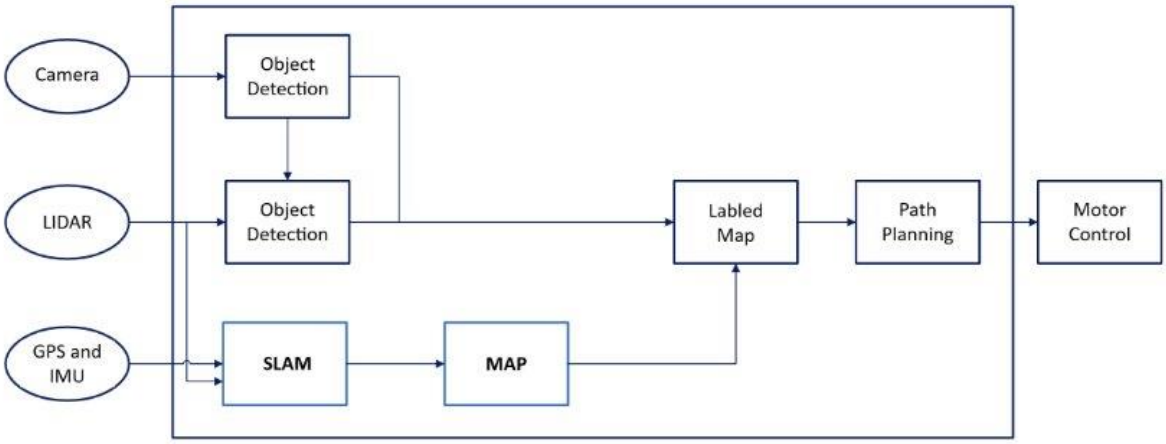
רשימת טבלאות

תקציר

הפרויקט הינו חלק מפרויקט המכונית האוטונומית של אוניברסיטת תל אביב, שמטרתו לחקור את עולם הרכבים האוטונומיים תוך הטמעת מערכות שונות ברכב של האוניברסיטה, לכדי הגעה להתנהגות אוטונומית מלאה. החלק שלנו בפרויקט זה הוא חקירת והטמעת אלגוריתם SLAM ברכב.

SLAM (Simultaneous Localization and Mapping) הוא אלגוריתם אשר בעזרתו הרכב מייצר מפה בהתבסס על נתונים שהוא מקבל מחיישנים תוך מיפוי עצמי במפה המיוצרת. בכך, האלגוריתם מאפשר לרכב להתמצא בסביבה לא מוכרת בעזרת חיישנים שונים המותקנים על הרכב.

עבור פרויקט זה היו שתי מטרות עיקריות, האחת היא לבצע סקירה של אלגוריתמים קיימים אשר התאימו מבחינת הנתונים הטכניים לחיישנים הנמצאים ברכב ולמחשבי הרכב, להטמיע ולחקור אותם במחשבים הביתיים. האלגוריתמים שנבחרו לאחר שלב זה, אשר התאימו לנתונים הטכניים והטמעתם והרצתם בוצעה בהצלחה היו Lio RF, Fast LIO. מטרתנו הסופית בפרויקט הייתה הטמעת אלגוריתמי ה SLAM הנבחרים ברכב האוטונומי של האוניברסיטה, וחקר של העבודה והביצועים של האלגוריתמים האלו בהינתן חיישני הLiDAR והINS אשר מותקנים ברכב. כפי שנפרט בהמשך הפרויקט, ראינו כי במעבר לעבודה עם חיישנים ברכב ישנן תקלות לעומת העבודה במחשבים הביתיים עם הקלטות של מאגרי נתונים. בסופו של דבר, מטרתנו הייתה להביא את פרויקט הרכב האוטונומי צעד אחד קדימה לנסיעה אוטונומית מלאה.



איור 1 – דיאגראמת בלוקים

1 הקדמה

מטרות הפרויקט הן:

1. בחירת אלגוריתמים אשר יתאימו לסביבת העבודה של רכב האוניברסיטה, הטמעת אלגוריתמי SLAM במחשב הביתי והרצתם עם מאגר מידע קיים תוך בדיקת תקינות.
2. הטמעת והרצת אלגוריתמי הSLAM הנבחרים ברכב האוניברסיטה עם חיישן ה LiDAR של חברת Ouster (דגם OS1-128) והחיישן INS-IMU (דגם INS-DL). ההרצה על מחשבי הרכב תתבצע בזמן אמת עם הנתונים אשר מתקבלים מהחיישנים המוזכרים לעיל.

הפרויקט הוא אחד מאבני הבניין של פרויקט המכונית האוטונומית של אוניברסיטת תל אביב שמטרתו לחקור ולהטמיע את עולם הרכבים האוטונומיים. תחום זה נמצא בחזית הקדמה הטכנולוגית ומשלב בעיות הנדסיות מורכבות כגון בחירה והטמעת חיישנים (ליידר, מצלמות ועוד) עם התממשקות כל המערכות השונות ברכב לכדי נסיעה אוטונומית, ומציאת פתרונות שיאפשרו לרכב להתמצא במרחב לא מוכר שמשתנה כל הזמן. החלק שלנו בפרויקט זה יאפשר לרכב לתפקד ללא כל התערבות אנושית. על הרכב האוטונומי להתמצא במרחב בזמן אמת, ובאמצעות אלגוריתם SLAM ניתן לעשות זאת.

שלבי העבודה של הפרויקט כללו:

1. סקירה של אלגוריתמי SLAM קיימים וסינון לפי מאפייני סביבת העבודה של הרכב: חיישני LiDAR ו-INS-IMU, תאימות עבודה ל ROS2 בגרסת ה foxy ול Ubuntu 20.04 בסביבת מכונה וירטואלית.
2. הטמעת שני אלגוריתמים מתאימים במחשב הביתי, והרצתם.
3. הטמעת האלגוריתמים הנבחרים במחשב הרכב האוטונומי של האוניברסיטה, וניסיונות הרצתם לשם השוואת תוצאות ובחירה של אלגוריתם.

ישנם שני סוגי אלגוריתמי SLAM עיקריים, האחד, Visual SLAM, המשתמש בנתונים ממצלמה, והשני, הוא הסוג שבו השתמשנו בפרויקט זה והוא מבוסס בעיקר מנתוני Point Cloud אשר מתקבלים מחיישני LiDAR. אחת הסיבות העיקריות לבחירת אלגוריתם SLAM אשר מבוסס על נתונים מחיישני LiDAR הוא דיוק בסריקת סביבת הרכב בתנאים משתנים. היתרון הבולט הוא הדיוק של הLiDAR בתנאי מזג אוויר כגון ערפל\אובך [4], שכן הנתונים שמתקבלים ממנו לא מושפעים מנתוני מזג האוויר, ואילו בהינתן התבסוסת על מצלמה הנתונים שמתקבלים משתנים, דבר אשר מקשה על פיענוח מדויק של סביבת הרכב וכתוצאה מכך עלול להתקבל מיפוי לקוי.

2 רקע תיאורטי

- אלגוריתם SLAM

SLAM (Simultaneous Localization and Mapping) הוא אלגוריתם שימושי עבור תחום הרובוטיקה בכלל, ותחום הרכבים האוטונומיים בפרט. מטרתו לאפשר לרכב להתמצא בסביבה לא מוכרת ודינאמית תוך התבסוסות על נתונים אשר מגיעים ממספר חיישנים (LiDAR או מצלמה, INS, IMU, GPS). הוא עושה זאת באמצעות מיפוי של הסביבה בזמן אמת, עדכון מפה שכוללת את כל מסלול הרכב מתחילת נסיעתו ועד הנקודה הנוכחית ומיקום של הרכב במפה זו.

ישנם דרכים שונות לממש אלגוריתם SLAM:

- ❖ SLAM מבוסס מסנן, הוא אלגוריתם אשר מתייחס לבעיית הSLAM בתור בעיית שערוך מצב. באלגוריתם זה, המצב שמשערכים הוא המיקום הנוכחי והמפה. סוג זה של אלגוריתם מתבסס בעיקר על מסנן קלמן (Kalman filter), ואחת הדרכים לממש אותו מבוצעת בשני שלבים. בשלב הראשון, האלגוריתם מבצע שערוך למצב, ובמצב השני הוא משלב את הנתונים שהתקבלו מהחיישנים ומעדכן את השערוך. עבור חלק מהמימושים באמצעות מסנן מניחים עולם לינארי ואילו עבור אחרים לא ולכן מבצעים פעולות מתמטיות שמטרתן להתמודד עם אי הלינאריות כגון לינאריזציה או שערוך של התפלגות הסתברות.[1]
- ❖ SLAM מבוסס גרף, הוא אלגוריתם המתייחס לבעיית הSLAM בתור בעיית גרף. הקודקודים של הגרף מייצגים את המידע על מיקום הרכב, ואילו הקשתות מבטאות את הקשר בין נקודות מיקום, מדידות הIMU וכן את המפה כולה אותה האלגוריתם מייצר. אלגוריתם זה תחילה מבצע שערוך למיקום של הרכב, לאחר מכן מודד את השגיאה בין המיקום המשוערך והמיקום בפועל, ופועל לצמצם את השגיאה כמה שיותר על מנת לתת מפה מדויקת ככל הניתן.
- ❖ SLAM מבוסס Deep Learning הוא תחום חדש בפיתוח. מטרתו להשתמש ברשתות נוירונים וב- Deep Learning על מנת לשפר את ביצועי האלגוריתם, בעיקר עבור אלגוריתמים המשתמשים בחיישן המצלמה כחיישן העיקרי. בעת שימוש במצלמה בתור חיישן ויזואלי תנאי התאורה והסביבה משתנים ומקשים על פעולה תקינה של האלגוריתם. בעזרת רשתות נוירונים ניתן לבצע זיהוי תמונה בצורה אשר תשפר את איכות התמונה לברורה יותר, ובכך לתת לאלגוריתם נתוני חיישן מדויקים יותר.

- מערכת ROS

(Robot Operating System) ROS הוא תוכנת קוד פתוח המהווה סביבת עבודה נוחה עבור פיתוח תוכנות עבור רובוטים. ROS מאפשר לבנות כל חלק מהרובוט בנפרד ולקשר ביניהם בקלות [5]. ROS מאפשר להתמקד בקבלת נתונים של כל חלק ממערכות הרובוט. לעומת זאת, ללא ROS, על המפתח לפתור גם את בעיות התקשורת בין החלקים, דבר אשר יכול להיות מסורבל מאד עבור מערכות גדולות ומסובכות, כמו מערכות של רכבים אוטונומיים.

משתמשים בROS במגוון רחב של תחומים כגון חקלאות, לוגיסטיקה, רובוטים נותני שירות, רחפנים, כמובן גם רכבים אוטונומיים ועוד.

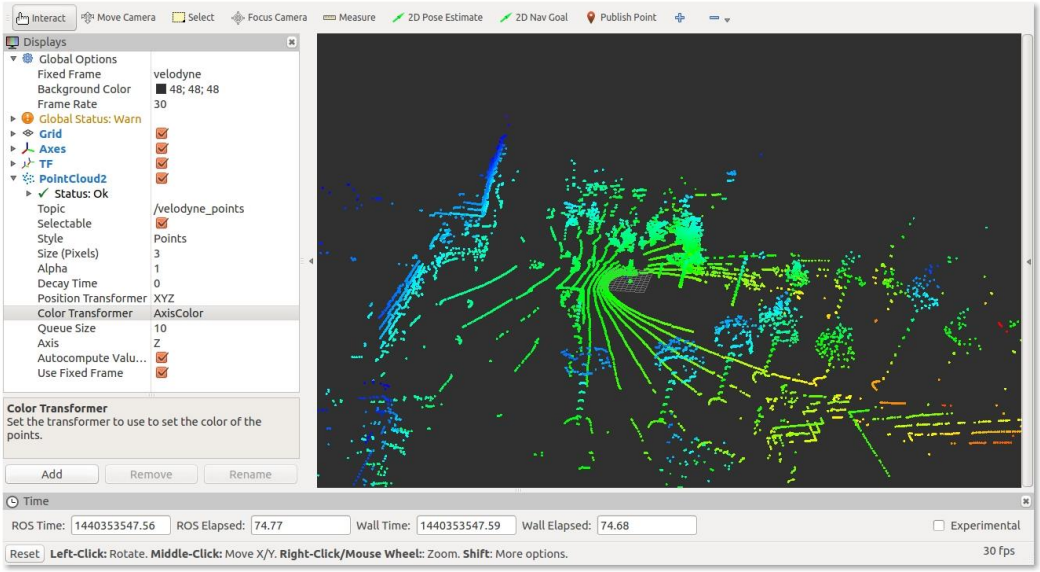
מערכת ה ROS עובדת בתצורת מבנה של גרף בצורה הבאה:

- Nodes (צמתים) – צומת הינו החלק שבו מתבצע עיבוד, כגון קבלה ושליחה של נתונים לצמתים אחרים, ביצוע בקרה, תכנון של פעולות בהתבסס על מידע שהתקבל או על מידע נשלח ועוד [7].
- Messages (הודעות) – הודעות הן מבנה נתונים בהן יש מידע אשר מעוניינים להעביר בין צומת לצומת. ההודעות ניתנות להקלטה באמצעות קובץ rosbag ובכך לאפשר את הרצתם שוב ושוב

לשם ניסויים. את ההודעות ניתן להכניס לכלים ויזואליים, גם הם נמצאים בROS, כמו למשל Rviz על מנת לבצע ניסויים או על הרובוט עצמו, או על סימולציה שלו.

- Topics (נושאים) – נושאים הם הכלי באמצעותו מועברות ההודעות בין הצמתים [6]. כל צומת יכולה לקבל או לשלוח הודעה בנושא מסוים. על מנת לעשות זאת עליה לבצע publish אם היא שולחת הודעה בנושא, או לבצע subscribe על מנת לקבל הודעות בנושא. כאשר צומת שולחת הודעה (עושה publish) בנושא מסוים, הצמתים שביצעו subscribe לנושא זה מקבלים את ההודעה.
- Services (שירותים) – שירותים הוא כלי נוסף לצומת שמאפשר לה להכריז על קבלת\שליחת הודעה מסוג נושא מסוים. כלי זה הוא עבור פעולה שצומת יכול לבצע לכדי תוצאה אחת. כלי זה דומה לאינטראקציית שרת-לקוח. על צמתים להכריז על פרסום שירות, ואם יש צומת שצריך להשתמש בשירות זה עליו לשלוח אליו הודעת Request, ולחכות עד שיקבל תשובה על ידי הודעת Reply.
- Parameter Server (שרת פרמטרים) – הוא מאגר מידע משותף לצמתים בו אגור מידע שלא משתנה לעיתים קרובות, או שלא משתנה בכלל. הצמתים משתמשים במידע במילון זה לשם אחסון או שליפה של מידע בזמן הרצה.
- Master (מאסטר) – תפקיד המאסטר הוא לאפשר לצמתים למצוא אחד את השני, ובכך לאפשר להם להעביר הודעות. המאסטר מספק שמות לצמתים, רושם אותם, עוקב אחר הצמתים שמבצעים publish ו-subscribe לנושאים, וגם עוקב אחר הצמתים שמבצעים Request ו-Reply עבור שירותים.

תוכנת ה Rviz – כלי ויזואלי תלת ממדי של ROS שבאמצעותו ניתן לסמלץ את סביבת הרובוט ואת נתוני החיישנים. תצוגת הכלי נראית בצורה הבאה:



איור 2 – תצוגת תוכנת Rviz

בתמונה זו ניתן לראות סביבת רובוט אשר מתקבלת על ידי חיישן ליידר מסוג Velodyne, בסביבה זו ניתן לראות למשל עצים המקיפים את הרובוט ואת הרחובות.

- LIDAR

Lidar (Light Detection and Ranging) הוא טכנולוגיה אופטית באמצעותה ניתן לקבוע מרחקים על ידי כיוון לייזר לפני השטח של עצם ומדידת הזמן שלקח לאור לחזור. ליידר יש שימושים רבים כגון יצירת מפות, מיפוי אטמוספרי, ומיפוי סביבה של רובוט או רכב אוטונומי [8]. חישוב המרחק בין התקן הליידר לבין נקודה אותה אנו מעוניינים לבדוק הינו:

$$R = c \cdot \frac{\Delta t}{2}$$

R – המרחק מהנקודה (m)

c – מהירות האור $\left(\frac{m}{s}\right)$

Δt – הזמן שלוקח לאור לחזור מהנקודה (s)

מרכיבי מערכת הליידר:

- Beam steering unit – יחידה זו אחראית על פליטת וכיוון אלומת הלייזר. לייזר הוא תוצר של פליטת אלומת אור צרה מאד, מונוכרומטית, קוהרנטית עם קווים מקבילים [9][10].
- Optics – המערכת האופטית של התקני ליידר היא חלק מיחידת Beam steering unit. במערכת זו, כיוון האלומות משתנה לזווית הרצויה, בין אם האלומה מגיעה מההחזרה מהעצם, או מכיוון פליטת הלייזר של התקן הליידר.
- Receiver unit – יחידה זו כוללת גלאי אור ואופטיקה רלוונטית. ביחידה זו מומר האור המוחזר מהעצם לסיגנל חשמלי.
- Processing unit – יחידת העיבוד מקבלת את הנתונים ממוצא הקולט (receiver), מבצעת להם דיגיטציה ומעבדים את הסיגנל הדיגיטלי לכדי מידע תלת ממדי point cloud. בנוסף, ביחידה זו מתבצע סינון הרעשים בעקבות הדיגיטציה של הסיגנל קודם לכן.

סדר הפעולה:

1. אלומת הלייזר מופעלת
2. השעון מתחיל לעבוד
3. אלומת הלייזר עוברת דרך המערכת האופטית
4. אלומת הלייזר פוגעת בעצם ומוחזרת
5. אלומת הלייזר עוברת דרך מערכת קולט האור
6. האלומה עוברת בגלאי האור ומומרת לסיגנל חשמלי
7. השעון מפסיק
8. ממיר אנלוגי לדיגיטלי ממיר את הסיגנל החשמלי לסיגנל דיגיטלי
9. הסיגנל הדיגיטלי מעובד לכדי point cloud.

- INS-IMU

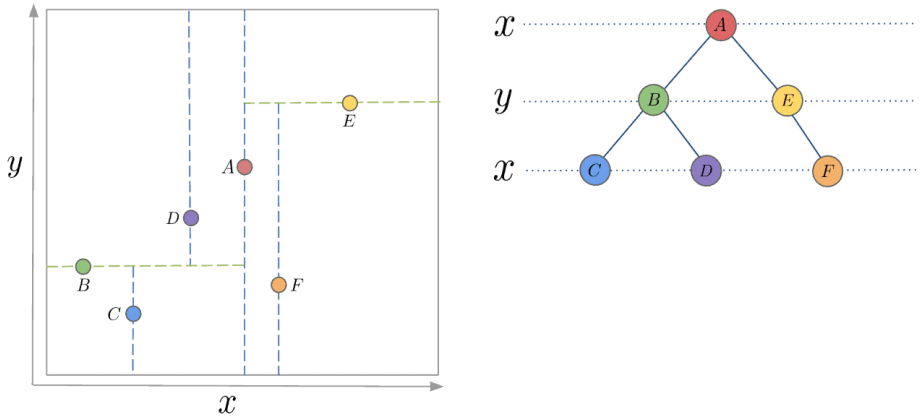
IMU (Inertial Measurement Unit) – יחידת מדידה אינרציאלית הוא התקן אלקטרוני אשר מודדת מומנטים ואת התדירות והתאוצה הזוויתית של המערכת בו הוא מותקן [12].

INS (Inertial Navigation Unit) – מערכת ניווט אינרציאלית הינו התקן אשר משתמש בחיישני תנועה וסיבוב על רכיבי הרכב על מנת לחשב את מיקום והמהירות של המערכת בו ההתקן מותקן [13]. לרוב, וכך גם בפרויקט זה, משלבים את ה IMU ואת ה INS. בשילוב זה, הנתונים אשר מתקבלים מהIMU מועברים ל INS על מנת לקבל מידע על המיקום היחסי של המערכת, המהירות שלה והתדירות הזוויתית במקביל. מערכת כזו רלוונטית במיוחד עבור אלגוריתמי SLAM לרכבים אוטונומיים, שכן באמצעות המידע המתקבל מחיישנים אלו ניתן לבצע חישובים שמספקים מידע נוסף על מיקום הרכב.

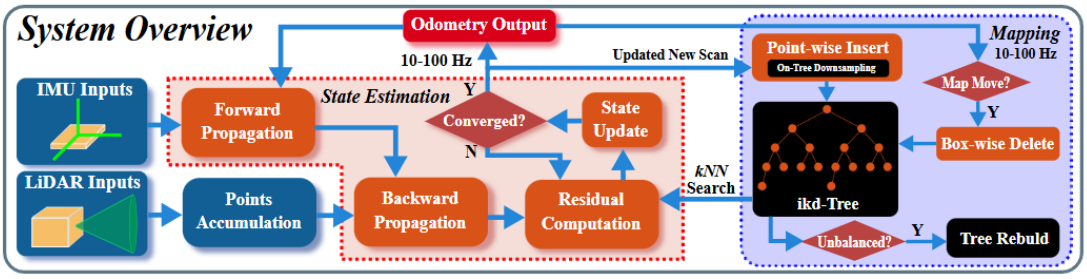
- FAST LIO

FAST LIO (Fast Lidar-inertial odometry) הוא אלגוריתם SLAM שמטרתו לבצע מיפוי ליידר והתמצאות במרחב באמצעות מימוש עם מסנן קלמן [14]. באלגוריתם זה ישנו שימוש ב Extended Kalman Filter, אשר מניח עולם לא לינארי, ומתמודד עם זה באמצעות לינאריזציה. אלגוריתמים מבוסס EKF נחשבים ליעילים וגמישים במיוחד עקב הנחת ההתנהגות הלא לינארית. גורמים נוספים אשר תורמים ליעילות האלגוריתם הינם:

1. רכיב ה Point Cloud אשר מתקבל לצורך בניית המפה לא עובר כל עיבוד, ובכך תיאור הסביבה של הרכב מדויקת יותר.
2. המפה מעודכנת באמצעות מבנה ikd tree, מבנה אשר מאפשר זמני חישוב קצרים יותר. מבנה זה הוא מבנה שמירת נתונים מסוג kd tree אשר מתעדכן וגדל. יתרון בולט של שימוש במבנה נתונים זה הוא הגעה וחיפוש מהיר של נקודות במבנה.



איור 3 – מבנה ikd tree



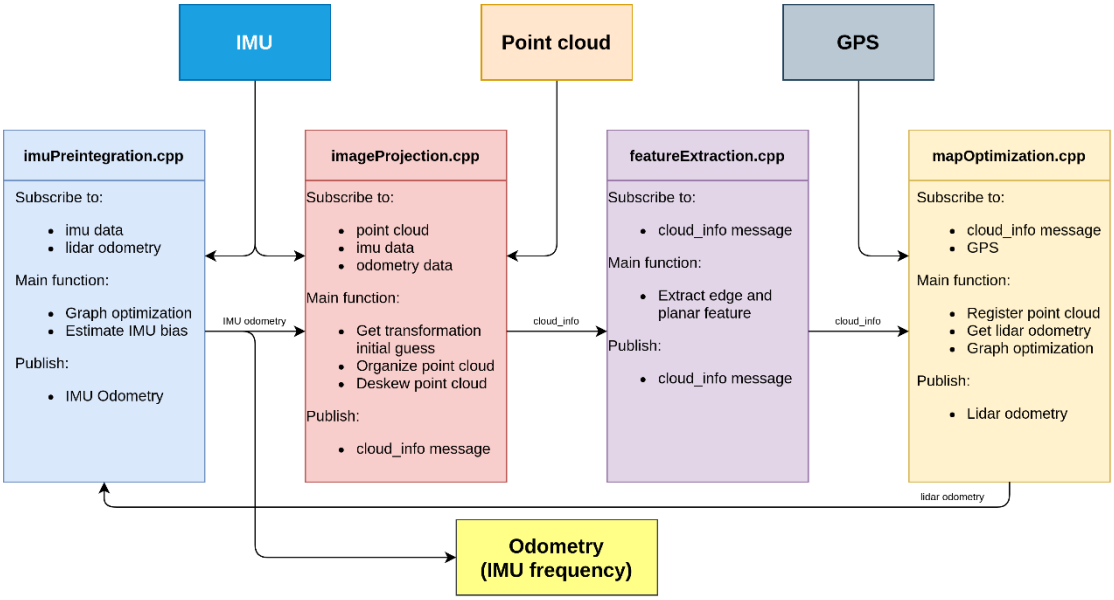
איור 4 – Pipeline של האלגוריתם Fast Lio

- LIORF

LIO RF הוא אלגוריתם SLAM מבוסס גרף. בעזרת אלגוריתם זה ניתן לייצר מפה מתעדכנת בזמן אמת של סביבת הרכב ולהתמצא בסביבה לא מוכרת. אלגוריתם זה הינו אלגוריתם משופר הנועד לעבוד עם ros2 foxy אשר מבוסס על אלגוריתם בשם Lio sam [15].

שלבי פעולת האלגוריתם:

1. האלגוריתם מתחיל בהערכת תנועה ראשונית בין סריקות ליידר בעזרת אינטגרציה של ה IMU
2. מבוצעת השוואה בין סריקות ליידר עוקבות על ידי התאמת נקודות מפתח (כמו קצוות ומשטחים) כדי לחשב את השינוי במיקום ונבחרים רק פריימים בהם מיוצג שינוי במיקום או בזמן (על מנת להפחית עומס חישובי ולא להפוך את הגרף לרועש ולא יציב מהרבה נתונים)
3. בניית גרף מפריימים נבחרים אלו (יצירת Node חדש בגרף אשר אליו מחוברות מדידות IMU קודמות אשר מתארות איך הרובוט זז בין שני פריימים, מדידות ליידר המבוססות על התאמה בין הסריקות, וזיהוי חזרה למיקום קודם – Loop closure).
4. אופטימיזציה ועדכון המפה התלת מימדית בזמן אמת על הגרף כולו כדי לתקן שגיאות מצטברות ולשפר את דיוק המיקום והמפה.

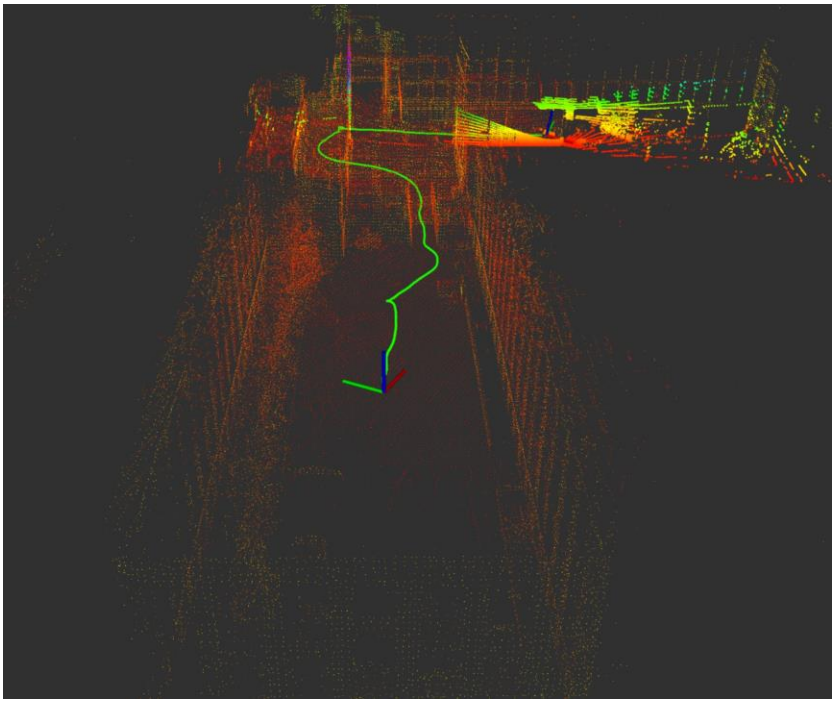


איור 5 – Pipeline של האלגוריתם Fast Lio

בגרף זה ניתן לראות את תרשים הזרימה של אלגוריתם LIORF, בתרשים זה קיימים 4 קבצי קוד עיקריים בשפת c++, האלגוריתם imuPreintegration הינו אחראי על שערוך התנועה בסריקות עוקבות דרך חיישן ה IMU, אלגוריתם ה imageProjection אחראי באופן כללי על ארגון ה point clouds לפי זמן וסידורן. אלגוריתם ה featureExtraction אחראי על זיהוי נקודות חשובות (קצוות ומישורים). ולבסוף אלגוריתם ה mapOptimization אחראי על מיפוי ואופטימיזציה של איחוד כל הנתונים ובניית הגרף תוך חישוב מיקום מדוייק.3

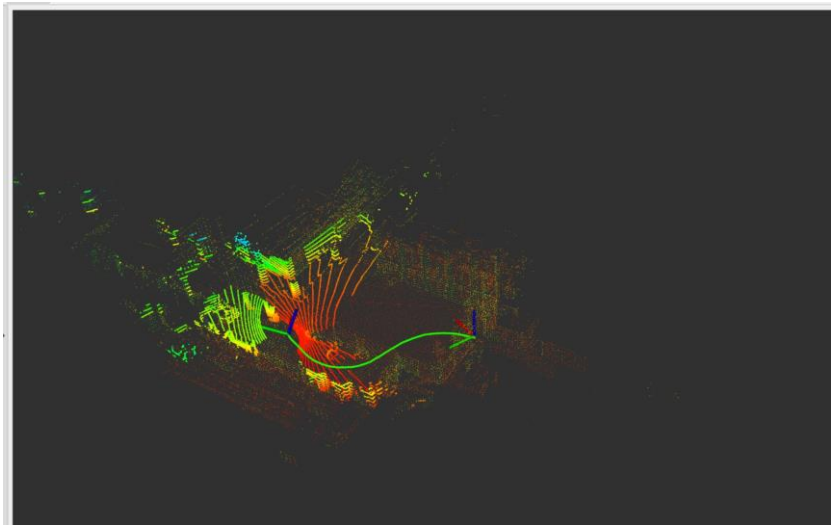
3 סימולציה

- אלגוריתם ה LIORF הוא אלגוריתם SLAM שמשלב ליידר ו IMU בתוך גרף לבניית מפה מדויקת והתמצאות בזמן אמת.



איור 6 – הרצת הסימולציה של אלגוריתם ה Liorf

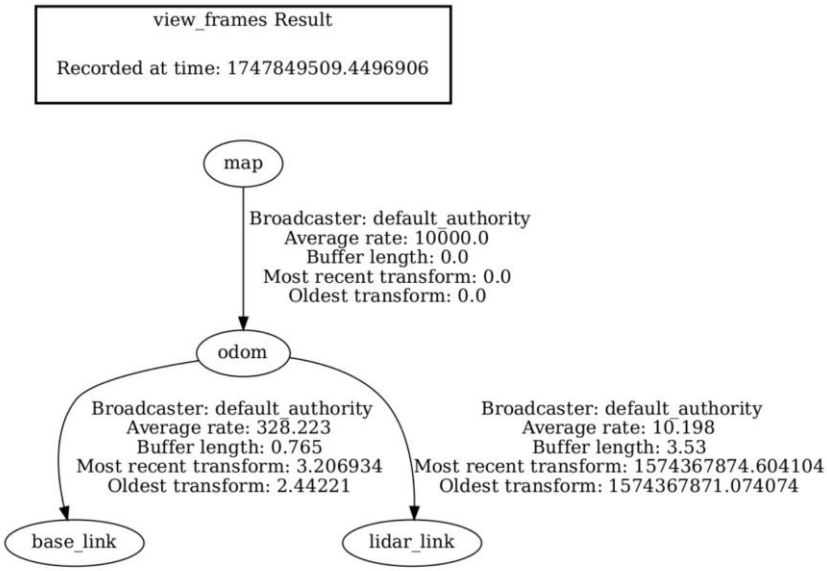
באיור זה ניתן לראות מפת SLAM תלת מימדית המורכבת ממסלול ונתוני point cloud (נקודות ענן).



איור 7 – נקודת מבט שונה למפה התלת מימדית של אלגוריתם ה Liorf

בשתי התמונות ניתן לראות ריצה ממאגר נתונים NCLT אשר מדמה הליכה במבנה סגור עם חיישן ליידר מסוג Velodyne עם 16 ערוצי דגימה ורזולוציה הוריזונטלית של 1800.

לצורך הבנת מבנה מערכת הפריימים והאם יש שגיאות או ניתוקים בין odom, base_link, map ו lidar_link . בוצע ניתוח של TF Tree בעזרת הפקודה ros2 run tf2_tools view_frames



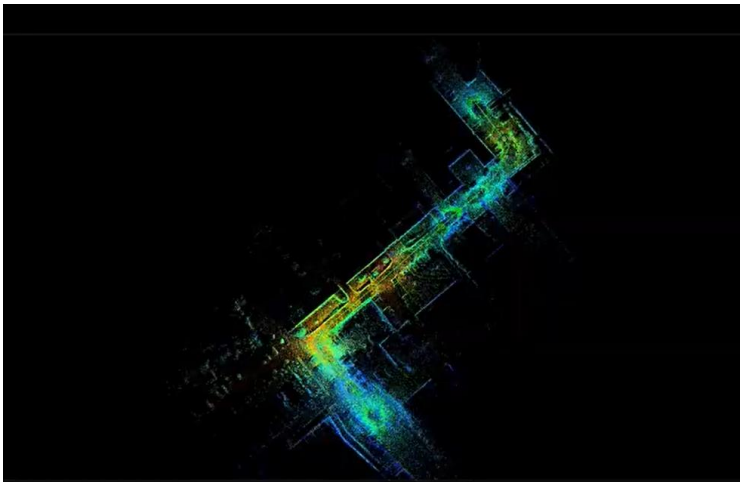
איור 8 – סימולציית TF (Transform Frames) דרך ה ROS2 של המערכת במהלך ריצה

התרשים מציג את סימולציית ה-(Transform Frames) TF של המערכת במהלך ריצה ב־ROS2. הוא מתאר בצורה היררכית את הפריימים השונים, את מיקומם היחסי ואת תדירות העדכון שלהם. בתרשים ניתן לראות את הקשרים המרחביים בין פריימים כמו map, odom, base_link ו־ lidar_link, כולל מידע על קצב שידור וזמני טרנספורמציה (הגרף הנ"ל הינו גרף תקין ושלם של הרצת מאגר המידע).

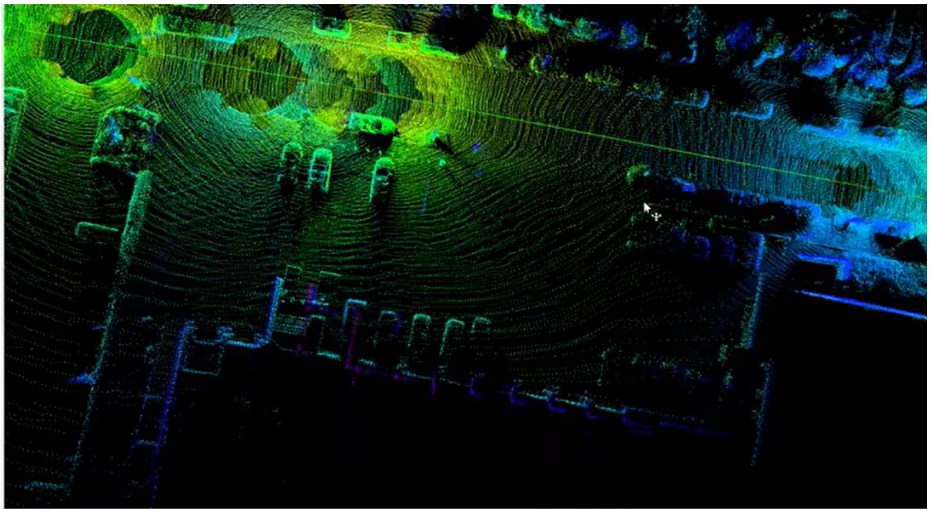
- אלגוריתם הFAST LIO - הוא אלגוריתם SLAM שמשלב LiDAR ו־IMU לביצוע מיפוי והתמצאות בזמן אמת באמצעות פילטר קלמן (EKF) ומבנה נתונים יעיל מסוג IKD-Tree .

בסימולציות הבאות ניתן לראות ריצות של האלגוריתם על מאגר הנתונים KITTI אשר מדמה SLAM בסביבה משתנה של רכב אוטונומי.

מאגר נתונים זה הוקלט על ידי חיישן ה Velodyne בהקלטת ליידר של 360 מעלות.



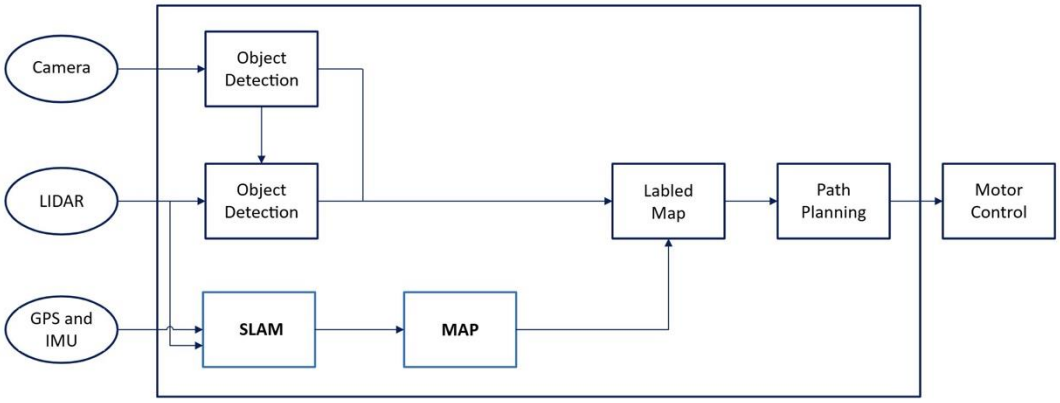
איור 9 – הרצת הסימולציה של אלגוריתם הFAST LIO



איור 10 – הרצת הסימולציה של אלגוריתם הFAST LIO בZOOM IN

לאחר בדיקת האלגוריתמים במערכות המחשב הביתיות ניתן היה להבין את איכות המפות המיוצרות וכמו כן גם להבין את התקלות בהתאמת האלגוריתמים לגרסאות של התוכנות השונות וכיצד ניתן לפתור אותן. שלב זה היווה חלק מהותי בפרויקט אשר אפשר את המשך השלבים הבאים בצורה חלקה יותר.

4 מימוש



איור 2 – דיאגראמת בלוקים

התרשים מציג את ארכיטקטורת המידע ברכב אוטונומי, שבה חיישנים כמו מצלמה LiDAR, GPS ,ו IMU אשר מספקים נתונים למערכת שתאפשר בסופו של דבר לרכב לנהוג בצורה אוטונומית לחלוטין. המידע עובר עיבוד ראשוני לזיהוי עצמים, במקביל להרצת אלגוריתם SLAM שמחשב את מיקום הרכב ובונה מפה של הסביבה. המידע המשולב מוזן למודול המפה ,(MAP) היוצר מפה מסומנת (Labeled Map) הכוללת עצמים מזוהים. מפה זו משמשת את מערכת תכנון המסלול (Path Planning) שמפיקה מסלול בטוח לרכב, ומעבירה את הפקודות למערכת הבקרה המוטורית (Motor Control) להפעלת הרכב בפועל. על מנת שהמידע שמתקבל מהחיישנים יעבוד עם אלגוריתמי הSLAM יש צורך בסביבת עבודה שתתמוך בכך. כדי לא לעבוד על כל אינטגרציה בנפרד, נבחר להשתמש בתוכנת Ros אשר מאפשרת תקשורת נוחה בין הרכיבים השונים במערכת הרכב האוטונומי המוצגות בדיאגרמה זו.

4.1 תיאור חומרה

המערכת הותקנה ברכב מסוג KIA Niro השייך למעבדת הרכב האוטונומי של אוניברסיטת תל אביב. על הרכב הותקנו החיישנים הבאים:

- Ouster OS1-128 חיישן ליידר
- חיישן INS-DL של חברת Inertial Labs
- חיישן ליידר של חברת Innoviz אשר אינו בשימוש על ידינו מאחר וחיישן ה Ouster עוקף את חיישן זה ביכולותיו.
- חיישני מצלמות

המערכות הללו מתקשרות באמצעות מחשבים מסוג Jetson של חברת NVIDIA אשר בעלות יכולות עיבוד גרפיות גבוהות.

איור 11 – חיישן ה LiDAR Ouster OS1-128 אשר הותקן על גג הרכב

איור 12 – הרכב בשלמותו עם כל חיישניו

4.2 תיאור תוכנה

1. <u>למידת התוכנות והבנת מהו SLAM</u>

בשלב הראשון בוצע מחקר תיאורטי להבנת עקרונות מערכת SLAM - מערכת שמאפשרת לרכב או לפלטפורמה ניידת לבנות מפה של הסביבה תוך כדי חישוב מיקומה היחסי בתוך אותה מפה. נלמדו ההבדלים בין סוגי SLAM שונים, בנוסף להבנה תאורטית זו, למדנו כיצד להפעיל ולהתמודד עם מערכות Linux ו־ ROS בנוסף ליצירת Nodes פרסום/הרשמה ל־ Topics ניהול פריימים (TF) והרצת סימולציות.

שלב זה סיפק לנו את הבסיס התיאורטי והמעשי שהיווה תשתית חיונית להתקדמות בפרויקט.

2. <u>מעבר על אלגוריתמי SLAM קיימים</u>

בשלב הראשוני של הפרויקט בוצעה סקירה רחבה של אלגוריתמי SLAM מתוך מטרה למצוא את האלגוריתמים אשר מתאימים בצורה המיטבית ביותר לחיישני הרכב האוטונומי, הבחירה התבצעה גם תוך התחשבות ביכולות העבודה בזמן אמת.

עברנו על מגוון רחב של אלגוריתמי SLAM בקוד פתוח, תוך הצלבה מול טבלת השוואה שבנינו (טבלה מספר 1 בנספחים), שהתחשבה בפרמטרים חשובים כמו:

- תאימות לגרסאות Ubuntu ו־ ROS כולל ROS1 וROS2
- סוגי חיישנים נתמכים LiDAR, IMU, GPS
- שימוש ב־ GPU לעומת CPU בלבד (כחלק מדרישות הפרויקט)
- מאגרי נתונים נתמכים כגון KITTI, Mulran, M2DGR
- דרישות חישוב ומשאבים
- יציבות בשטח וסימולציות

בין האלגוריתמים שנבדקו: LeGo-LOAM, Cartogtapher, Kudan , LIO SAM ,FAST LIO Lidar SLAM

תהליך ההשוואה כלל בדיקת קבצי קונפיגורציה, ניתוח כללי של קוד מקור והשוואת יכולות מיפוי בסביבות שונות מהרצות קיימות. כל הנתונים סוכמו בטבלת החלטה אשר סייעה לנו לצמצם את הבחירה לשני אלגוריתמים מובילים.

לאחר בחינה מעשית ותיאורטית, רוב האלגוריתמים נפסלו עקב מגבלות תאימות, ביצועים לא מספקים או חוסר תמיכה ב־ ROS2 נבחרו להמשך בדיקה שני אלגוריתמים בלבד:

LIORF - גרסה מתקדמת של LIO-SAM עם תמיכה מלאה ב־ROS2 Foxy

FAST LIO - אלגוריתם מהיר ומדויק עם שילוב אינרציאלי-ליידר הדוק

3. <u>מימוש שלב הסימולציה</u>

שלב אשר תוצאותיו הוסברו בפירוט לפני כן בו התוקנו האלגוריתמים ונבדקו בסימולציות במחשבינו הביתיים. על מנת לבחון את תקינות האלגוריתמים תוך ההתמודדות עם סביבות דינאמיות כולל שיפור מסלול לאורך זמן התקיימו כמה סימולציות לאלגוריתמים אשר נבחרו- LIORF (שהינו גרסה משופרת של האלגוריתם LIO-SAM) והאלגוריתם FAST LIO.

- שלבים מקדימים להתקנה:

1. התקנת מכונה ווירטואלית - מאחר ומערכות ההפעלה של המחשבים הניידים שלנו הינם על בסיס Windows אנו נרצה להתקין מכונה ווירטואלית בשם VMWARE אשר מאפשרת הורדה של תוכנת הפעלה השונה מתוכנת ההפעלה המקורית של המחשב.
2. התקנת מערכת ההפעלה - בעזרת פתיחת המכונה הווירטואלית הורדנו והתקנו מערכת הפעלה המבוססת LINUX בשם Ubuntu בגרסת ה 20.04 אשר הינה אותה הגרסה הקיימת במערכות הרכב.
3. התקנת ה ROS - הותקנה תוכנת הROS2 בגרסת הfoxy וכל חבילותיו הנוספות המאפשרות את פעולתו התקינה.

- התקנות האלגוריתמים:

התאמה בהתקנות האלגוריתמים - בבדיקה מקיפה של אלגוריתמי ה SLAM ניתן היה לראות כי מרבית האלגוריתמים מותאמים למערכת ההפעלה ubuntu 22.04 ובנוסף לכך גם מותאמים לתוכנת ה ros2 אך בגרסת ה humble בלבד, ואלגוריתמים אשר תאמו למערכת ההפעלה

ubuntu 20.04 הותאמו לתוכנת ה ros1 - בעקבות כך נאלצו ליצור התאמה ולחפש פתרונות נוספים.

❖ אלגוריתם ה LIORF:

ראשית כל התקנו את הגרסה המקורית הנקראת LIO SAM מהגיט– בהתקנה זו נתקלנו בקשיי קומפילציה רבים בעקבות אי ההתאמה ל ros2.

לאחר התקנתו נכחנו לדעת כי קיימת גרסה מתקדמת של אלגוריתם זה בשם LIORF אשר בעל התאמה גבוהה יותר לגרסת ה ros2 ובמיוחד לגרסת ה ros2 foxy .

ההרצה כללה בדיקת עמידה בדרישות, השוואת מפות מול נקודות במסלולים שונים ובנוסף לכך נבחנה יציבות האלגוריתם.

נתונים טכניים:

1. יכולת עבודה עם IMU בעל 6 צירים או 9 צירים.
2. עבוד עם IMU בעל קצב נתונים נמוך (כמו 50Hz)
3. בעל קבצי קונפיגורציה ועבודה מוכנים עם מאגרי נתונים שונים כגון M2DGR, Mulran
4. עבודה עם חיישני ליידר שונים כגון Ouster, velodyne
5. בעל יכולות LoopCloser
6. בעל יכולת לשליטת בכמות ה Core אשר ינוצלו במחשב עם עבודה מקבילית

❖ אלגוריתם FAST LIO:

האלגוריתם המקורי נכתב עבור Ros1, ולאחר מכן בוצעו התאמות עבור Ros2. לצורך התקנת האלגוריתם נעשו התקנות של סביבת העבודה- התקנת PCL 1.8 והתקנת Eigen 3.3.4., לאחר ההתקנה שכפלנו את הגיט ולבסוף בנינו אותו.

במהלך ההתקנה נתקלנו בבעיית אי תאימות של ספריית rclcpp לגרסת ה ros foxy שתוקנה בעזרת עדכון החלק הרלוונטי בקוד.

נתונים טכניים:

1. עובד עם IMU בעל 6 או 9 צירים.
2. כולל קבצי קונפיגורציה למאגרי נתונים כמו KITTI, NTU VIRAL, UrbanLoco
3. תומך בחיישני LiDAR שונים כמו.Velodyne, Ouster, Hesai, Livox
4. לא כולל מנגנון Loop Closure כחלק מהחבילה המקורית – אך ניתן לשלב עם קוד חיצוני.
5. מאפשר ביצוע מקבילי יעיל בזכות שימוש בעיבוד אינקרמנטלי ומבני נתונים מהירים-ikd) tree)

❖ התקנת האלגוריתמים ברכב:

לאחר בדיקה של שני האלגוריתמים בשלב הסימולציה, הותקנו האלגוריתמים במחשבי הרכב האוטונומי באוניברסיטה. ההתקנה במחשבי הרכב נעשתה מאפס כאשר כל ההתאמות בוצעו בצורה חלקה בעקבות הלמידה מהסימולציות וההתקנה הביתית.

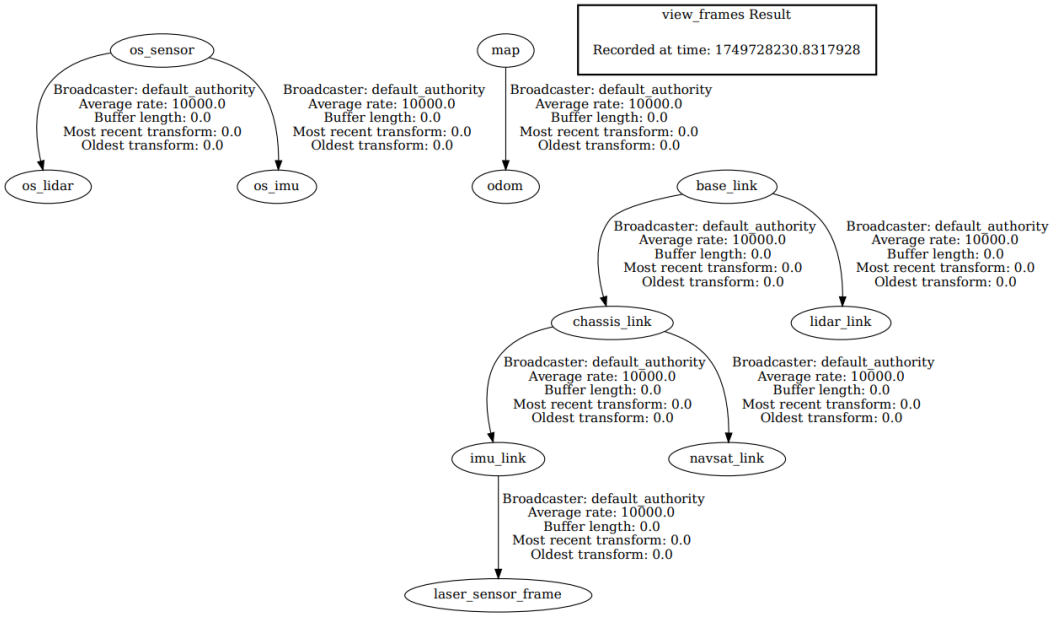
במחשבי הרכב קיימים כמה מחשבי Jetson אשר אחד מהם מחובר ישירות לחיישן הליידר של חברת Ouster. חיישן ליידר זה ברכב פועל בפרמטרים N_scans=10, Horizontal_scans=1024.

לאחר התקנה מוצלחת של שני האלגוריתמים (Fast-LIO, LIORF) על מנת לוודא כי הינם עובדים כראוי הרצנו את מאגר הנתונים משלב הסימולציה במחשבי הרכב ווידאנו כי התוצאות אכן חוזרות על עצמן.

5 ניתוח תוצאות

5.1 השוואות בין תוצאות הסימולציה לעבודה בזמן אמת

LIORF- במהלך שלב ההרצה בפועל, הצלחנו להעלות את מערכת ה SLAM על מחשב הרכב ולבצע חיבור בינה לבין חיישן ה־ LiDAR עם קליטת ענן הנקודות שהינו מייצר ולצפות בו דרך תוכנת הRVIZ באלגוריתם ה LIORF. עם זאת, האלגוריתם עצמו לא הצליח להשלים תהליך SLAM מלא – כלומר, לא נוצרה מפה דינמית, ולא התבצע חישוב מיקום בזמן אמת .אחת הבעיות המרכזיות שזוהתה הייתה מבנה פריימים (TF Tree) לא שלם בחיבורו: מתוך הפלט של view_frames דרך תוכנת ה ros נצפתה התמונה הבאה:



איור 13 – TF Tree

בתמונה זו ניתן לראות כי ה os_sensor (שהינו הרכיב אשר מקבל את הנתונים מהליידר) אינו מחובר ל odom ולרכיב ה base link אשר בונים את המפה התלת מימדית השלמה. מצב זה מונע מהאלגוריתם לחשב מיקומים יחסיים ולבנות מפה מסונכרנת.

באיור 7 סימולציית TF ניתן לראות עץ TF תקין אשר נוצר בסימולציה במחשב הבייתי, מההשוואה ניתן לראות כי במחשב הבייתי ישנה קישוריות מלאה בין המרכיבים ואילו בעץ זה לא קיימת קישוריות זו.

למרות שהמערכת לא הפיקה תוצר סופי של מפה, שלב זה היה קריטי להבנת מבנה המערכת ולזיהוי נקודות התורפה שבה. בוצע תיעוד של כל הממצאים, ניתוח שגיאות, ואימות ראשוני לתקלות. תהליך זה מהווה בסיס להמשך פיתוח, התאמה, וניפוי תקלות – ויביא לשיפור משמעותי בהרצות הבאות.

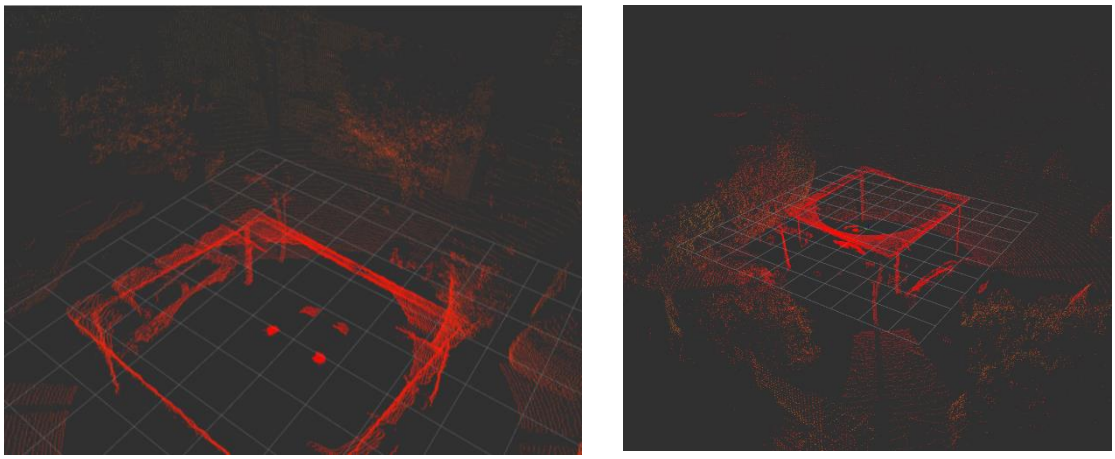
<u>תקלות בסימולצית Fast Lio ברכב :</u>

תקלת מידע חסר עבור שדה "ring" בזמן ריצת האלגוריתם - בזמן ניסיון עבודה עם האלגוריתם ברכב תוך כדי ריצת האלגוריתם התקבלו התראות חוזרות עבור מידע חסר עבור שדה "ring": [fastlio_mapping-1] Failed to find match for field 'ring'. הבעיה נוצרת בגלל אי תאימות של אובייקט "Point" עבור חישובים עם ספריית Eigen וגם אי תאימות בכמה משתנים אשר מחזיקים את ערך ה "ring" מול חישובים עם ספריית PCL. בעיות אלו קיימות ספציפית במימוש עבור חיישני ouster. גם במבנה ה Point וגם במשתנים השונים הקצו 8 ביט לערך ה ring אך למען החישובים עבור חיישן ה ouster נדרשת רזולוציה של 16 ביט. פתרון לתקלה זו מפורט

במדריך למשתמש, למרוץ פתירת תקלה זו לאחריה אין אנו הגענו לתוצאה כלשהי עם אלגוריתם זה. חוסר תקשורת בין החיישנים לבין האלגוריתם – בעת ההרצה ברכב, נתוני הPoint Cloud של הליידר במוצא האלגוריתם לא נראו בRviz. לא נמצא המקור לתקלה. בעקבות תקלה זו גם לא נוצרה מפת SLAM מתעדכנת בזמן אמת.

5.2 ביצועי המערכת מבחינת זמן אמת

בעת הרצת אלגוריתם ה LIORF במחשבי הרכב התקבלו מספר תמונות.



איור 14 – הרצת אלגוריתם ה LIORF בזמן אמת

תמונות אלו התקבלו מחיישן הליידר ומציגות את ה point clouds של סביבת הרכב האוטונומי בחנייה באוניברסיטה דרך חיישן ה Lidar של חברת Ouster.

פרמטרים כמותיים

	אלגוריתם LIO SAM	אלגוריתם FAST LIO
קצב היענות SLAM	10-12Hz	2.5-4.6Hz
דיוק מרחק סטטי	לא נבחן בזמן אמת	לא נבחן בזמן אמת
דיוק מרחק דינאמי	ללא נבחן בזמן אמת	לא נבחן בזמן אמת

טבלה 1 – פרמטרים כמותיים

6 סיכום, מסקנות והצעות להמשך

סיכום ובחינת תוצאות הפרויקט:

פרויקט זה כלל שני חלקים עיקריים, האחד חיפוש אלגוריתמים מתאימים מבחינת הנתונים הטכניים והרצתם התקינה במחשבים הביתיים. השני, הטמעה והרצה תקינה של אלגוריתמים נבחרים במחשבי הרכב האוטונומי של אוניברסיטת תל אביב.

- **שלב הסימולציה הביתית** – שלב זה היווה הכנה מקדימה ומעמיקה אשר פרסה את התשתית לעבודה עם הרכב בשלב הבא. העבודה התיאורטית בשלב זה כללה סקירת מידע תיאורטי בכל הנוגע לאלגוריתמי SLAM, חיפוש וסינון אלגוריתמי SLAM בהתאם לנתונים הטכניים של המערכת וכן היכרות עם סביבת העבודה שהוגדרה. העבודה המעשית בשלב זה כללה התקנה של מספר אלגוריתמים על מנת לבדוק את התאמתם בפועל לסביבת העבודה המבוקשת, ובנוסף כדי ללמוד על התקלות האפשריות במהלך העבודה עם האלגוריתמים ובכך להקל על העבודה במחשבי הרכב.
  במהלך ההתקנות והסימולציה נתקלנו בקשיים עקב חוסר תאימות של אלגוריתמים ומאגר הנתונים Kitti שהוסבו מ ROS1-ל ROS2-וחוסר תמיכה בעמודי ה GitHub-שלהם.
  האלגוריתמים אשר נבחרו לעבודה ברכב הינם Fast LIO ו- LIORF. אלגוריתמים אלו הותקנו במחשבים הביתיים בהצלחה ובוצעו באמצעותם סימולציות על גבי מאגרי נתונים שונים אשר הראו בהשוואה חזותית פעילות תקינה. בנוסף לכך, הדרישה הכמותית של קצב היענות האלגוריתם נענתה בשלב זה.
- **שלב ההרצה במערכת הרכב –** האלגוריתמים שנבחרו בשלב הקודם Fast LIO ו- LIORF הותקנו במחשבי הרכב בהצלחה. בשלב ההתקנה היו תקלות אשר נתקלנו בהן גם בעת ההתקנה במחשבים הביתיים (סעיף 7). באמצעות הידע המקדים שצברנו בשלב הסימולציה הביתית פתרנו אותן בצורה חלקה. במהלך הניסויים, עם זאת, נתקלנו בבעיות התאמה בהן לא נתקלנו קודם לכן ולא ציפינו להן.
  בעיות אלו בסיסן בתקלות תקשורת בין האלגוריתמים לחיישנים, כפי שפירטנו קודם לכן ב5.1.
  על מנת לשלול גורמים נוספים, ביצענו סימולציות עם האלגוריתמים במחשבי הרכב על גבי מאגרי הנתונים בהם השתמשנו בסימולציה במחשבים הביתיים. הסימולציות רצו באופן תקין במחשבי הרכב.
  בעקבות בעיות ההתאמה האלו, לא התקבלה תוצאה של יצירת מפת SLAM מתויגת במחשבי הרכב.
  כמו כן, בעקבות אתגרים אלו גם לא בוצעה השוואת הפרמטרים הכמותיים בחלק זה. למרות זאת, הושגו ממצאים אשר מהווים תיעוד והבנה של תקלות מרכזיות. כל אלו מהווים בסיס יציב להמשך פיתוח.

הצעות להמשך שיפור ביצועי המערכת

- מעבר לעבודה עם גרסאות ROS ו Ubuntu עדכניות – באמצעות מעבר זה ניתן יהיה להשתמש באלגוריתמים אשר פותחו בגרסאות אלו (ולא פותחו בגרסאות ישנות ועברו המרה לגרסאות יותר עדכניות). בנוסף, ישנו סיכוי גבוה יותר כי האלגוריתמים אשר פותחו בסביבות עדכניות יהיו עם עמודי GitHub פעילים. שני אלה יאפשרו פיתוח יעיל וטוב יותר.
- עבודה עם מאגר נתונים אשר נתמך על ידי Ros2 באופן מלא לדוגמא הקלטות הרכב של האוניברסיטה, או ביצוע פרויקט המרת נתוני Kitti עבור Ros2 אשר יהיה מתועד.
- המשך חקירת בעיית ההתאמה בתקשורת בין החיישנים לאלגוריתמים, אשר תוביל להמשך פיתוח מעמיק בנושא הSLAM במחשבי הרכב.

7 תיעוד הפרויקט

מדריך למשתמש:

- פירוט תקלת rclcpp בתאימות עם ros2 foxy:

בעת בניית הפרויקט בעזרת colcon build עלתה שגיאה אשר מנעה מהפרויקט להיבנות בהצלחה, השגיאה מציינת כי פונקציה בשם "create_service" לא קיימת עבור אובייקט rclcpp. הפרויקט משתמש בפונקציה זו בקובץ laserMapping.cpp 2 שהוא הקובץ העיקרי הממשמש את אלגוריתם ה SLAM. כדי לנסות למצוא פתרון לבעיה תחילה ניסינו לחפש את הבעיה ב Issues בעמוד ה GitHub של הפרויקט, למזלנו אנשים נתקלו בבעיה זו בעבר והוצע commit אשר מתקן את הבעיה נקודתית עבור משתמשי foxy, commit זה לא התקבל כפתרון קבוע לפרויקט עקב חוסר פעילות של מנהלי הפרויקט.הפתרון המוצע מעדכן את השימוש בפונקציית "create_service" בכך שהפרמטרים שהפונקציה מקבלת צריכים להיות שונים במערכות foxy.

הפתרון הוצע ע"י המשתמש "hannnys" תחת issue #334 בעמוד ה GitHub.

```
945  -        map_save_srv_ = this->create_service<std_srvs::srv::Trigger>("map_save", std::bind(&LaserMappingNode::map_save_callback, this, std::placeholders::_1, std::placeholders::_2));
946  -
     945 +    map_save_srv_ = this->create_service<std_srvs::srv::Trigger>("map_save", [this](std_srvs::srv::Trigger::Request::SharedPtr req, std_srvs::srv::Trigger::Response::SharedPtr res)
{map_save_callback(req, res);});
```

איור 15 – פירוט תקלת rclcpp בתאימות עם ros2 foxy ופתרונה

- תקלת מידע חסר עבור שדה "ring" בזמן ריצת האלגוריתם:

פתרון לתקלה זו ניתן לפתור על ידי שינויים בקובץ preprocess.h אשר מגדיר מבנים עבור מחלקת preprocess שמסדרת את המידע מן החיישנים עבור העיבוד המרכזי של האלגוריתם.

לאחר החלפת המשתנים ההודעה נעלמת.

```
88   88      struct EIGEN_ALIGN16 Point
89   89      {
90   90        PCL_ADD_POINT4D;
91   91        float intensity;
92   92        uint32_t t;
93   93        uint16_t reflectivity;
94       -     uint8_t ring;
     94  +     uint16_t ring;
95   95        uint16_t ambient;
96   96        uint32_t range;
97   97        EIGEN_MAKE_ALIGNED_OPERATOR_NEW
     @@ -107,7 +107,7 @@ POINT_CLOUD_REGISTER_POINT_STRUCT(ouster_ros::Point,
107  107         // use std::uint32_t to avoid conflicting with pcl::uint32_t
108  108         (std::uint32_t, t, t)
109  109         (std::uint16_t, reflectivity, reflectivity)
110      -       (std::uint8_t, ring, ring)
     110 +       (std::uint16_t, ring, ring)
111  111         (std::uint16_t, ambient, ambient)
112  112         (std::uint32_t, range, range)
113  113       )
```

איור 16 – החלפת המשתנים לפתרון התקלה

קישור לגיט של הפרויקט:

https://github.com/Tuval1112/tau_autonomous_car_slam_liorf_fastlio/tree/main

נספחים:

	סוג מאגר הנתונים המתאים	גרסת האובונטו הרלוונטית	גרסת התוכנת הROS הרלוונטית	שימוש בIMU	שימוש בLIDAR	שימוש בGPS	שימוש ב GPU במקום ה CPU
FAST_LIO יכול לרוץ טוב עם הOUSTER Fast LIO 2		16.04 and higher	מותאם לROS1 Melodic קיימת חבילה עבור ROS2	כן	כן	אופציוניל	CPU
LIO_SAM יכול לעבוד טוב עם ה ouster והlidar one של אינוביז	Foxy, galactic	20.04	ROS1+ROS2	כן	כן	כן	אפשר להתאים ל GPU
MLOAM יכול להתממשק לכמה ליידרים מצריך כוח חישוב גבוהה	עובד עם KITTI	16.04 18.04 20.04	ROS1 וגם ROS2	אופציונלי	כן	כן	אפשר להתאים לעבודה עם GPU
A-LOAM	עובד עם KITTI	18.04 16.04 20.04	ROS Kinetic or Melodic ROS2 Humble	אופציונלי	כן	כן	אפשר להתאים לGPU
LeGo-LOAM מצריך כוח חישוב נמוך	Ground Optimization עובד עם KITTI	18.04 16.04 20.04					
Cartographer by google		20.04	FOXY/ GALACTIC ROS 1	כן	כן	כן	עובד טוב פוטנציאלית עם הGPU
DLIO		20.04	ROS2 (דרך feature/ros2) וגם ROS1	כן	כן	לא ידוע	עובד טוב עם הGPU!!
li_slam_ros2			ROS2	כן	כן	אופציונלי	
Kudan Lidar Slam- KdLidar		20.04	עובד עם ROS1 אך יכול להיות מותאם גם ל ROS2	כן	כן	כן	לא יועד לעבודה עם הGPU אך כן אפשר להתאים

טבלה 2 – השוואת אלגוריתמים קיימים

- מקורות

[1] "The Types of SLAM Algorithms",

https://medium.com/@nahmed3536/the-types-of-slam-algorithms-356196937e3d

[2] "Benefits of Lidar vs. Cameras in Self-Driving Cars",

https://www.hesaitech.com/benefits-of-lidar-vs-cameras-in-self-driving-cars/

[3] "The Types of SLAM Algorithms",

https://medium.com/@nahmed3536/the-types-of-slam-algorithms-356196937e3d

[4] "Graph SLAM: From Theory to Implementation",

https://federicosarrocco.com/blog/graph-slam-tutorial

[5] "ROS - Robot Operating System",

https://www.ros.org/

[6] "Robot Operating System",

https://en.wikipedia.org/wiki/Robot_Operating_System

[7] "ROS\ Concepts",

https://wiki.ros.org/ROS/Concepts

[8] "Lidar",

https://en.wikipedia.org/wiki/Lidar

[9] "לייזר",

https://he.wikipedia.org/wiki/%D7%9C%D7%99%D7%99%D7%96%D7%A8

[10] "What Is a Laser?",

https://spaceplace.nasa.gov/laser/en/

[11] "An Explanation of LiDAR Components and Their Functions in a LiDAR Sensor for ADAS and Autonomous Vehicles",

https://leddartech.com/app/uploads/dlm_uploads/2022/04/Tech-Note_LiDAR-Components-and-Their-Functions-in-LiDAR_V1.0_EN-1.pdf

[12] "Inertial measurement unit",

https://en.wikipedia.org/wiki/Inertial_measurement_unit

[13] "Inertial navigation system",

https://en.wikipedia.org/wiki/Inertial_navigation_system

[14] "FAST-LIO: A Fast, Robust LiDAR-inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter",

https://arxiv.org/pdf/2010.08196

[15] " github: YJZLuckyBoy/ liorf",

https://github.com/YJZLuckyBoy/liorf