The Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

C

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

מערכת הקלטה

לחיישנים לרכב אוטונומי בפלטפורמת ROS™

פרויקט מס' 20-1-1-2166

דו"ח סיכום

מבצעים:

מייקל שטילמן 319399333

בר ויצמן 301393625

מנחים:

רועי רייך אוניברסיטת ת"א

מקום ביצוע הפרויקט:

אוניברסיטת ת"א

Table of Contents

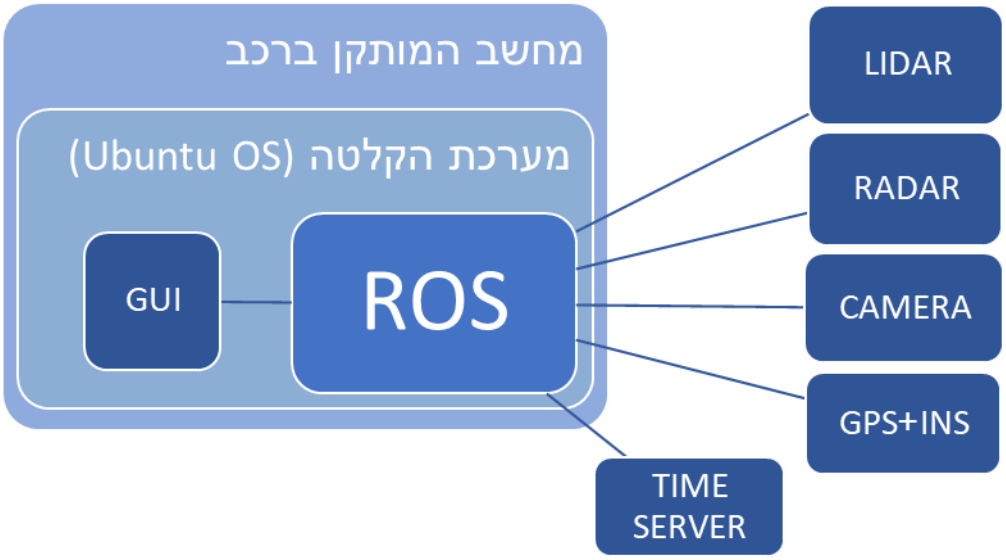
רשימת איורים

רשימת טבלאות

תקציר

בפרויקט זה בנינו מערכת הקלטה וריכוז מידע בזמן אמת, למספר של חיישנים אשר מותקנים על רכב שמיועד להיות אוטונומי. פרויקט זה אמור להוות פתרון לבעיית הנגישות של מערכת כזו. המערכת מבוססת פלטפורמת ROS (Robot Operating System), אשר מאפשרת נגישות זו, מכיוון שהיא open source, ומכילה חבילות על רוב החיישנים הנפוצים. בנוסף המחשב הנמצא ברכב מבוסס לינוקס, ובכך משפר את ביצועי המערכת.

דיאגרמת בלוקים:



איור 1: דיאגראמת בלוקים בסיסית של המערכת ההקלטה

המערכת קולטת מידע בצורה מסונכרנת מהחיישנים המותקנים על הרכב. על הרכב מותקנים לידר, רדאר, מצלמה, שרת זמן, ו-GPS הכולל גם מד תאוצה, מהירות, ועוד.

בנוסף לקליטת המידע מהחיישנים, המערכת יכולה להקליט של המידע המועבר מהחיישנים, ולאחר מכן ניתן להתבונן במידע שנאסף לצורך בקרה של נסיעות הרכב, או בעתיד, עיבוד המידע של הדרך בזמן אמת, כגון זיהוי תמרורים או רכבים אשר בסביבת הרכב.

כמו כן המערכת יכולה להציג את המידע במספר דרכים שונות, כמו להפיק גרפים, להראות הקלטה של הנסיעה ולצפות בערכי ה-INS-IMU-GPS.

בנוסף המערכת מציגה שילוב (fusion) בין המידע המופק מהחיישנים של הלידר והרדאר, ובכך מספקת לנו תצוגה תלת ממדית של החלל מסביב לרכב.

1 הקדמה

על מנת להגשים את חזון הרכב אוטונומי, נאלצנו לאסוף מידע על סביבת הרכב בזמן אמת ממספר רב של חיישנים המותקנים על הרכב:

- לידר (VLP 16 by Velodyne) [1]
- רדאר (ESR 2.5 by Delphi) [2]
- מד תאוצה-מיקום-מהירות-gps(Global - Inertial measurement unit-Inertial navigation system) Positioning System מחברת Inertial Labs) [3]
- מצלמה (Prosilica GT1930c by Allied Vision)
- שרת זמן (TA2000 by Time Machine)

סנכרון המידע מכלל החיישנים תורם לפיתוח ועבודה של אלגוריתמים שונים, אשר מנצלים מידע זה לטובת מערכות שונות ברכב האוטונומי, בנוסף מידע זה יסייע לבדיקות ולטסטים של הרכב בשלבי הבנייה והתכנון שלו.

מערכת ההקלטות של הרכב האוטונומי נבנתה על בסיס ROS, זאת משום ש-ROS הינה פלטפורמה מודולרית. הכוונה היא שעל מנת להוסיף או לשדרג חיישנים, לא יהיה צורך בבנייה מחדש של המערכת כולה, אלא בשינוי נקודתי בקוד של המודול של אותו החיישן. בנוסף ROS מתקשרת עם מגוון חיישנים שמשתמשים בפרוטוקולים שונים ומרכזת את התקשורת ביניהם למערכת אחת עם פרוטוקול תקשורת אחיד ביניהם. כמו כן מערכת זו הינה רווחת בשימוש בתעשייה היום, ונחשבת לסטנדרט בתחומים אלו ותחומי רובוטיקה בפרט.

בנוסף לכך ROS הינה מערכת חינמית ו-open-source עם קהילה פעילה, מה שתורם לזמן החיים של המערכת וגם גורם לכך שניתן לתחזק אותה בקלות רבה יותר על ידי מהנדסים שבאים מחוץ לתחום.

האלטרנטיבות שקדמו לבניית הפרויקט הינן תוכנות ל-Windows אשר באות יחד עם כל חיישן, כאשר כל תוכנה דואגת להציג מידע של החיישן אליו היא מקושרת. כמו כן נעשה שימוש ב-MATLAB לצורך סנכרון בין החיישנים השונים, אם זאת אין בהן תמיכה בכל החיישנים וגם השימוש ב-MATLAB עשוי להיות מסורבל למשתמש.

אחת הדוגמאות למערכת הקלטות הקיימת בתעשייה כיום היא Leddarvision של חברת Leddartech[10], אך החיסרון שלה היא מחיר הקנייה של המערכת וכך שהיא סגורה ולא ניתנת לפיתוח נוסף (לא open-source), משמע לא ניתן לשדרג אותה או לערוך אותה מבלי לפנות לחברה מחדש.

במערכת כפי שפעלה ב-Windows, לא היתה קיימת אפשרות לנתח את כל המידע בו זמנית של נקודת זמן כלשהיא. סנכרון המידע וה-gui החדש מאפשרים גישה נוחה לכל המידע שנאסף, ומבטיחים שחותמת הזמן של כל חיישן תהיה זהה בכל רגע.

מהסיבות המוסברות לעיל יש חשיבות רבה בפיתוח מערכת הקלטות לחיישנים ב-ROS.

2 רקע תיאורטי

2.1 מה זה ROS?

– ROS (Robot Operating System)

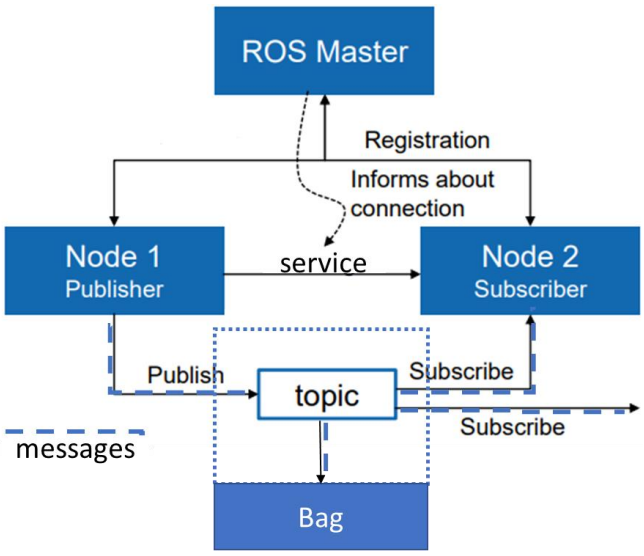
"מערכת הפעלה לרובוטים" מעניקה מסגרת עבודה (framework) גמיש ונוח לצורך בניית רובוטים ומערכת שונות, תוך כדי זה שהיא מאפשרת כתיבה מודולרית של כל חלק במערכת.

לדוגמא בפרויקט זה לכל חיישן יש "מיני-תוכנה" (node) משלו שדואג לתקשר את החיישן הנ"ל, לעבד את המידע שלו ולהעביר אותו הלאה, מבלי להסתמך על שאר החיישנים ועל מצב המערכת.

בנוסף לכך ROS מאפשרת גישה לספריות וכלים רב שימושים, אשר עזרו לנו לאורך הפרויקט, לדוגמא תוכנת Rviz: זו מאפשרת ויזואליזציה של ה-data שמתקבל מהחיישן. בנוסף לכך מאפשרת קריאת יחסי המיקום בין החיישנים והצגת תצוגה גרפית של מספר חיישנים בו זמנית על גבי אותו המרחב.

ROS כתובה בסביבת קוד פתוח ונתמכת על ידי חברת Open Robotics, למערכת זה ישנה קהילת מפתחים פעילה והמשכיות. כמו כן יצרני הרובוטים והחיישנים מכירים ב-ROS וחלקן הגדול מייצר חבילות תוכנה על מנת שניתן יהיה לעבוד עם חיישנים אלו בפלטפורמה.

2.2 כיצד ROS עובד, ומושגים בסיסיים ב-ROS



איור 2: דיאגרמת תיאור אופן העבודה של ROS

2.2.1 Master

תפקידו של המאסטר הוא לנהל רישום ומעקב של שאר הרכיבים הפועלים בזמן ריצה של ROS.

המאסטר מאפשר תקשורת בין חיישנים השונים ובין מיני-תוכנות (nodes) שונות שרצות (messages, services).

בנוסף ה-Master מחזיק את 'שרת הפרמטרים' אשר מחזיק פרמטרים גלובליים שכל מיני-תוכנה (node) יכולה לגשת עליו.

2.2.2 Node

Node – תוכנה פשוטה יחסית שבדרך כלל תפקידה הוא לבצע בפעולות וחישובים שקשורות לרכיב בודד או למטרה מאוד ספציפית.
לדוגמא, במקרה של פרויקט מהסוג שאנו ביצענו היה צורך מינימלי ב-Node עבור כל חיישן.

2.2.3 Message

Message – הדרך העיקרית בה ה-node מחליף מידע עם node-ים אחרים.
הודעה הינה מבנה נתונים (בדומה ל-Struct בשפת C), יכול להיות מורכב מ-integer, float numbers, bool, string וכדומה, ואף להכיל מערכים ולהכיל הודעות אחרות. ניתן להתייחס להודעה בודדת כחבילת מידע.
ההודעות נשלחות בפרוטוקול TCP/IP, עליו נרחיב בהמשך.

2.2.4 Topic

Topic – קו התקשורת העיקרי דרכו שnode-ים מתקשרים זה עם זה.

ה-topic מתפקד כ-bus עליו נשלחים ומתקבלים ה-message-ים. לכל node יש אפשרות לעקוב (subscribe) אחר topic, וגם לפרסם (publish) הודעות על topic בעל שם מסוים.
פעולות אלו נעשות באופן אנונימי, כלומר ה-node לא יודע מי עוד מפרסם או עוקב אחר ה-topic ולכן כתוצאה מכך הוא גם אינו יודע עם איזה node-ים אחרים הוא מתקשר.

2.2.5 Service

Service – דרך נוספת בא ה-node יכול לתקשר עם node-ים אחרים,
דרך תקשורת זאת מורכבת מ-2 הודעות (message): הודעת 'בקשה' ששולחת נתונים והודעת 'תשובה' שעונה עליהם באופן שמוגדרת בהודעות.
בשונה ממודל התקשורת ב-Topic שדומה ל-bus בו התקשורת מתבצעת בדרך כלל לכיוון אחד, והתקשרות היא בין מספר node-ים לא מוגדר, Service מבצעת תקשורת בין 2 node-ים בלבד באופן לא רציף, בסגנון של בקשה ואז המתנה לתשובה.

2.2.6 Package

Package או חבילת ROS הינה הוסף של Node-ים ו\או messages ו\או services אשר עובדים יחדיו על מנת לשרת מטרה יחידה, למשל לתקשר עם או להגדיר חיישן או רובוט מסוים.

2.2.7 Bag

Bag – הם קבצי אחסון שאפשריים לנו לשמור הודעות (message) ממספר topic-ים שונים.
ההודעות ממוינות על פי שדה הזמן של כל הודעה, ועל פי ה-Topic ממנו הם התקבלו.

ROS מאפשר לשדר מחדש את ההודעות ששמורות בקובצי bag, ויש לזה מספר יישומים כמו דיבוג מערכות, סמלוץ חיישן או תנאים מסוימים, או כמו במקרה של פרויקט זה, הקלטה וצפייה בהקלטות.

2.3 חיבורים ופרוטוקולים שונים

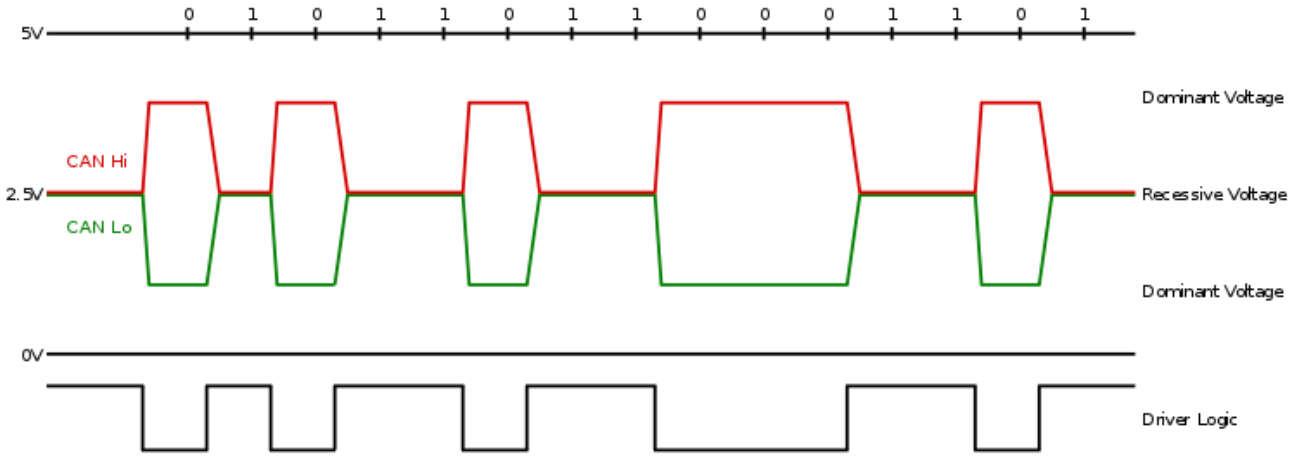
בפרויקט זה נאלצנו לעבוד עם מספר חיבורים ופרוטוקולים שונים בשביל לתקשר עם החיישנים:

2.3.1 Kvaser CAN bus

CAN bus (Controller Area Network) – פרוטוקול המאפשר תקשורת בין מספר מיקרו-בקרים והתקנים שונים, תוכנן עבור מחשבי רכבים, על מנת לחסוך בנחושת.

הפרוטוקול מורכב מ-bus מרכזי המורכב מזוג חוטים הנקראים CAN-H ו-CAN-L המאפשרים מעבר של הודעות למספר יעדים אשר נמצאים על ה-BUS.

CAN-H נע בין 3.75 ל-2.5 וולט ו-CAN-L נע בין 1.25 ל-2.5 וולט, כאשר CAN-H במצב מתח גבוהה CAN-L יהיה במצב מתח נמוך וזה יתפרש כ-0 לוגי (וההפך). הסיבה לעברת מידע בצורה זו היא לצורך מניעת רעשים ולהקטנת הסיכוי לשגיאות. בשני קצוות הקווים ישנם נגדים (בדרך בכלל בגודל של כ-120 אום) לצורך מניעת תהודה והחזרות מידע.



איור 3: תיאור אותות ה-CAN בסטנדרט ISO 11898-2 (High speed CAN).

התקשורת על ה-CAN BUS נעשית בפורמט של שליחת הודעות.

כל היעדים המחוברים ל-BUS רואים את כל ההודעות שעוברות על הקו, אך לכל הודעה יש תווית CAN Frame המודיעה על מוצאו של השולח וכך כל היעדים בוחרים אם לקבל ולהתייחס הודעה או לא. כמו כן אם מספר יעדים רוצים לשלוח הודעה בו זמנים יש לכל הודעה מספר עדיפות וההודעה עם המספר העדיפות הגבוהה יותר נשלחת ואילו היעדים שמהם לא נשלחה ההודעה ינסו לשלוח את ההודעה שוב כעבור פרק זמן מסוים.

Kvaser USB to CAN – אמצעי אשר עוזר לנו לחבר מחשב לרשת ה-CAN bus, באמצעות המרת ביטים סריאליים העוברים על ה-BUS שהוא מקבל לבתים שהמחשב יכול לעבוד איתם (ולהפך), כמו כן ה-Kvaser לוקח את תפקיד ה-CAN BUS controller, בכך שהוא מהל את הרשת ודואג לבדוק לאיזה הודעה מגיע לעלות על ה-BUS.

הרדאר של הרכב מחובר ל-Kvaser בסטנדרט ISO 11898-2.

2.3.2 RS-232

(Recommended Standard 232) הוא תקן תקשורת סריאלי (מעביר את המידע ביט אחר ביט) שהיה פעם סטנדרטי ברוב המחשבים (למשל לחיבור מודמים ומדפסות). כיום עקב מגבלת המהירות שלו, הוא פחות ופחות נפוץ.

בתקן זה יש ישנם כמה שדות, מלבד שדה המידע, אשר עוזרים ליצור תקשורת חסונה ונטולת שגיאות, והם:

ביטי הסנכרון, ביטי הזוגיות (לזיהוי שגיאות), וקצב העברת המידע (Baud rate), כמו כן ישנם מספר דרכים להעביר מידע בתקשורת סריאלית ולכן חשוב שהמכשירים היום מתואמים מבחינת פרוטוקול תקשורת.

ביטי הסנכרון –ביטים (start ו-bits stop) אשר מסמנים כל תחילה וסיום של פיסת מידע.
ביט ההתחלה מסמן את זה שהקו עבר מ-1 ל-0, וביט הסיום מסמן את זה שהקו חזר ל-1.

ביט זוגיות – בדיקת שגיאות פשוטה, כל ביטי המידע מתווספים ואז הביט משנה את ערכו על פי זוגיות של מספר ביטי ה-1, לעיתים רבות במערכות ללא הרבה רעש לא משתמשים בו כלל כדי להגדיל את מהירות התקשורת.

שימושי בטווחים רועשים, אך מאט את התקשורת ודורש תיאום נוסף בין שני הצדדים המתקשרים.

קצב העברת הנתונים (Baud rate) קובע כמה מהר לוקח למידע לעבור בקו ונמדד בביט לשניה.

קצב זה קובע את התדירות בה המכשיר דוגם את המתח בקו, ולכן על מנת ליצור תקשורת עם מכשיר נוסף עליהם לתאם קצב העברת נתונים זהה. הקצבים הסטנדרטים הינם: 1200, 2400, 4800, 9600, 19200, 38400, 57600

ו-115200 bps, קצב של מעל115200 bps לא מומלץ מאחר ורוב המיקרו-בקרים לא מתמודדים היטב איתו, נוצרים בהם איחורים בדגימה ובכך התקשורת נופלת.

חיישן ה-INS של הרכב מחובר למחשב בחיבור זה, בקופיגורציה הבא:
קצב העברת המידע בערוץ הסריאלי (Serial baud rate) הינו 115,200 bit/s, 8 ביטי מידע, ללא שימוש בביט הזוגיות וביט עצירה בודד.

2.3.3 TCP/IP וניתוב רשתות

TCP/IP הוא פרוטוקול תקשורת נפוץ באינטרנט וברשתות מחשבים, המאפשר תקשורת בין רכיבים ומחשבים במודל client\server בלי תלות בטכנולוגיה והמבנה הרשת.

הפרוטוקול מורכב משני פרוטוקולים שונים ומ-2 שכבות:
בשכבת התעבורה פרוטוקול TCP אשר מחלק את המידע לפקטות קטנות ושולח אותן לפורט מסוים, בצד השני אותו המידע מורכב חזרה, ושולח בחזרה הודעת Ack (acknowledgment) כדי להודיע שהפקטה הגיעה ליעדה בהצלחה ובשלמותה, ואם לא, הוא שולח הודעת Nak (negative-acknowledgment).

בשכבת הרשת לכל פריט (ברשת) יש כתובת IP ייחודית משלו כך שכל פקטה אמורה למצוא את דרכה ליעד הנכון.

שילוב של שני הפרוטוקולים האלו מאפשר להגיע מספר יעדים (ויישומים) שונים בכל מחשב ורכיב ברשת על ידי שימוש במספרי פורט שונים, ובכך לאפשר תקשורת בין יישומים שונים על אותו המחשב.

ניתוב רשתות – לצורך תקשורת בין רשתות שונות. לכל רשת שאינה סגורה יש כתובת gateway שדרכה ניתן להתחבר לרשת חיצונית ואפילו לאינטרנט. אך כדי לבצע ניתובים אלו בהצלחה יש לדעת להגדיר טבלת ניתובים אשר מגדירה מה הן כתובת תת-רשת וכיצד להגיע ל-gateway אם רוצים כתובת שלא תואמת את התת רשת.

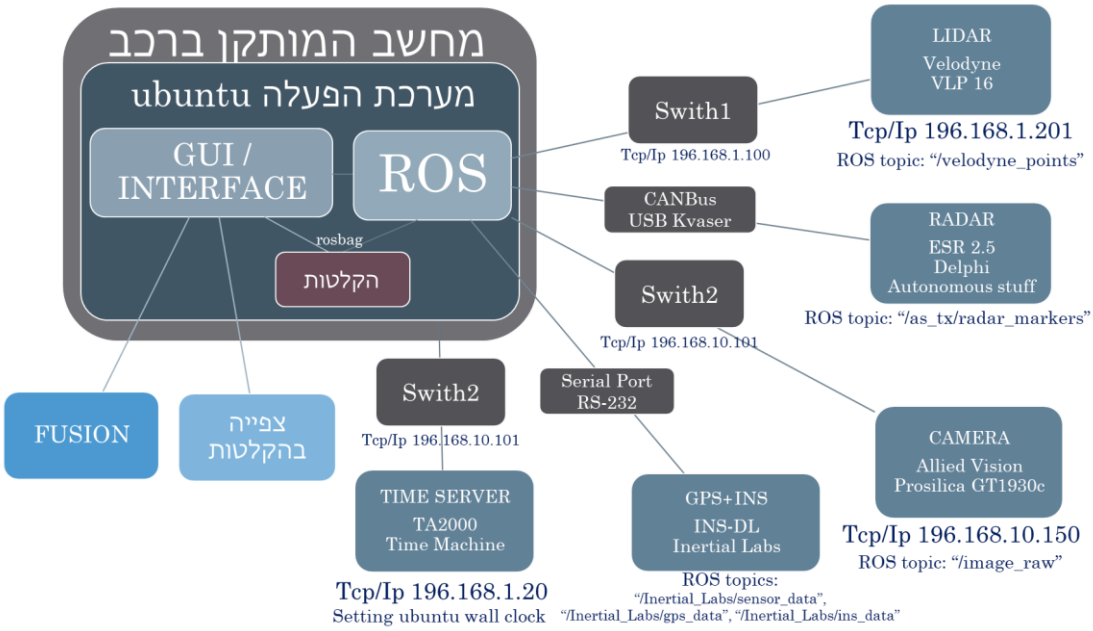
המצלמה, הלידר, ושרת הזמן מתקשרים עם המחשב דרך נתבים בפרוטוקול זה. כפי שניתן לראות לפי טבלת ניתובים המופיע בפרק המימוש (ראה טבלה מספר 1(a)).

3 מימוש

כמו שכבר ציינו מערכת ההקלטה מומשה באמצעות ROS, וה-node-ים שהיו חסרים נכתבו על ידינו ב-python. "מנוע" ההקלטה מומש באמצאות קבצי 'bag' ועל ידי שימוש בrosbag.

תצוגת משתמש (gui) בוצעה במסגרת rqt_gui של ROS, כאשר ה-widget-ים שהיו חסרים נכתבו ב-python באמצעות ספריית PyQt.

על מנת להפיק את המידע מתוך החיישנים נעזרו בחבילות ROS שנוצרו על ידי החברות שמפיצות את החומרה.



איור 4: מבנה מערכת ההקלטה

3.1 תיאור חומרה

- **מחשב -**

מעבד:

$Intel^{(R)}\ Core^{(TM)}\ i7-6700TE\ CPU\ 2.4GHz$

$RAM: 8192\ MB\ (4096 * 2)$

$Type: DDR4$

$Speed: 2133\ {}^{MT}/_{S}$

לוח אם:

$Manufacturer:\ iEi$

$Product\ Name:\ SAF3$

דיסק קשיח (לינוקס שבו מותקן ה-ROS):

$2.5"\ SATA\ SSD\ 3M$

$Size:\ 476\ GiB\ \ (512\ GB)$

דיסק קשיח ל-Windows:

$KINGSTON\ SA400S3$

$Size:\ 447\ GiB\ (480\ GB)$

- **LIDAR (VLP 16 by Velodyne)** - חיישן בעל 16 קרניי לייזר אשר מסתובבות 360 מעלות, ובכך מספקות מידע על סביבת הרכב. מותקן על גג הרכב.

מחובר על ידי נתב, נקרא לו switch1.

כתובת הנתב: 192.168.1.100

כתובת הלידר: 192.168.1.201 - ניתן גם להיכנס לכתובת זאת בדפדפן כדי לשנות את הגדרות הלידר.

- **RADAR (ESR 2.5 by Delphi)** - רדאר אשר מותקן בחלק הקדמי של הרכב, נועד כדי לזהות ולעקוב אחר אובייקטים אשר חולפים מול הרכב.

הרדאר מחובר באמצעות Kvaser USB-CAN bus למחשב (usb למחשב ו-CAN bus לרדאר) אשר ממיר את רצף ביטים הסריאלי מהרדאר לחבילות בתים שהמחשב יכול לעבד (וההפך). על מנת להתחיל לקבל את המידע מהרדאר יש לעלות על התקשורת בbus, לשלוח פקודה (פקטה) אשר מכניסה את הרדאר למצב עבודה, ואז לרדת מהbus על מנת לקבל את המידע.

במחשב הרדאר (ה-Kvaser) מחובר ל-usb

בלינוקס הרדאר נמצא ב- "can0"

- **(Inertial navigation system-Inertial measurement unit - Global System) INS-IMU-GPS**
(INS-DL by Inertial Labs) Positioning - סט חיישנים (GPS, מד תאוצה, מגנומטר, גירוסקופ) אשר מספקים מידע בזמן אמת על מיקום, תאוצה, מהירות, שדה מגנטי, מעלות סיבוב של הרכב.

במחשב ה-INS מחובר ל-COM4
בלינוקס ה-INS נמצא ב-"ttyS3"

- **CAMERA (Prosilica GT1930c by Allied Vision)** - מצלמה עם מפתח של כ180 מעלות, הממוקמת על גג הרכב, אשר משקיפה אל כיוון הנסיעה של הרכב.

מחובר על ידי נתב, נקרא לו switch2.

כתובת הנתב: 192.168.10.101

כתובת המצלמה: 192.168.10.150

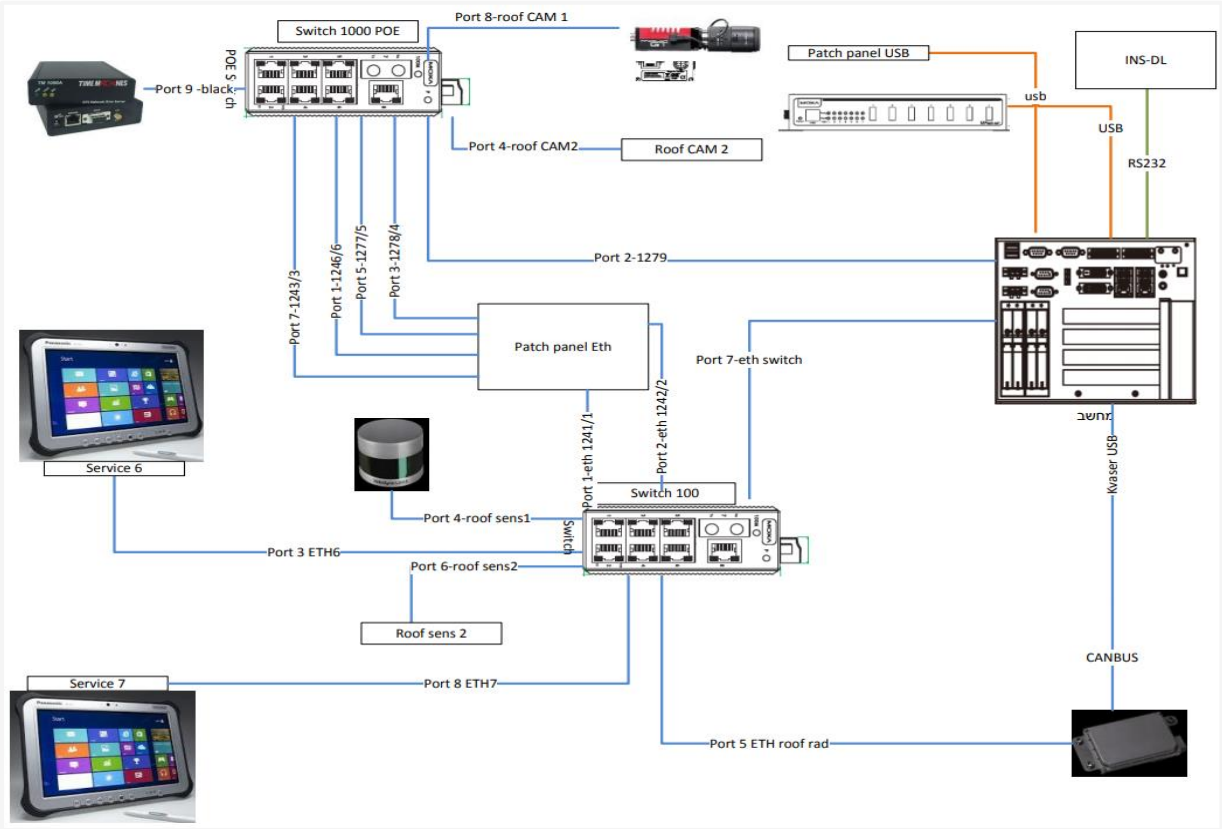
- **TIME SERVER (TA2000 by Time Machine)** - יחידת זמן אשר מספקת חותם של השעה הנוכחית לחיישנים האחרים למערכת לצורך תזמון מדידות ושאר המידע בעת פעולת ההקלטה של המערכת.

מסנכרן את השעון של מערכת ההפעלה (בהפעלת ROS הפקטות החיישנים מקבלים חותמת זמן על פי מערכת ההפעלה)

מחובר על ידי נתב switch2.

כתובת הנתב: 192.168.10.101

כתובת שרת הזמן: 192.168.1.20



איור 5: מפת החיבורים של החיישנים ברכב

3.2 תיאור תוכנה

3.2.1 הגדרת מערכת ההפעלה וסביבת עבודה בסיסית

עברנו הכרות עם מערכת ROS ובמקביל חיפשנו חבילות שתומכות בחיישנים.

ראינו שגרסת ROS הכי מתקדמת החיישנים תומכים בהם היא גרסת Melodic שעובדת במקביל ל-Ubuntu 18 ולכן על המחשב הותקן Ubuntu 18.04.

לאחר התקנת ה-ROS הגדרנו סביבת עבודה catkin של ROS, הגדרנו את הגדרות הרשת של המחשב (כזכור המחשב מחובר ל-2 switch-ים וגם לאינטרנט).

Comment	Interface	Netmask	Address
סוויצ' – 1: switch 100	Switch1- enp0s31f6	255.255.255.0	192.168.1.101
***תיאום עם הגדרות התעשייה האווירית**	Switch1- enp0s31f6	255.255.255.0	192.168.0.30
תיאום עם הגדרות התעשייה האווירית	Switch1- enp0s31f6	255.255.0.0	10.1.1.101
סוויצ' – 2: switch 1000 POE	Switch2 - enp1s0	255.255.255.0	192.168.10.100
תיאום עם כתובת שרת הזמן	Switch2 - enp1s0	255.255.255.0	192.168.1.100
תיאום עם הגדרות התעשייה האווירית	Switch2 - enp1s0	255.255.255.0	132.4.6.100
תיאום עם הגדרות התעשייה האווירית	Switch2 - enp1s0	255.255.255.0	10.1.1.100
Lidar	Switch1- enp0s31f6	-	192.168.1.201
Camera	Switch2 - enp1s0	-	192.168.10.150
Time Server	Switch2 - enp1s0	-	192.168.1.20

טבלה 1: (a) טבלת הגדרת הרשת במערכת ההפעלה, (b) טבלת כתובות החיישנים המחוברים ב-TCP/IP

***תיאום עם הגדרות התעשייה האווירית – שניתובים אלו אוזנו לצורך תיאום עם מערכת הוינדוס.**

3.2.2 התקנת חבילות ROS לחיישנים

התחלנו לנסות לתקשר (לקבל data) עם החיישנים המותקנים ברכב על ידי התקנת חבילות ROS ייעודיות.

חלק זה מפורט בצורה רחבה יותר בחוברת התיעוד שמוזכרת בסוף הקובץ.

3.2.2.1 התקנת לידר

וידאנו שהגדרות הרשת במחשב מסוגלות לתקשר עם הלידר, ולאחר מכן להתקין את האוסף חבילות של ROS עבור מוצרי velodyne ואת הדרייברים.

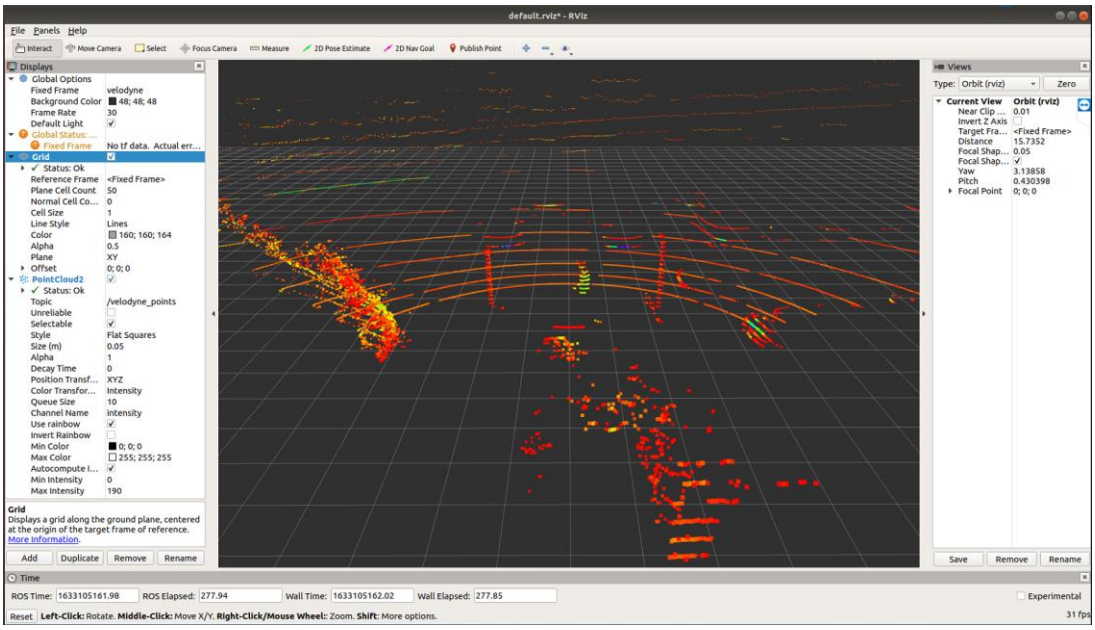
ה-Topic-ים שהיינו מעוניינים להקליט:

/velodyne_points[1] – מכיל את הוויזואליזציה של החיישן, אשר ניתן להציג בתוכנת Rviz.

כל פקטה מכילה בסביבות 250,000 בתים במערך רצוף (של בתים) אשר מכילות מידע על מיקומה של כל נקודה על הצירים xyz (4 בתים לכל ציר) ביחס למיקום הלידר, כמו כן ה"עוצמה" (צבע) של הנקודה והזמן בו היא הופיע. קצב הפרסום הינו בסביבות 10 Hz (פקטה לכל סיבוב של חיישן הלייזרים).

[1] /velodyne_points - מכיל הודעה מסוג sensor_msgs/PointCloud2

משקלו של המידע עבור נקודה בודדת הינו 22 בתים, כלומר בסביבות 11 אלף נקודות.



איור 6: תצוגה ויזואלית של ה-Topic "/velodyne_points" בתוכנת Rviz של ROS.

3.2.2.2 התקנת רדאר

ראשית הגדרנו במערכת ההפעלה את החיבור CAN-bus תחת השם can0 ככה שנוכל לתקשר לקרוא את ה-raw data שמתקבל וגם לשלוח בעת הצורך.

לאחר מכן התקנו חבילת ROS בשם socketcan_bridge, מטרת החבילה הזאת היא הפיק את ה-raw data מתוך can0 אל תוך topic כדאי ש-node עדיתי כלשהו יוכל לעבד את data.

התקנו את חבילות ה-ROS של delphi-esr, אוסף חבילות עבור רדארים של היצרן.

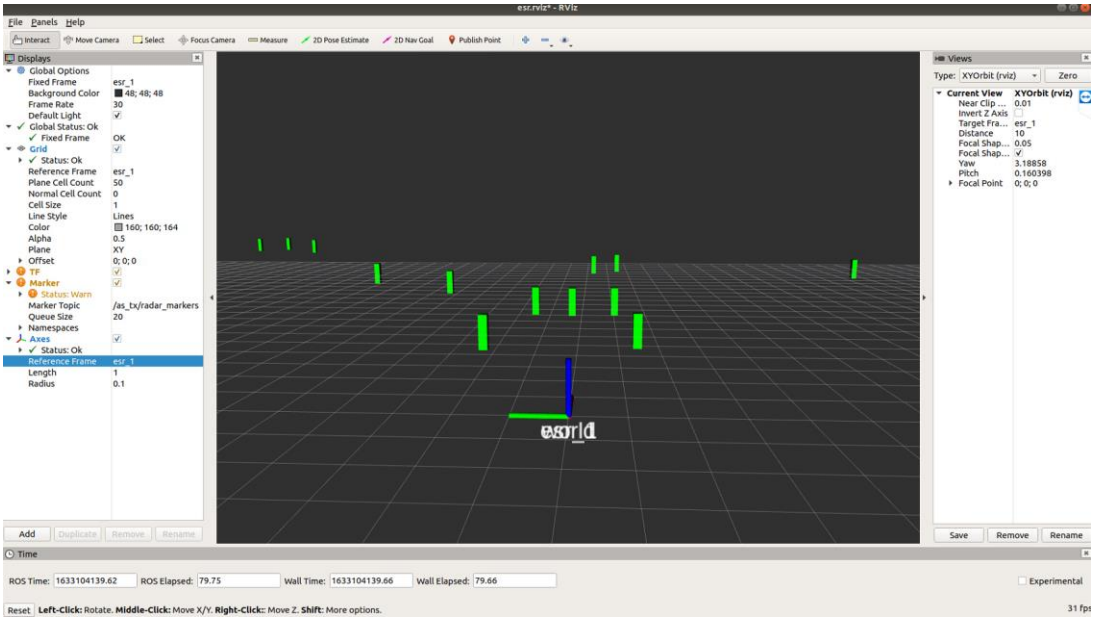
ה-Topic-ים שהיינו מעוניינים להקליט:

/as_tx/radar_markers[2] – ויזואליזציה של החיישן, מציג את האובייקטים שהרדאר מזהה. ניתן להציג בתוכנת Rviz.

כל פקטה מדווחת על מיקום, צבע ומשך הזמן הרלוונטי עבור ה-"מרקר"(אובייקט פוטנציאלי בודד שהרדאר זיהה), למשל אצלו כל ה-"מרקרים" ריבועים ירוקים, ראה איור 7.

מפני שהרדאר מחובר בחיבור can bus למחשב, ה-Node של החיישן מקבל באופן קבוע מידע רציף מה-Bus. על מנת להציג את מידע כמה שיותר מהר, יצרן החיישן בחר להשתמש בסוג של message אשר מפרסם מידע על אובייקט יחיד (נקרא "מרקר"), ולכן מתקבל קצב פרסום מהיר של כ-300 Hz.

[2] /as_tx/radar_markers - מכיל הודעות מסוג visualization_msgs/Marker.



איור 7: תצוגה ויזואלית של ה-Topic "/as_tx/radar_markers" בתוכנת Rviz של ROS.

3.2.2.3 התקנת מצלמה

וידאנו שהגדרות הרשת במחשב מסוגלות לתקשר עם המצלמה.

התקנו דרייברים של היצרן עבור מצלמות Prosilica gige, ולכן מכן את חבילות ה-ROS הבאות: prosilica_gige_sdk, prosilica_driver.

ה-Topic-ים שהיינו מעוניינים להקליט:

/image_raw[3] - ויזואליזציה של החיישן (פייד המצלמה), ניתן להציג בתוכנת Rviz ועוד דרכים רבים.

מכיל תמונה בגודל 1936x1216 פיקסלים בקידוד bayer rggb8.

קצב הפרסום תלוי ברמת החשיפה של חיישן המצלמה (בין 2 ל- 50 Hz), ניתן לקבע את הרמת החשיפה על מנת לקבל קצב פרסום קבוע.

/image_raw/compressed[4] – מכיל את הגרסה דחוסה יותר של התמונה בפייד של המצלמה.

כלומר אותה התמונה אשר מועברת ב-/image_raw אך לאחר כיווץ jpeg.

[3] /image_raw - מכיל הודעות מסוג sensor_msgs/Image

[4] /image_raw/compressed - מכיל הודעות מסוג sensor_msgs/CompressedImage

איור 8: תצוגה ויזואלית של ה-Topic "/image_raw" בתוכנת image_view של ROS.

3.2.2.4 התקנת INS

נתנו הרשאות למשתמש שאיתו אנו עובדים (administrator) של קריאה וכתיבה ל-/dev/ttyS3. זהו בעצם הפורט הסריאלי COM4 עליו מחובר ה-INS.

לאחר מכן התקנו את חבילת ROS עבור מוצרי INS של חברת המוצר inertiallabs_ins.

ה-Topic-ים שהיינו מעוניינים להקליט[5]:

/Inertial_Labs/sensor_data – מכיל את המידע הבא:

Gyro(x,y,z) , Accelation(x,y,z) , Magnetic (x,y,z) , Temprature , Input Voltage , Pressure , Barometric height.

/Inertial_Labs/ins_data – מכיל את המידע הבא:

GPS INS Time, GPS IMU Time, Millisecond of the week, Latitude, Longitude, Altitude, Heading , Pitch , Roll, Orientation quaternion, East Velocity, North Velocity, Up Velocity values, Solution status, Position STD, Heading STD, Unit Status.

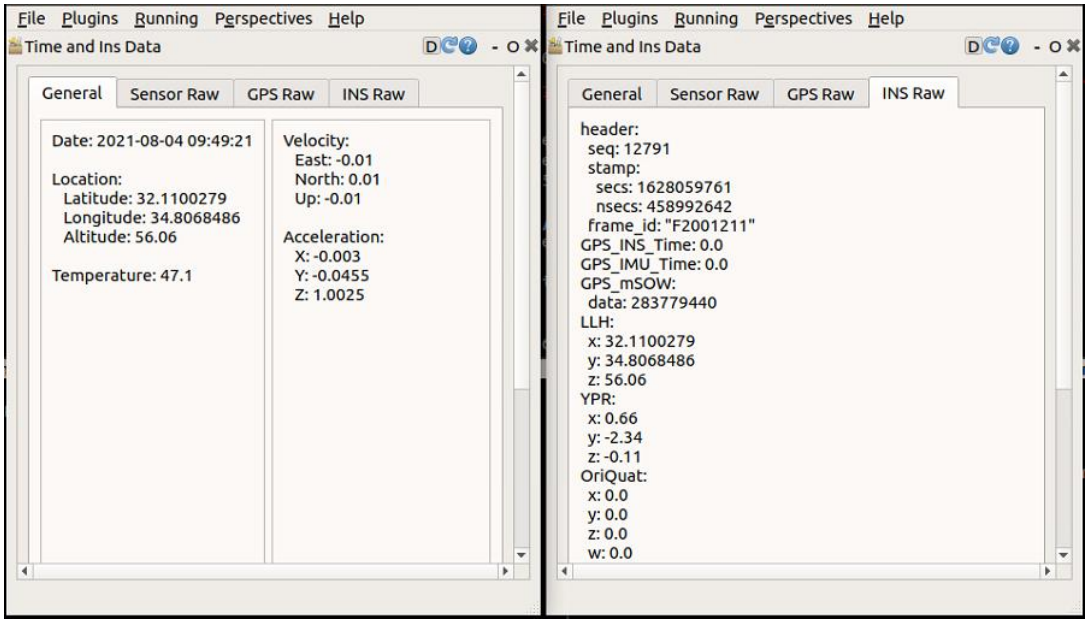
/Inertial_Labs/gps_data – מכיל את המידע הבא:

Publish Latitude, Longitude , Altitude , Ground Speed , Track Direction, Vertical Speed values.

יחדיו הם מהווים את כל המידע שהחיישן מפיק.

החיישן מפרסם את המידע בקצב של 50 Hz.

[5] קישור לפורמט ההודעות ל-Topic-ים שאנו מעוניינים להקליט בחיישן ה-INS – כאן.



איור 9: תצוגה ויזואלית של ה-Topic-ים של חיישן ה-INS בתוכנת rqt_ins_display אשר כתבו בעצמנו.

3.2.2.5 התקנת שרת הזמן

שרת הזמן לא מתחבר באופן ישיר ל-ROS, אך אנו זקוקים לו כדי לכוון את השעון של מערכת ההפעלה, שבו ROS כן משתמש על מננת לתת חותמות זמן לפקטות של החיישנים.

אנו נעזרנו בתוכנת chrony שנותנת שירות המאפשר לסנכרן את השעון של מערכת ההפעלה עם שרת NTP (שרת הזמן שמותקן ברכב).

3.2.3 GUI

לאחר שהגענו למצב בו ניתן לקבל מידע data מכל החיישנים ולהציגו בצורה ויזואלית התחלנו לבדוק כיצד ניתן להכין gui שיספק את הצרכים של הפרויקט והמשתמש, כגון צפייה נוחה בכל החיישנים, והקלטותיהם. ואפשרות לנתח מידע מכל החיישנים בנקודות זמן שונות.

לאחר כמה בדיקות ובירור עם המנחה, החלטנו ללכת על rqt_gui של ROS, זוהי פלטפורמה המאפשרת לשלב מספר widget-ים (תוכנות gui שקשורות ל-ROS) על גבי אותו הצג. ניתן לשנות את גודלו ומיקומו של כל widget ולטעון צגים עם widget-ים מוגדרים מראש.

rqt_gui משולב בהתקנת ה-ROS ומגיע עם widget-ים רבים, ובנוסף ניתן לכתוב widget-ים חדשים ומקוריים בתור חבילות ROS בשפת python ובשימוש חובה בספריית PyQt (ספרייה לכתיבת gui).

ה-widget-ים שמגיעים עם rqt_gui שימושיים מאוד, והשתמשנו בחלק מהם במימוש הפרויקט (image_view כדי לראות את הפייד של המצלמה וב-Rviz כדי להציג את הרדאר והלידר).

כמו כן ביצענו שינוים בקוד של rqt_bag widget (נותן לנו gui נוח לקבציי bag) כדי להתאים אותו יותר לצרכים של הפרויקט:

גרמנו לכך שהוא יתחיל לפרסם את ה-messages על ה-topic-ים השמורים הישר עם פתיחתו וגם שיפרסם את

ה-topic-ים השמורים תחת קידומת מיוחדת על מנת למנוע התנגשות עם מידע המתקבל מחיישנים בזמן אמת בעת צפייה בהקלטה. כמו כן הורדנו את כפתור ההקלטה משום שכתבנו widget מיוחד לזה כדי להקל על המשתמש.

כתבנו widget בשם rqt_car_sensors_recorder אשר מאפשר למשתמש לבחור בקלות את החיישנים שאותם הוא רוצה להקליט, לבחור את התיקייה, משך ההקלטה, מיקום הקובץ ושמו.

כתבנו widget בשם rqt_ins_display אשר מציג בצורה נוחה את המידע הרלוונטי של חיישן ה-INS, באופן דומה לתוכנה הייעודית של ווינדוס, וגם מאפשר צפייה ב-raw data על ידי בחירה בתגית הרצויה.

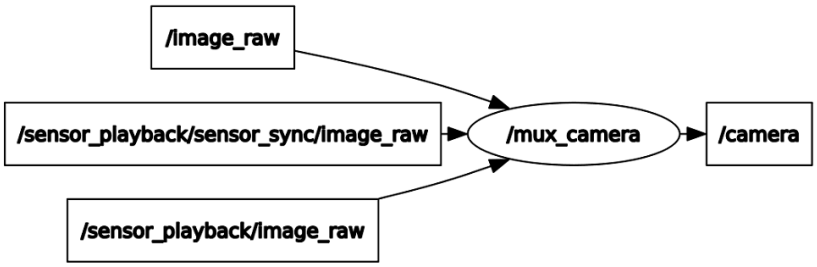
כמו כן כתבנו widget בשם rqt_feed_switch שנעזר ב-service בשם /set_display שגם אותו כתבנו, אשר בוחר אם להציג את הפייד של ההקלטה או הפייד של זמן האמת על ידי שימוש במספר MUX node עליהם נרחיב בסעיף הבא.

3.2.4 בחירת פייד לצפייה

כפי שניתן לראות בדיאגרמה שבסוף הסעיף (איור :12 דיאגרמת החיבורים של כל ה-node-ים (עגולים) וה-topic-ים (מלבנים) במערכת ההקלטה.

חץ יוצא[נכנס] מ[ל]מלבן (topic) ל[מ]עיגול (node) מסמן subscription [publication].), בשונה מה-Topic-ים של המידע שמתקבל בזמן אמת, ל-Topic-ים של ההקלטה יש קידומת מיוחדת (prefix), וזאת על מנת למנוע התנגשות בין ה-messages.

להקלטות יש קידומת של /sensor_playback, ולהקלטות המסונכרנות (עליהם נרחיב בהמשך) יש תת קידומת של /sensor_sync, משמע /sensor_playback/sensor_sync.



איור 10: גרף ה-topic-ים עבור node ה-mux. ניתן לראות כי הוא עושה subscribe ל-3 topic-ים שיש לנו עניין להציגם על המסך המשתמש, ועושה publish לאחד מהם (לפי בקשר המשתמש) עדי להציגו על המסך.

כדי לבחור להציג את ההקלטה או את המידע מהחיישנים בזמן האמת ב-GUI, השתמשנו מספר פעמים ב-mux node שמגיע בחבילת topic_tools יחד עם התקנת ה-ROS, על מנת למפות אחד מכל זוג של topic-ים (אחד בלייב ואחד הקלטה) ל-topic מוצא של ה-mux ולקרוא לו בסוג החיישן (לדוגמא: /camera או /lidar), ראה איור 10: גרף ה-topic-ים עבור node ה-mux. ניתן לראות כי הוא עושה subscribe ל-3 topic-ים שיש לנו עניין להציגם על המסך המשתמש, ועושה publish לאחד מהם (לפי בקשר המשתמש) עדי להציגו על המסך..

כל mux node מציע service בשם {MUX_NAME}/mux_select המאפשר לבחור את topic המוצא מבין ה-topic-ים שהוא עוקב אחריהם.

כדי לשנות את טופיק המוצא (בין זה של הקלטה לזמן אמת וההפך) לכלל ה-mux node-ים כתבנו node שמציע service בשם set_display/ אשר משנה את המוצא בכל ה-mux-ים בתלות ב-string שהוא מקבל ("livefeed", "recording", "sync_recording") בהתאם, כתבנו node בשם display_src_controller אשר דואג לספק את ה-service הנ"ל ולבצע את מה שתואר פה.

כל ה-widget-ים שמציגים את ה-data של החיישנים ב-GUI עובדים עם ה-topic-ים שהם המוצא של ה-mux-ים.
בנוסף הוספנו widget ל-GUI (rqt_gui) ששולח את אותו ה-service שכתבנו והזכרנו קודם בלחיצה על כפתורים מתאימים.

כעת כל מה שנדרש כדי לבחור אם לצפות בהקלטה או בזמן אמת הוא ללחוץ על הכפתור על הצג.

3.2.5 אופן ההקלטה

ההקלטה מתבצעת על ידי שימוש ב-rosbag, מצאנו כי זוהי הדרך הקלה והמועילה ביותר להקליט messages ב-ROS.
לאחר ניסיונות רבים ראינו כי דרך היעילה והבטוחה ביותר להקליט היא להריץ את rosbag דרך ה-shell עם buffer מספיק גדול (2048 MB), מצאנו כי ה-API של rosbag ל-python איטי מידיי למטרות שלנו ואף כי המימוש שנעשה בעזרו ב-rqt_bag אינו יציב (עוד סיבה לזה שהסרנו את כפתור ההקלטה מ-rqt_bag ב-GUI שלנו).

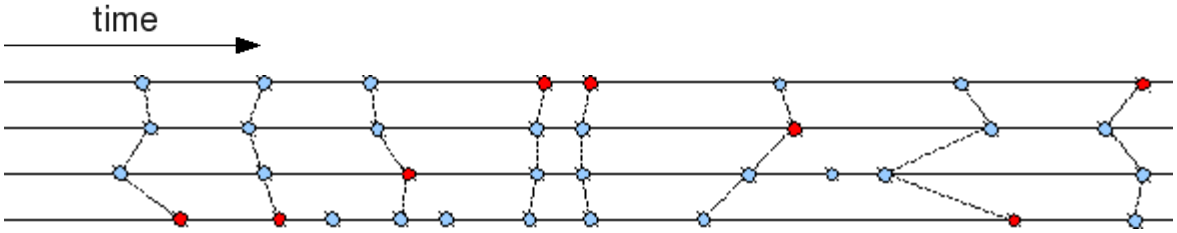
כדי להקליט עם rosbag דרך ה-shell, ה-widget ההקלטה שכתבנו (rqt_car_sensors_recorder) פותח תת-תהליך של rosbag עם הפרמטרים בהתאם לבחירת המשתמש בממשק, ואז מכבה אותו כשמשתמש רוצה להפסיק להקליט.

3.2.6 הקלטה מסונכרנת

בנוסף הוספנו את האופציה להקליט בפורמט מסונכרן, על ידי כתיבת node בשם sensors_synchronizer. בהרצת ה-node יש להזין את החיישנים שברצוננו לסנכרן כפרמטרים וכאשר כל אחד מהחיישנים הנ"ל מפרסם הודעה בפרק זמן קצר מספיק, ה-node מפרסם את אותם ההודעות ב-Topic-ים יעודים להמתחילים בקידומת "sync_". על מנת לצפות בהקלטה של אותם ה-topic-ים המסונכרנים יש לבחור בצפיית הקלטה מסונכרנת ב-rqt_feed_switch.

לכדי לממש זאת השתמשנו ב-python api של ספריית message_filters של ROS, תוך שימוש במסנן ApproximateTime Synchronizer. האלגוריתם על פיו המסנן עובד הוא דיי מורכב ומפורט באופן מלא בקישור המופיע בפרק המקורות [16].

אם לסכם בקצרה האלגוריתם מוצא pivot ובהסתמכות עליו הוא מוצא את סט ההודעות בתוך פרק הזמן שהוגדר, עובר על ההודעות בתורים (כל הודעה נמצאת בתור משלה על פי ה-topic בו היא נשלחה).
לצורך חיפוש הודעה מתאימה יותר, מניח שתגיע הודעה עתידית בזמן אופטימלי (הודעות מגיעות בפרקי זמן מינימליים) יותר ואם היא לא מגיעה אז נקבע סט ההודעות המסונכרנות.



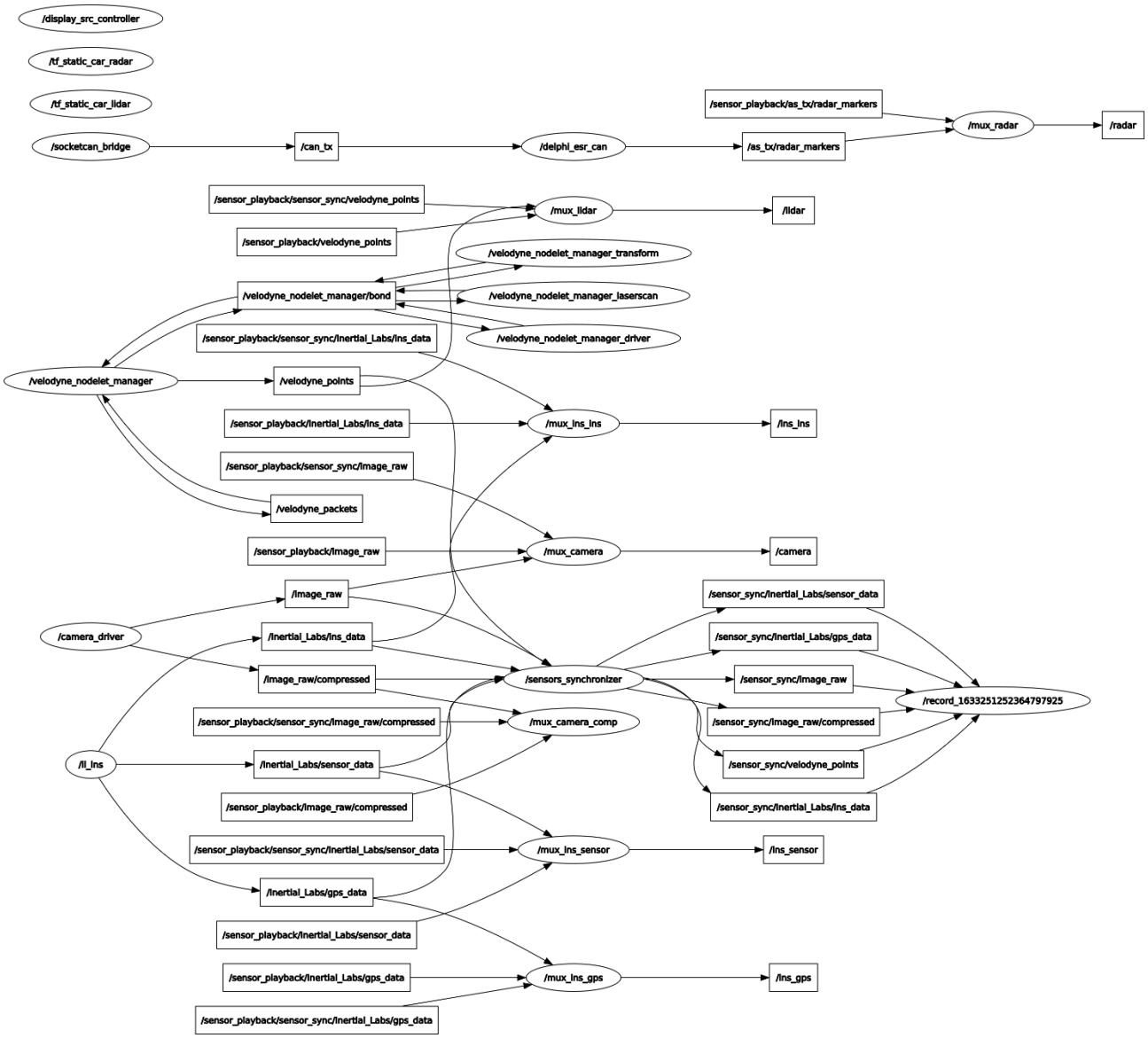
איור 11: דוגמא לביצוע האלגוריתם ApproximateTime.

כאשר כל קו מיצג topic וכל נקודה מיצגת הודעה (הנקודה האדומה היא ה-pivot) וכל קבוצה המחוברת עם קווים מקווקוים הינה סט ההודעות הנבחר.

במקרה שלנו החיישן שקצב הפרסום שלו שהכי איטי היא הלידר בקצב של כ- 10 Hz זהו גם הקצב הסיבוב של הלייזרים שלו, לכן בעקבות האלגוריתם ההודעות המסונכרנות גם הן מתאימות את עצמן לקצב של כ- 10 Hz.

חיישן הרדאר אינו נכלל בסנכרון, זאת משום שפורמט ההודעות שהוא מוציא תלוי בהודעות הקודמות שלו ולכן הוא אינו מתאים לשיטת סנכרון זאת.

3.2.7 מפת ה-node-ים וה-topic-ים של ROS במערכת ההקלטה



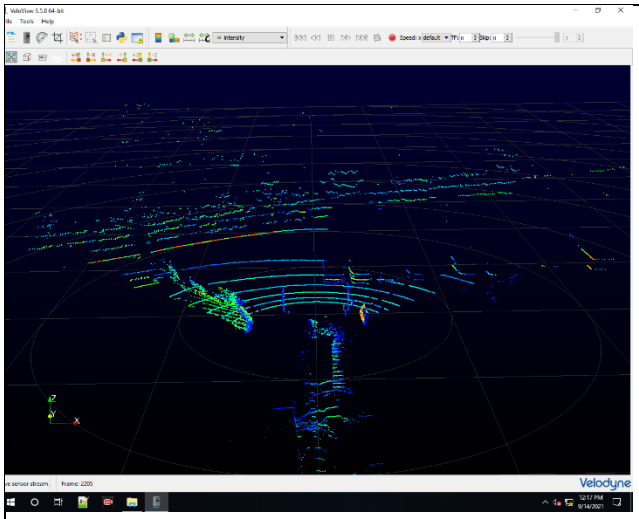
איור 12: דיאגרמת החיבורים של כל ה-node-ים (עגולים) וה-topic-ים (מלבנים) במערכת ההקלטה.

חץ יוצא[נכנס] מ[ל]מלבן (topic) ל[מ]עיגול (node) מסמן subscription [publication].

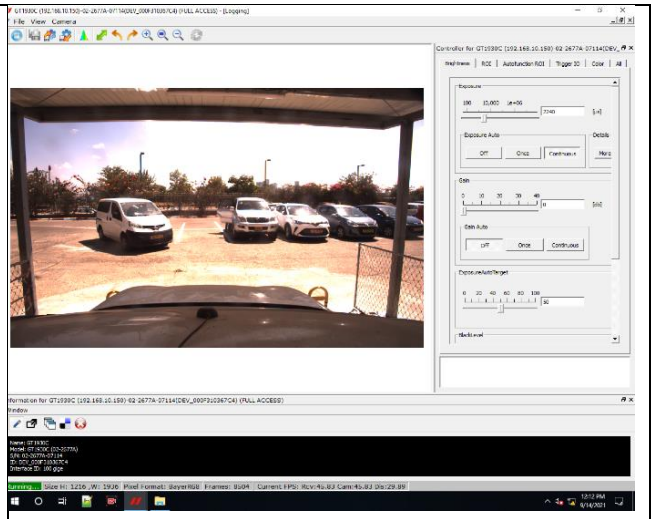
4 ניתוח תוצאות

4.1 השוואה למצב הקיים ברכב לפני ביצוע הפרויקט

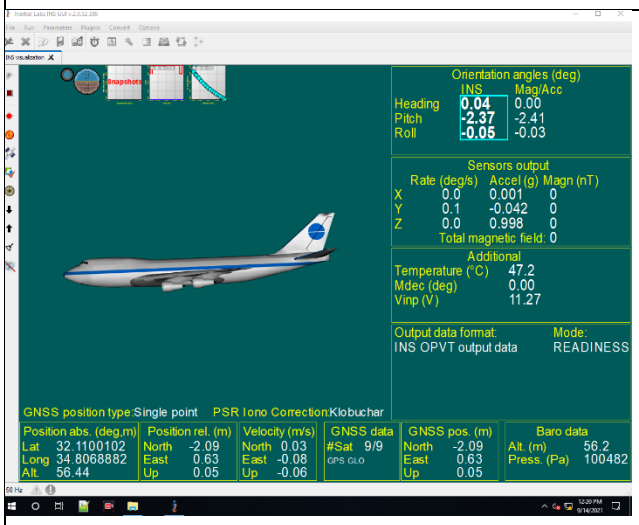
כפי שהוזכר בפרק ההקדמה, לפני תחילת הפרויקט ניתן היה לצפות בנתוני החיישנים בתוכנות ייעודיות לכך, במערכת ההפעלה windows.



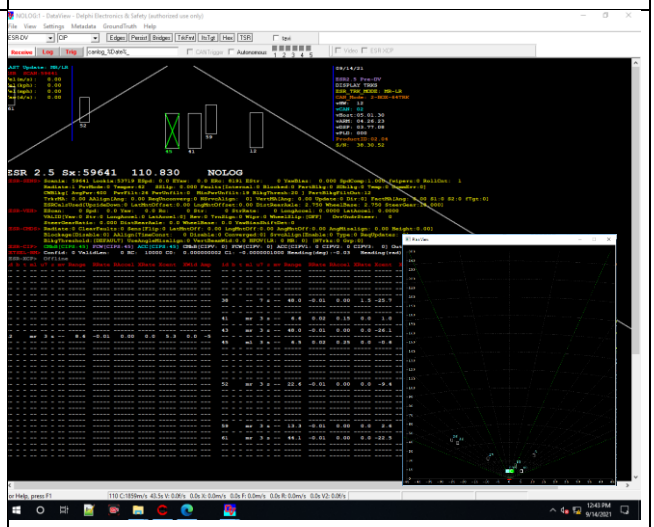
(b) מערכת הצפייה בחיישן הלידר



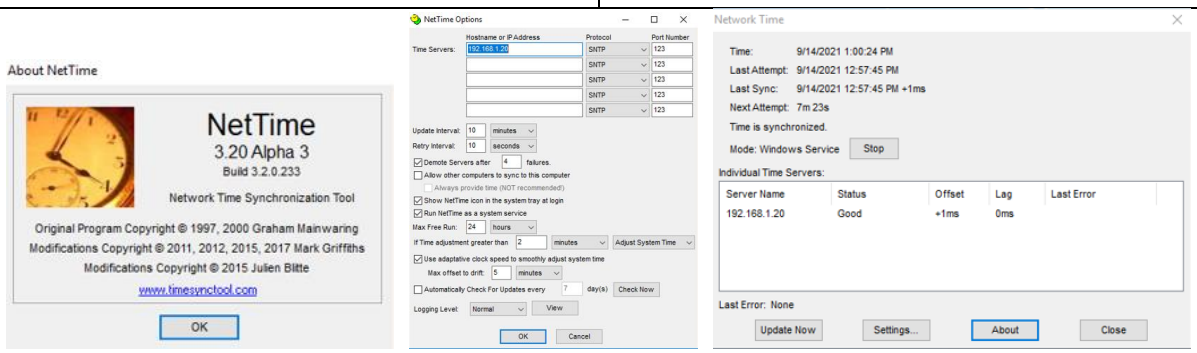
(a) מערכת הצפייה בחיישן המצלמה



(d) מערכת הצפייה בחיישן ה-INS



(c) מערכת הצפייה בחיישן הרדאר

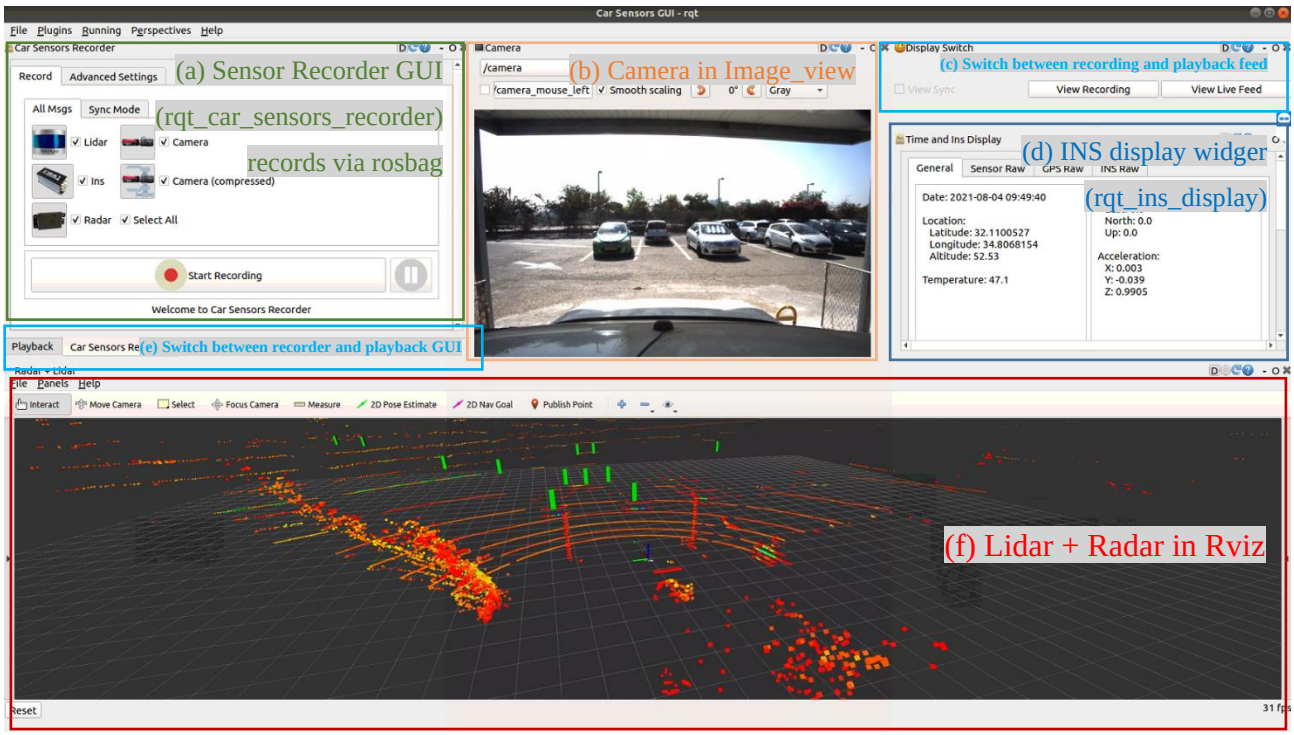


(e) מערכת סנכרון מערכת ההפעלה עם שרת הזמן

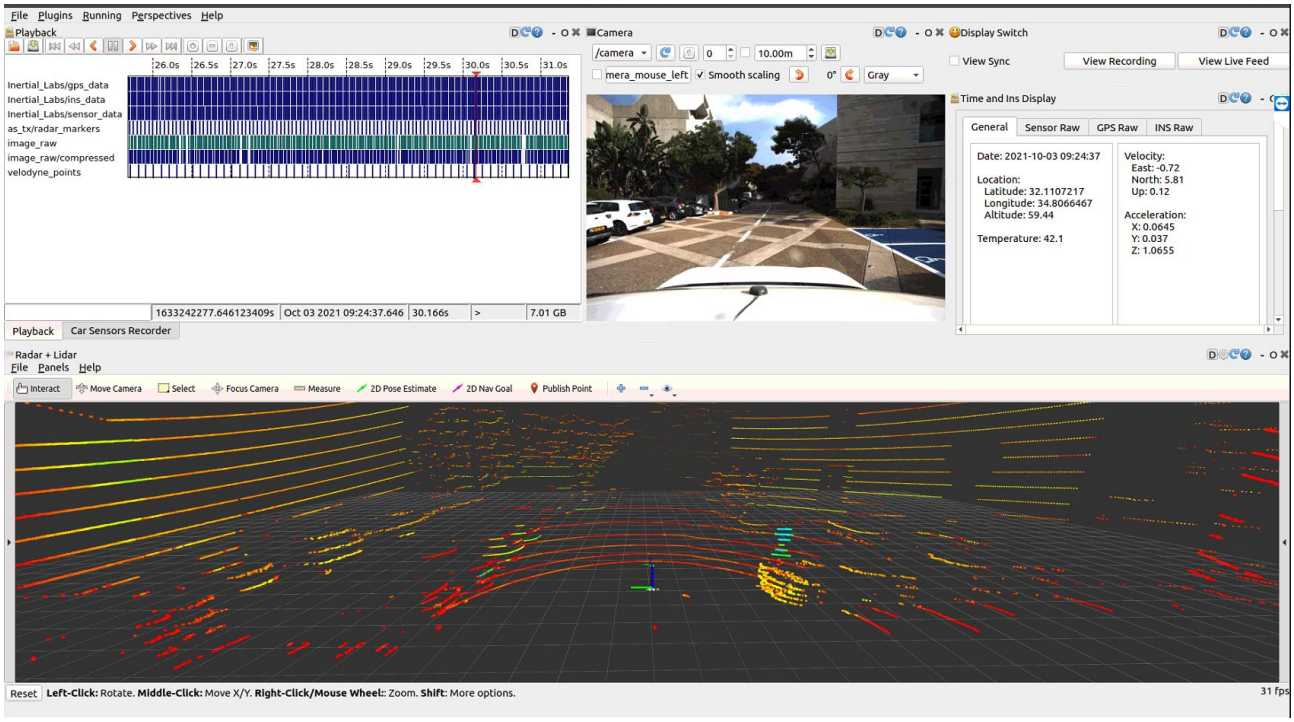
איור 13: תוכנות ה-windows לצפייה במידע המופק על ידי החיישנים.

כפי שניתן לראות נדרשו 5 תכנות שונות כדי לראות את כל המידע הרצוי, בעוד כל אחת מהן תופסת חלון.
כמו כן תהליך התפעול שלהן הוא לא מובן מעליו, למשל כדי לצפות בנתונים של הרדאר, יש לעלות על ה-can bus (ובזאת להדליק עוד תוכנה, את תוכנת ה-KVASER), ולשלוח את "פקודת הבקשה לקליטת הנתונים" כדי "להעיר" את רדאר, ואז בנוסף לזה צריך להדגיר את התוכנת הרדאר המופיע ב-איור 13: תוכנות ה-windows לצפייה במידע המופק על ידי החיישנים. כדי שתקלוט את המידע על פי הגדרות ה-can bus.

הממשק שבנינו מאחד לצפייה את כל החיישנים, ומאפשר הקלטה בצורה נוחה של כל החיישנים שנרצה להקליט. כמו כן עקב הגודל של הקלטות המצלמה, הוספנו את האפשרות להקלטת המידע מהמצלמה באיכות נמוכה יותר, מה שמאפשר ביצוע הקלטות ארוכות לצורך ניתוח מידע, שאינן מכבידות על הזכרון באותה המידה.
הקלטה בפורמט סינכרוני (Sync mode), דוגם מכל החיישנים באינטרוול שבו לכולם יש מידע, ובכך מאפשר ניתוח מידע רב משתני של יחידת זמן נתונה המגיע מחיישני הרכב (ראה איור 14: צילום מסך של מערכת ההקלטה ב-ROS (במצב צפייה לייב), עם תגיות עבור כל widget וציון החיישנים הקשורים עליו..
בנוסף ניתן לראות שחיברנו את ההצגה של הלידר והרדאר(איור 14: צילום מסך של מערכת ההקלטה ב-ROS (במצב צפייה לייב), עם תגיות עבור כל widget וציון החיישנים הקשורים עליו.), ובכך אפשרנו צפייה נוחה של סביבת הרכב.
דאגנו להציג את המידע הרלוונטי מה-INS בצורה נוחה(איור 14: צילום מסך של מערכת ההקלטה ב-ROS (במצב צפייה לייב), עם תגיות עבור כל widget וציון החיישנים הקשורים עליו.), וכמו כן ניתן להחליף את ה-feed על ידי ה-rqt_feed_switch.

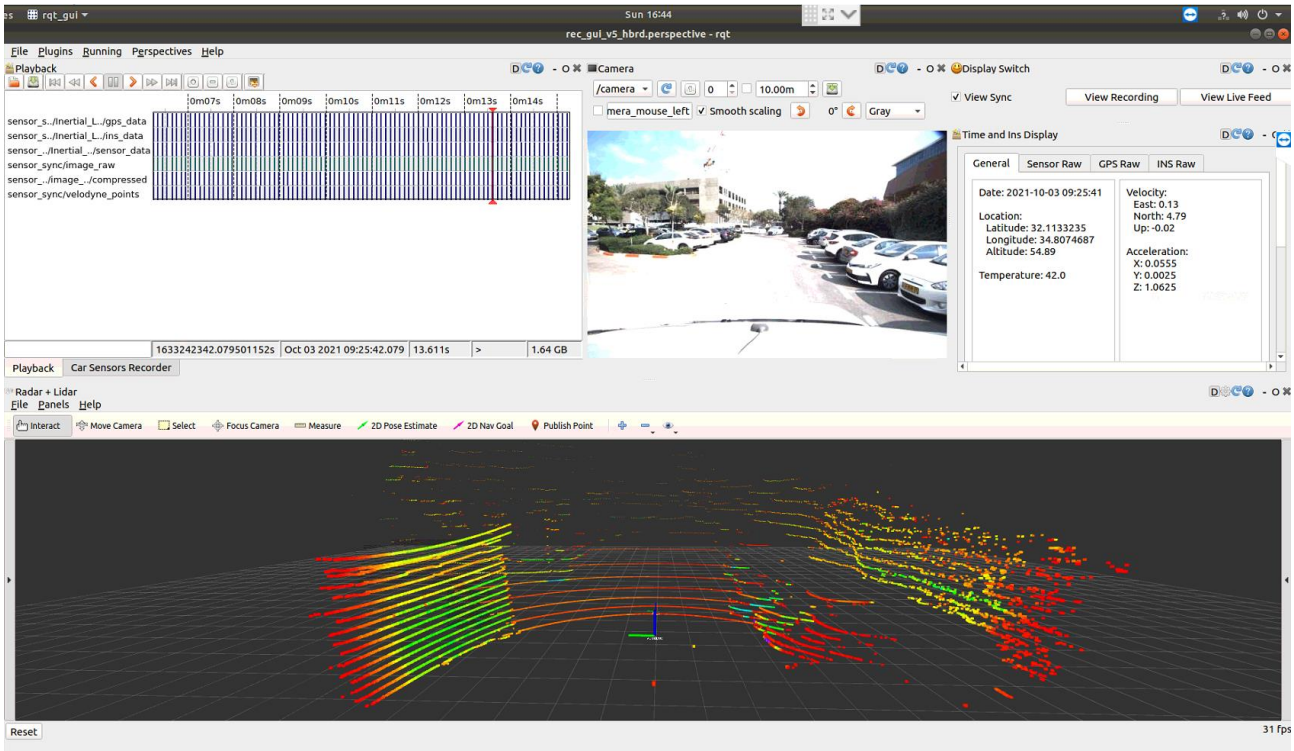


איור 14: צילום מסך של מערכת ההקלטה ב-ROS (במצב צפייה לייב), עם תגיות עבור כל widget וציון החיישנים הקשורים עליו.



איור 15: צילום מסך של מערכת ההקלטה ב-ROS (במצב צפייה בהקלטה)

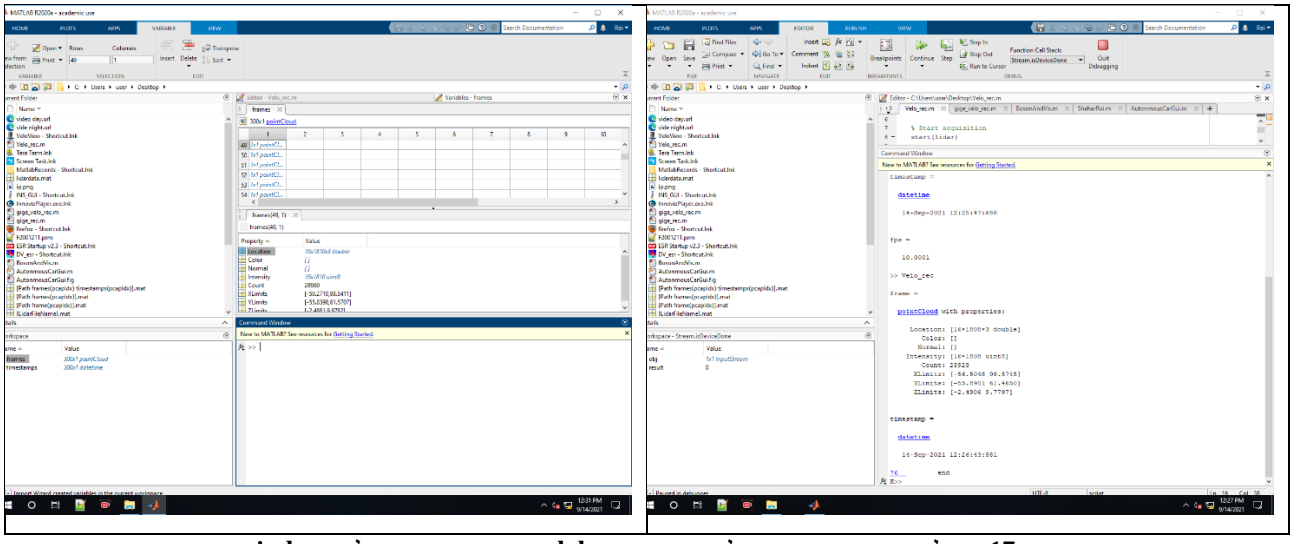
ניתן לראות כי rqt_car_sensors_recorder התחלף ב-rqt_bag שערכנו למטרות הפרויקט ביחס לצילום הקודם.



איור16 : צפייה בהקלטה שהוקלטה במצב סנכרון. פינה העליונה משמאל ניתן לראות שבשונה מההקלטה באיור הקודם, כאן ה-messages (הפסים הכחולים בציר של כל topic) מופיעים באותם ערכי ציר הזמן לעומת אלו באיור הקודם שלא הוקלטו במצב סנכרון.

כעת נשווה עם אמצעי ההקלטה שקדמו לפרויקט במערכת ההפעלה windows:

ניתן להקליט חלק מהחיישנים בעזרת סקריפט של matlab ולקבל את התוצאה כאובייקט, למשל מטריצה שאותה ניתן לשמור לקובץ.



איור 17: הקלטת המידע מחיישן הלידר באמצעות matlab במערכת ההפעלה windows.

ניתן לראות לפי האיורים האחרים בסעיף זה כי תהליך ההקלטה וחווית המשתמש הינה קלה יותר עבור המשתמש במערכת ההקלטה של ROS וכמו כן המערכת ב-ROS מקליטה את כל החיישנים בבת אחת וניתן גם להקליט במצב ששומר מדידה מכל החיישנים (חוץ מהרדאר) באותה חותמת הזמן.

ישנו יתרון להקלטה ב-matlab בכך שהוא שומר קובץ יחסית נוח לצורך ניתוח עתידי משום שהפורמט נוח לעבודה ב-matlab – תוכנה נפוצה לניתוח מידע.

כדי להגיע לתוצאה דומה במערכת ההקלטה ב-ROS, ניתן להשתמש בקובץ ה-bag (הפורמט שבו ההקלטה נשמרת), ולהעביר אותו דרך סקריפ שיחסית קל לכתוב (למשל בפייתון), אשר רק בלולאה על כל ההודעות לפי חותמת הזמן ב-topic הרצוי ודואג לסדר את המידע הרצוי לפורמט הרצוי למשתמש.

כמו כן ישנם גם חבילות ROS רבות מטפלות בסוגיה זאת, למשל אחת מהן הינה 31bag2csv .

חבילה זו מסדרת את תוכל השדות של ה-msg לפי התוכן שלהן. התקנו את החבילה הנ"ל, אך יש צורך בחבילות נוספות או כתיבת script-ים נוספים לצורך פירמוט סוגי מידע מורכבים יותר כמו ה-msg[1] של הלידר.

time	.header.se	.header.st	.header.st	.header.fr	.Mag.x	.Mag.y	.Mag.z	.Accel.x	.Accel.y	.Accel.z	.Gyro.x	.Gyro.y	.Gyro.z	.Temp	.Vinp	.Pressure	.Barometri
2021/08/0	0	1.63E+09	7.19E+08	F2001211	0	0	0	0.0015	-0.0415	0.997	-0.04	0	0	47.2	11.32	59.6	100314
2021/08/0	1	1.63E+09	7.39E+08	F2001211	0	0	0	0.002	-0.041	0.998	0	0	-0.02	47.2	11.31	59.51	100316
2021/08/0	2	1.63E+09	7.59E+08	F2001211	0	0	0	0.0015	-0.0415	0.997	-0.06	0	-0.02	47.2	11.31	59.6	100314
2021/08/0	3	1.63E+09	7.79E+08	F2001211	0	0	0	0.0015	-0.041	0.9985	0	0	-0.02	47.2	11.28	59.6	100314
2021/08/0	4	1.63E+09	7.99E+08	F2001211	0	0	0	0.002	-0.0415	0.9965	0	0.02	-0.02	47.2	11.19	59.6	100314
2021/08/0	5	1.63E+09	8.19E+08	F2001211	0	0	0	0.0015	-0.0415	0.998	0	0.02	-0.04	47.2	11.23	59.51	100316
2021/08/0	6	1.63E+09	8.39E+08	F2001211	0	0	0	0.0025	-0.041	0.998	-0.02	0.02	0	47.2	11.23	59.6	100314
2021/08/0	7	1.63E+09	8.59E+08	F2001211	0	0	0	0.0035	-0.0405	0.9975	0.02	-0.02	0	47.2	11.17	59.6	100314

איור18 : תוצאת ההמרה של ה-topic 5"/Inertial_Labs/sensor_data" לקובץ csv, תוך שימוש ב-bag2csv.

4.2 ביצועי המערכת מבחינת זמן אמת

למחשב יש קושי מסוים להתמודד עם כמויות המידע העצומות שמועבר ברגע נתון, לכן המחשב עובד באופן יותר "חלק" כאשר מקליטים כמה שפחות חיישנים.

קיבצי ההקלטה שוקלים דיי הרבה ויכולים בקלות להגיע ממעל ל-gb 7 עבור דקה של הקלטה, לכן אנו ממליצים להקליט רק את החיישנים הרצויים וגם להשתמש באופציה של לקליט את אות המצלמה בפורמט מכווץ במקום פורמט המקור (raw), דבר זה עוזר משמעותית להקטין את גודל קבצי ההקלטה.

כמו כן בהקלטה יש לוודא ש-buffer הכתיבה לדיסק (פרמטר שניתן לשנות ב-widget ההקלטה) מספיק גדול לכמות החיישנים שמוקלטים, מניסוים שביצענו לצורך הקלטות הכוללות את כל החיישנים, אך ללא תוספת ההקלטות המסונכרנות מספיק buffer של mb 2048 להקלטה באורך של 5 דקות.
כאשר rosbag (מנוע ההקלטה) חורג מגודל ה-buffer שהוקצה לו הוא מפיל את ההודעות שהוא מפספס, וכתוצאה מכך אנו ניראה רווחים ריקים בציר הזמן של ההקלטה אשר מעידים על זה שהbuffer היה מלא במשך זמן זה.

הקלטה במצב סנכרון (sync mode) גורמת לכפילות מידע, מכיוון שהיא מסננת topicים קיימים ומפרסמת את אותם ההודעות, לכן הקלטות שממשלבות הודעות מסונכרנות יחד עם כל ההודעות של חיישן דורשות buffer גדול מ-2048 (נגיד פי 2), ומאוד מכבידות על מחשב וגורמות לכך שמסך לא מגיב מספר שניות כל כמה זמן.
לכן אנו ממליצים לבצע הקלטות סנכרון נפרד מהקלטות כל ההודעות עבור מספר רב של חיישנים.

5 <u>סיכום, מסקנות והצעות להמשך</u>

5.1 סיכום מטרות הפרויקט

בספר זה הראנו כי עמדנו במטרות שהוצבו לנו:

יש לנו תקשורת של כל החיישנים גם עם מערכת ההפעלה וגם עם ה-ROS,
יש לנו GUI נוח למשתמש המציג את כל המידע הנחוץ עם יכולת הקלטה עם מצב סנכרון חיישנים ובנוסף צפייה בהקלטות שבוצעו.

כעת נשאר לנו לדון במה אפשר לשפר וכיצד ניתן להרחיב את הפרויקט.

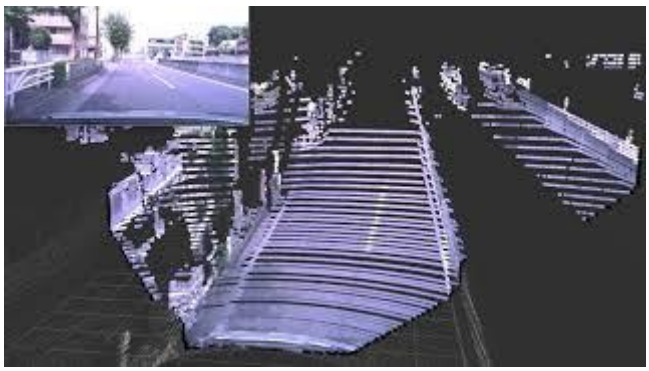
5.2 הצעות לשיפור ביצועי המערכת

ראשית תמיד ניתן לשדרג את חומרת המחשב, כלומר להגדיל את כמות הזיכרון ה-RAM, ו/או לשדרג מעבד חזק יותר. דברים אלו יקלו על המחשב להתמודד עם כמות המידע הגדולות שמועברות בכל רגע נכון של הקלטת החיישנים. אפשר גם לשדרג את הדיסק קשיח לאחד עם מהירות כתיבה מהירה יותר (בתנאי שלוח האם תומך בזה) כדי למזער את קצב התמלאות ה-buffer של rosbag בכתיבה לדיסק.

כמו כן ניתן לשדרג את rqt_car_sensors_recorder (ה-gui של ההקלטת החיישנים), כך שישנה את גודל ה-buffer בהתאם לחיישנים שנבחרו להקלטה על ידי המשתמש כדי למזער את הבעית הצפת ה-buffer.

5.3 הוספות תצוגות Fusion נוספות

ניתן לנצל יותר טוב את Rviz, תוכנת הוויזואליזציה של המידע ב-ROS והוסיף תצוגות Fusion נוספות.
לדוגמא להוסיף לתצוגה הקיים (radar + lidar) הטלה של התמונות מהמצלמה (נניח על ענן הנקודות של הלידר), לצורך קבלת תמונת מצב אף ברורה ומרשימה יותר עבור ההקלטה.



איור 19 : דוגמא ל-fussion בין המצלמה ללידר על ידי החבילה pixel_cloud_fusion[18].

5.4 הוספת חיישנים נוספים

ניתן לנצל את המודולריות של מערכת ROS, בכך שניתן באופן קל יחסי להוסיף חיישנים נוספים למערכת כדי שתקליט את המידע שהם מפיקים. חיישנים כמו מד קיברה, מצלחה אחורית וכולי.

5.5 ROS2

על מנת לשפר מעט את ביצועי המערכת ולדאוג להמשכיותה, ניתן לשדרג אותה ממערכת ROS Melodic ל-ROS2.
ROS2 עובד בצורה מהירה יותר עם מידע בזמן אמת, יציב יותר, תומך בעבודה עם Windows 10 ומערכות נוספות בנוסף ל-Ubuntu, הוא גם תומך בגרסאות חדשות יותר של C++ וב-Python3.

ROS2 בנוי כדי להעניק יותר גמישות למפתח, לדוגמא קבצי launch נכתבים בפייתון במקום בxml (פייתן שפה יותר גמישה ועשירה), ותהליך (process) יחיד יכול לעריץ בתוכו מספר node-ים לעומת node בודד.

חיסרון גודל של ROS1 הוא בזה שהמערכת הפכה להיות מוצר EOF (end of life), כלומר המפתחים הפסיקו לעדכן אותו. בניגוד לזה ROS2 מוציא גרסה חדשה כל שנה, ובכך מספק פתרונות שוטפים ליישומים עתידיים שיותקנו על המערכת.

במבט לאחור, אילו היה לנו את הידע במערכת ROS שיש לנו עכשיו כנראה שהיינו מנסים לעבוד ב-ROS2, הסיבה שבחרנו בתחילת הפרויקט ב-ROS1 היא ש-ROS1 יותר ידידותי למתחילים בכך שיש המון חומר למידה באינטרנט, וגם חבילות ה-ROS התאימו לגרסה זו בכך שלא היה צורך לערוך שינויים רבים בקוד שלהן. מכיוון שאנו סטודנטים לחוצים בזמנים החלטנו לבחור באפשרות הבטוחה יותר עבורנו.

5.6 עיבוד מידע החיישנים

5.6.1 טיפול בקבצי bag

קבצי bag הם פורמט הקבצים שבמידע שאנו מקליטים נשמר על rosbag.
כמו שכבר ציינו, המידע השמור הוא ה-message-ים של כל topic שהוקלט לפי סדר הגעת ה-massages, לכן המידע השמור לא בהכרח ישר מוכן לניתוח אנליטי, אלה במקרים מסוימים, לדוגמא ב-message מסוג pointcloud2 אשר אנו מפיקים מהלידר, יש להריץ אלגוריתם על מנת לקבל צורת מידע נוח לעבודה (למשל מטריצת נקודות עבור זמן נתון), משום שהפורמט של ה-message הנ"ל מסורבל.

ניתן לחפש באינטרנט חבילות שדואגות לטפל במקרים ספציפיים או לכתוב סקריפט פייטון אפשר ירוץ בלולאה על ה-messages ויסדר אותן לפורמט הנחוץ. בנוסף ישנם ספריות ב-matlab לטיפול בקבצי bag אשר מוציאות את המידע הנחוץ לפורמט מוגדר מראש.

5.6.2 פירמוט נתונים בזמן אמת

רעיון נוסף הוא לדאוג ליצור פורמט message אשר מראש נוח לניתוח ושימוש עתידי, בכך שניצור node אשר יעשה subscribe ל-topic הרצוי ויפרסם את המידע בפורמט הנחוץ לנו ל-topic חדש שאותו נקליט, ROS אף מגיע עם ספרייה שעושה את זה ברמה בסיסית.

בהתבסס על אותו רעיון ניתן גם לנצל את עניין הקוד הפתוח ולערוך את הקוד של יצרני החיישנים ולשנות את הפורמט ה-message שבו המידע מתפרסם כדי לקבל אף תוצאה טובה יותר משום שיהיה פחות עומס על המערכת.

5.7 רכב העתיד

אנו מעוניינים ביצירת רכב אוטונומי, ומעניין אותנו כיצד הפרויקט שלנו יתרום לכך.
אפשרנו גישה נוחה בזמן אמת לכל המידע שמגיע מהחיישנים, ובפרויקטים עתידיים יהיה אפשר להשתמש במידע הזה לצורכי ניתוח או כבסיס נתונים. למשל לשימוש בלמידה עמוקה למטרת זיהוי אובייקטים בדרך, כגון תמרורים, מכוניות, הולכי רגל או זיהוי הדרך עצמה.

בסופו של דבר המידע שאספנו יתרום ליצור כלים שיהפוך את הרכב לאוטונומי.

עשינו תיעוד של שלבי ההתקנה והתחברות לחיישנים ועוד..
לכן מי שמתעניין או רוצה לשחזר את הפרויקט מוזמן לקרוא כאן [9].

מקורות

דף נתונים של החיישן:

[1] לידר (VLP 16 by Velodyne) https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf

[2] רדאר (ESR 2.5 by Delphi) https://hexagondownloads.blob.core.windows.net/public/AutonomouStuff/wp-content/uploads/2019/05/delphi-esr-whitelabel.pdf

* על מחשב הרכב יש קובץ datasheet מפורט הרבה יותר, לא מצאנו קישור

[3] INS-DL - https://inertiallabs.com/wp-content/uploads/2020/08/INS-Datasheet.rev4_.8_August_2020.pdf

[4] (Prosilica GT1930c by Allied Vision - https://cdn.alliedvision.com/fileadmin/content/documents/products/cameras/Prosilica_GT/techman/Prosilica_GT_TechMan.pdf

[5] TA2000 by Time Machine - https://www.timemachinescorp.com/wp-content/uploads/TM2000AManual.pdf

ROS tutorials:

[6] http://wiki.ros.org/ROS/Tutorials - ROS official tutorials page

[7] https://github.com/ROBOTIS-GIT/ros_seminar - Lecture and Reference Material

[8] https://rsl.ethz.ch/education-students/lectures/ros.html - Programming for Robotics – ROS course

דוקומנטציית הפרויקט:

[9] חוברת תיעוד לבניית הפרויקט: https://github.com/Froshman/ROS_car_sensors_recorder

קישורים למקורות באינטרנט:

[10] Leddarvision של חברת Leddartech - https://leddartech.com/leddarvision

[11] מידע נוסף על פרוטוקול RS 232 https://learn.sparkfun.com/tutorials/serial-communication/all

[12] מידע נוסף על פרוטוקול RS 232 https://en.wikipedia.org/wiki/Serial_port

[13] מידע נוסף על פרוטוקול CAN BUS - https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1633138855430&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=The%20CAN%20communication%20protocol%20is,attempting%20to%20send%20a%20message

[14] מידע נוסף על פרוטוקול CAN BUS - https://en.wikipedia.org/wiki/CAN_bus

[15] מידע נוסף על פרוטוקול TCP/IP - https://en.wikipedia.org/wiki/Internet_protocol_suite

[16] מידע מפורט על האלגוריתם עליו התבסס הסנכרון הודעות - http://wiki.ros.org/message_filters/ApproximateTime

[17] קישור ל-github של bag2csv - https://github.com/AtsushiSakai/rosbag_to_csv

[18] עמוד דוקומנטציה של חבילת pixel_cloud_fusion - https://autoware.readthedocs.io/en/feature-documentation_rtd/DevelopersGuide/PackagesAPI/detection/pixel_cloud_fusion.html