פרויקט AutoRaceCar

עדי מזוז: 054-2591059
דימה קולטונוב: 052-2987612
אסף אנטר: 0507116501

תוכנן עניינים

מטרה

בניית פלטפורמת פיתוח לרכב אוטונומי הכוללת קבלת מידע מחיישנים שונים, יכולת הוספת חיישנים, שימוש בכלי פיענוח ושילוב אלגוריתמים.

יעדים

- ❖ תכנון ובניית הפלטפורמה – מכנית וחשמלית.
- ❖ סקר ספרות ובחירת כלים לכלל המערכת:
  - חיישנים בסיסיים: מצלמה, חיישן תנועה, בקר מהירות, מיקרו קונטרולרים ומחשב מרכזי.
  - סביבת עבודה ומערכת הפעלה.
  - שפת פיתוח וספריות שונות.
- ❖ ממשק מצלמה:
  - הוצאת תמונות: תמונת צבע, תמונת אינפרה ואדום ותמונת עומק.
  - הוצאת נתוני gyro: מהירות זוויתית לשלוש דרגות חופש.
  - הוצאת נתוני תאוצה: תאוצה קווית לשלוש דרגות חופש.
  - נתוני intrinsics ו- extrinsics על הרכיבים השונים במצלמה.
- ❖ ממשק מנוע:
  - כתיבת דרייבר הצרוב על מיקרו קונטרולר(ארדואינו ננו) המאפשר קבלת פקודות מגורם חיצוני ושליחתם למנועים.
  - כתיבת ממשק חיצוני המאפשר תקשורת מול הארדואינו ושליחת פקודות למנועים.
- ❖ ממשק חיישן תנועה:
  - כתיבת דרייבר הצרוב על מיקרו קונטרולר(ארדואינו אונו) המאפשר קריאת מידע והחיישן והעברתו לגורם חיצוני.
  - כתיבת ממשק חיצוני המאפשר תקשורת מול הארדואינו וקבלת המידע.
- ❖ ממשקי תקשורת(tcp):
  - ממשק client וממשק server המאפשרים העברת נתונים בין מחשבים.
- ❖ ממשק גרפי:
  - ממשק המציג בזמן אמת את כל נתוני החיישנים: תמונה, מהירות זוויתית, תאוצה קווית, תנועה ומהירות קווית וכמו כן, בעל יכולת תקשורת מול מחשב מרוחק.

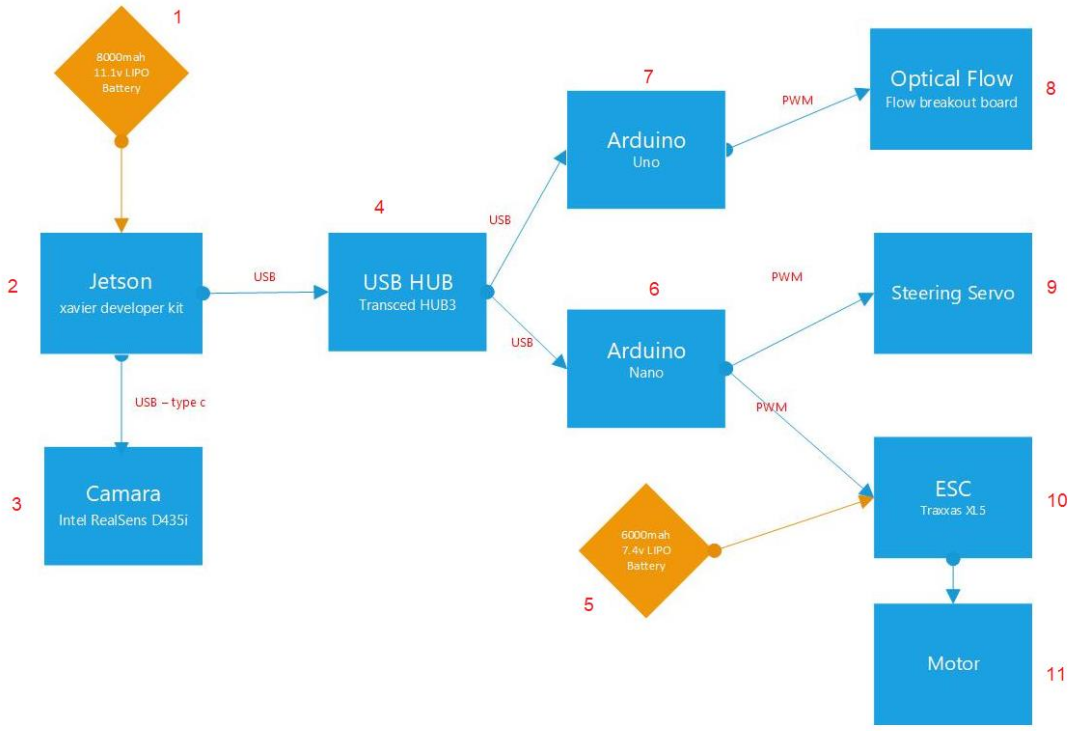
ניהול גרסאות

כל הפרויקט מנוהל ב github בהרשאת **private** כיוון שכל הפרויקט שייך לטכניון.
ניתן לעשות clone לפרויקט דרך הקישור:
https://github.com/asafanter/AutoRaceCar.git

ממליצים להמשיך ולנהל את הגרסאות דרך הגיט גם לדורות הבאים כדי שיהיה מקום יחיד שנגיש לכל הפרויקט וגיבוי מלא.

סרטון הדרכה נחמד למתחילים:
https://www.youtube.com/watch?v=SWYqp7iY_Tc

תיאור המערכת

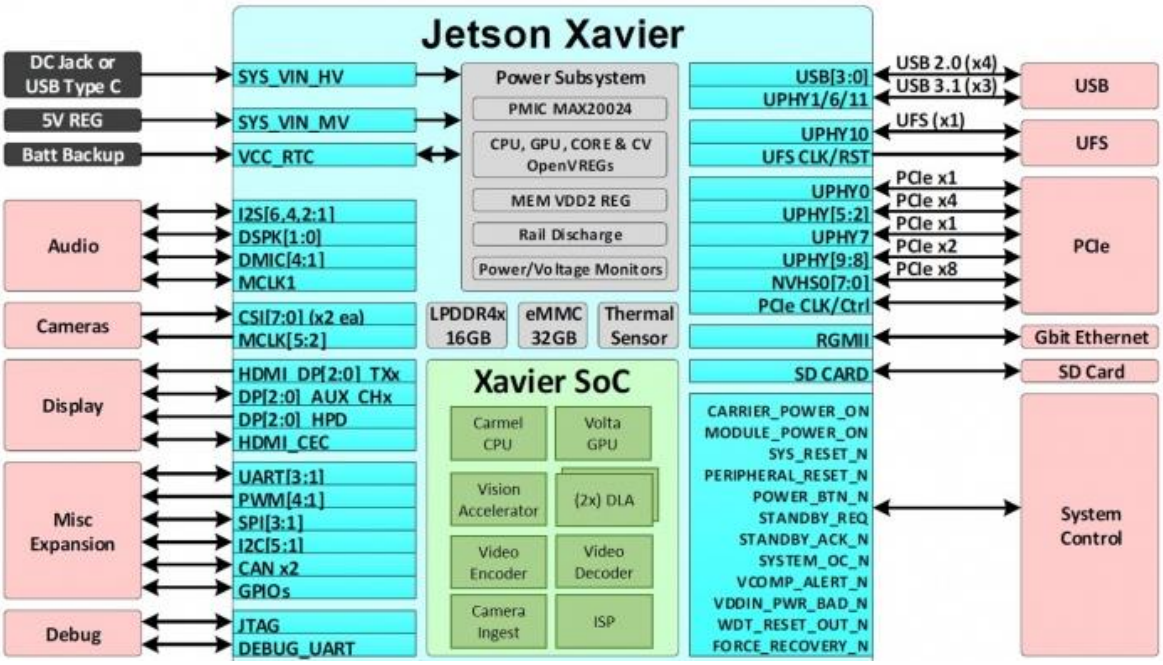


המערכת מורכבת ממחשב מרכזי, xavier(2) אשר מקבל מתח מספק(1).
אל ה- jetson מחוברים מאופן ישיר מצלמה(3) ומפצל usb(4).
המפצל usb מכיל 4 יציאות ואליו מחוברים שני מיקרו קונטרולרים, Arduino uno(7) ו – Arduino nano(6), בנוסף עוד שתי יציאות אלינן ניתן לחבר מקלדת ועכבר.
אל ה - Arduino uno מחובר החיישן תנועה optical flow(8).
אל – Arduino nano מחוברים servo(9) אשר שולט על הגלגלים ובקר מהירות(10) אשר שולט על המנועים(11) המקבלים מתח מסוללה(5) כאשר רכיבים 9,10,11,5 הם רכיבים אשר גלולים בחבילת הרכב.

סוללת מערכת

כמות: 1.
מתח: 11.1[V].
אנרגיה: 8000[mAh].

מחשב מרכזי (xavier)



מפרט:

GPU: 512-core Volta GPU with Tensor Cores
CPU: 8-core ARM v8.2 64-bit CPU, **cache:** 8MB L2 + 4MB L3
Memory: 16GB 256-Bit LPDDR4x **throughput:** 137GB/s
Storage: 32GB eMMC 5.1
Power: 10W / 15W / 30W
PCIE: 1 x8 or 1 x4 or 1 x2 or 2 x1 PCIe (Gen4)
Size: 105x105x60(mm)
WI-FI: NO

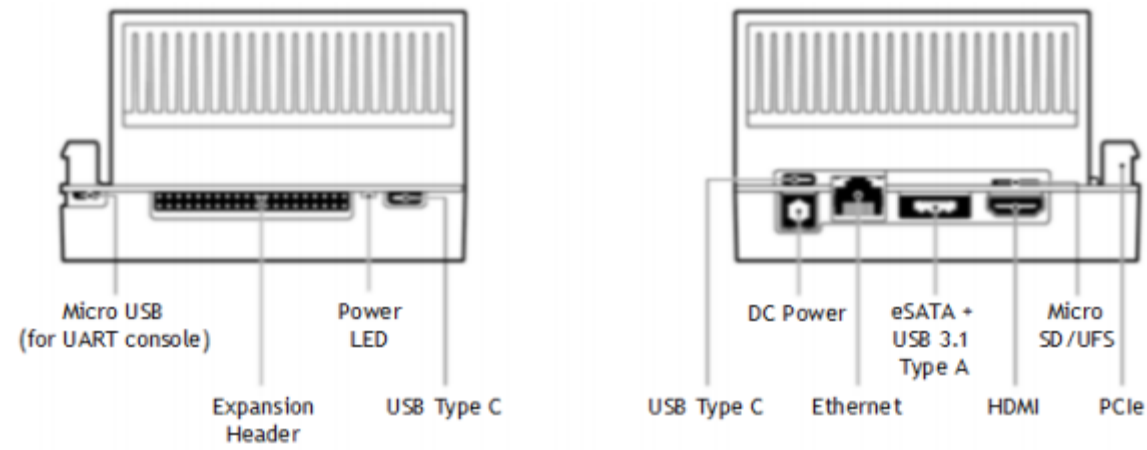
- מפרט מפורט ויותר מעמיק על החומרה בjetson ניתן למצוא פה:
https://developer.nvidia.com/embedded/develop/hardware
- **מכיוון שאין רכיב WIFI מובנה נאלצנו לחבר רכיב חיצוני שעליו מחוברות האנטנות.**
- הרכיב כולל (**DLA**) Deep Learning Accelerators ו Visual Accelerators (**VA**) .
- מכיוון שהרכיב כולל הרבה רכיבים תומכים (כמו מאיצים, GPU) לרכיב ישנם 4 רמות לצריכת הספק ב7 MODEי פעולה שונים הסבר על כך ניתן למצוא כאן:
https://www.jetsonhacks.com/2018/10/07/nvpmodel-nvidia-jetson-agx-xavier-developer-kit/

בנוסף על הכרטיס פיתוח (**Developer Kit**) ישנם כניסות וממשקים לעולם החיצוני (I/Os):

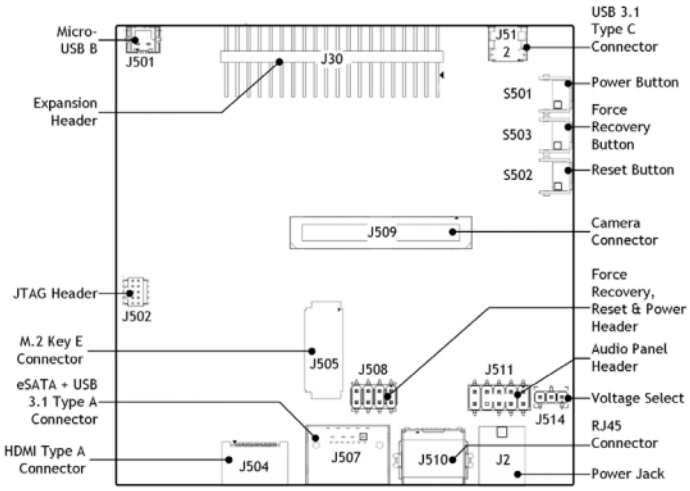
PCIe X16: x8 PCIe Gen4/x8 SLVS-EC
RJ45: Gigabit Ethernet
USB-C: 2x USB 3.1, DP (Optional), PD (Optional) Close-System Debug and Flashing Support on 1 Port
Camera Connector : (16x) CSI-2 Lanes **(not in use with realsense camera)**
40-Pin Header: UART + SPI + CAN + I^2C + I^2S + DMIC + GPIOs
eSATAp+USB3.0 Type A : SATA Through PCIe x1 Bridge (PD + Data for 2.5-inch SATA) + USB 3.0
HDMI Type A/eD/DP: HDMI 2.0, eDP 1.2a, DP 1.4
usD/UFS Card Socket: SD/UFS

Developer Kit Interfaces

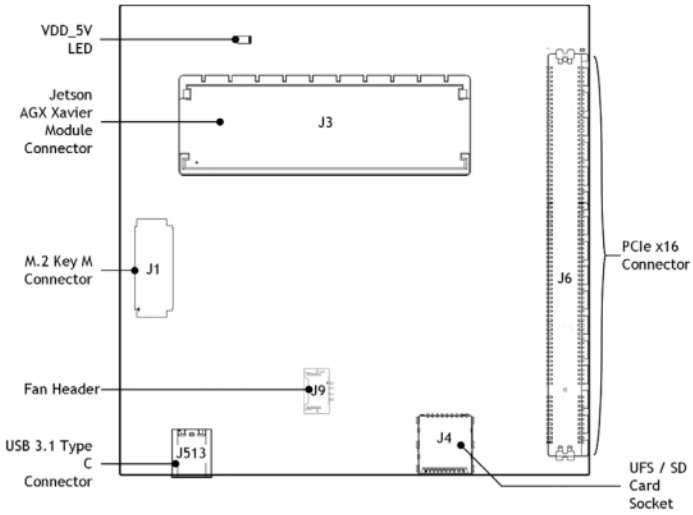
Front view (left) and rear view (right)



Bottom view of developer kit carrier board



Top view of developer kit carrier board

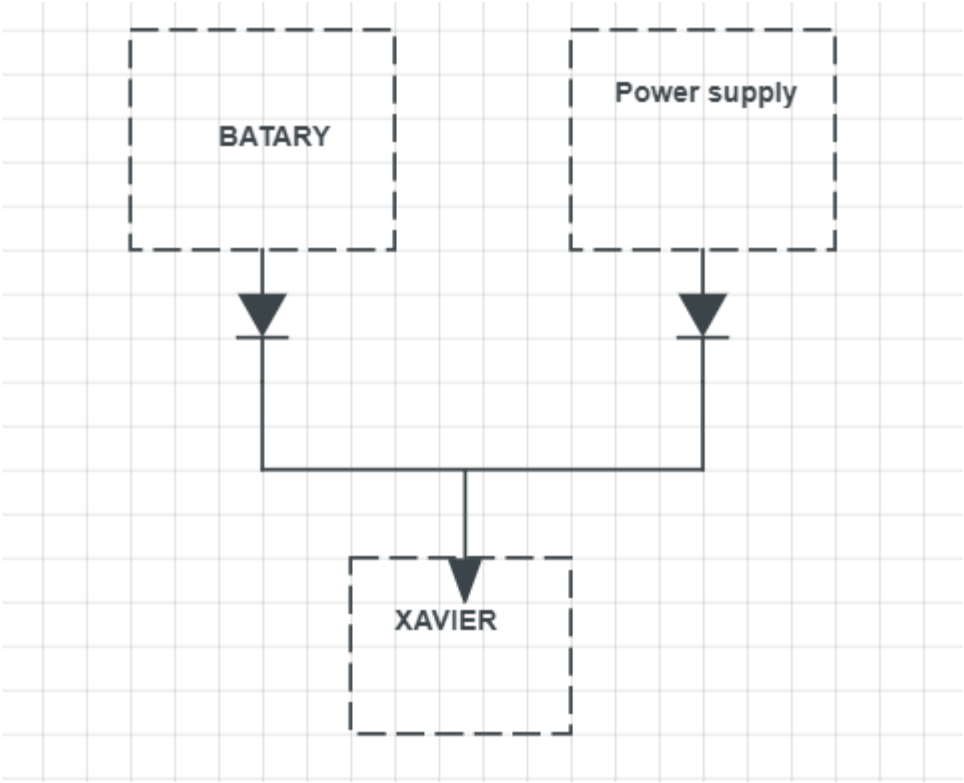


התאמות חומרה שנעשו לxavier :

- הוספת רכיב WIFI חיצוני.
- הרכבנו כבל שמאפשר לחבר את ה xavier למתח קיר ולסוללה מבלי לכבות את הרכיב במעבר בין הספקים.

מכיוון שהסוללה שמספקת את ה xavier מספקת 11.1v (כאשר לא מחוברים למתח קיר) והכבל המקורי שמגיע עם הxavier הוא עם ספק של19v ו1.5A, נתקלנו בבעיה של קפיצת מתח גדולה מידי וכתוצאה מכך בעת החלפת ההזנה ה xavier היה קורס. לכן החלפנו את כבל ההזנה לכבל עם ספק של v12 .

הכבל בנוי מ2 דיודות (ברכיב אחד) אשר מעביר את הזרם רק לכיוון הxavier , או מהקיר או מהסוללה.



***שימו לב:** הxavier לא נבדק בביצועים גבוהים (שימוש במאיצים וכוח חישוב גדול) עם הכבל שחיברנו לכן מומלץ להחליף לסוללה מתאימה יותר ואז לחזור לעבוד עם הכבל המקורי.

הפעלת מערכת:

בעת הפעלת המערכת יש להריץ את הפקודות הבאות:

- לביצועיים מרביים של הxavier יש להריץ:

```
$ sudo nvpmodel -m 0
```

מוד זה מפעיל את הxavier ללא הגבלת הספק וכל רכיב מוציא את הנדרש ממנו בזמן הריצה.

- להפעלת המאוררים של הרכיב יש להריץ:

```
$ sudo ./jetson_clocks.sh
```

- להגדלת הbuffer של ה USB (כדי לא לעבד מסגרות מהמצלמה\ לקריסת המצלמה) יש להריץ:

```
$ sudo gedit /sys/module/usbcore/parameters/usbfs_memory_mb
change 18 to 4000 and save in the file
```

*הסיבה שיש להגדיל את החוצץ ידנית היא שה xavier לא תומך כמו שצריך במצלמת הrealsense שאנו עובדים איתה (מצלמות שכן נתמכות ע"י nvidia הגדלת הbuffer נעשה באופן דינמי ואוטומטי).

לינקים שימושיים לגבי הxavier :

מיפוי GPIO

https://www.jetsonhacks.com/nvidia-jetson-agx-xavier-gpio-header-pinout/

פירוט על מאיצים שישנם על הxavier וארכיטקטורת מעבדים של הרכיב:

https://en.wikichip.org/wiki/nvidia/tegra/xavier

הסבר על עיצוב התרמי של הxavier והסבר על פיזור ההספק\חום ברכיב:

https://static5.arrow.com/pdfs/2018/12/12/12/22/1/565659/nvda_/manual/jetson_agx_xavier_thermal_design_guide_v1.0.pdf

מצלמות מומלצות לxavier :

https://www.e-consystems.com/nvidia-jetson-camera.asp

מצלמה

Intel RealSense depth Camera D435i

מפרט:

Image Sensor Technology : Global Shutter, 3um x 3um pixel size.

Maximum Range: Approx. 10 meters depends on calibration, scene and lighting condition.

Depth Technology: Active IR stereo, Field of View: $87^\circ \pm 3^\circ x 58^\circ \pm 1^\circ x 95^\circ \pm 3^\circ$

Minimum depth distance: 0.105m,

Depth output resolutions and ftame rate: up to 1280x720 and up to 90 fps (with USB3 cable).

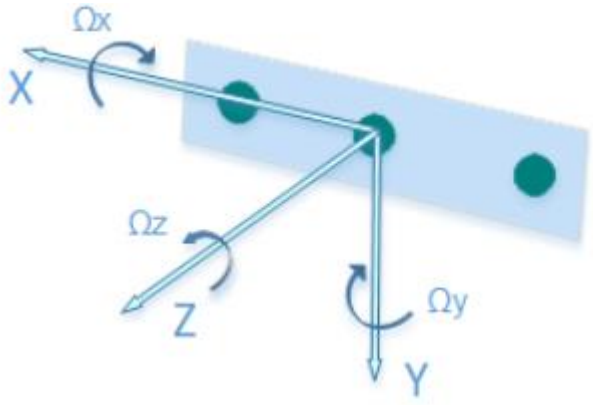
RGB: resolution and frame rate: up to 1920x1080 and 30fps

RGB sensor field of view: $69.4^\circ x 42.5^\circ x 77^\circ (\pm 3^\circ)$

Connection: USB-C 3.1 Gen1

Power Req.: 5v, 0.7A

The inertial measurement unit (IMU) : used for the detection of movements and rotations in 6 degrees of freedom (6DoF). An IMU combines a variety of sensors with gyroscopes to detect both rotation and movement in 3 axes, as well as pitch, yaw and roll.



1. The positive x-axis points to the right.
2. The positive y-axis points down.
3. The positive z-axis points forward

- יחדות ה acceleration הם $meter/sec^2$
- יחידות הgyro הם radian/sec
- עבור מצלמה זו יחידת הIMU אינה מכוילת לכן קיים כלי כיול שהוא חלק מהSDK , כמו כן מתייחסים למידע מהIMU כאל stream (כמו stream של מסגרות ממצלמה). הIMU עובר כיול במהלך הפעלת הstream ופרמטרי הintrinsic של הIMU מחושבים ושמורים לגישה במהלך הפעלה זו.
- פרמטרי ה extrinsic של depth-IMU שמורים כקבועים במערכת כי תלויים במיקומים הפיזיים של הרכיבים.
- מידע נוסף וחשוב עבור יחידת הIMU ניתן למצוא כאן:

https://www.intelrealsense.com/how-to-getting-imu-data-from-d435i-and-t265/

- מסמך Datasheet מפורט שבנוסף מסביר לעומק על אופן הפעולה של מצלמת העומק, וחושף את הפונקציונליות של המצלמה:

https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf

- בנוסף הסבר עם נוסחאות על משמעות כל פיקסל במפת העומק ניתן למצוא כאן:

https://github.com/IntelRealSense/librealsense/tree/master/tools/depth-quality

- כמו כן קיימת יחידת POSE שבעזרתה ניתן לעשות מעקב למסלול, לא השתמשנו באופציה זו בפרויקט שלנו כלל, ממליץ להעמיק בנושא.

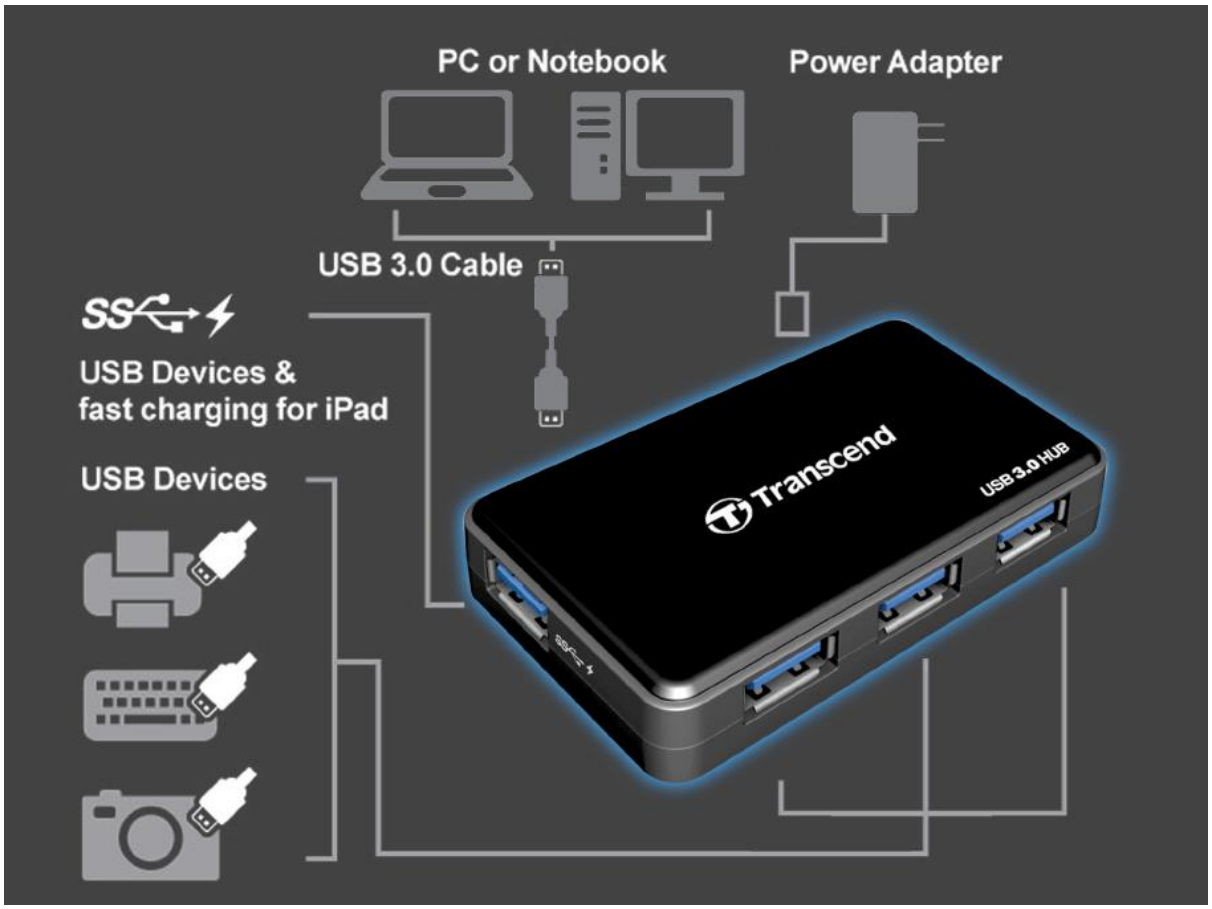
קישורים שימושיים:

גיט רשמי עם הרבה דוגמאות ותיעוד על אופן פעולת המצלמה נמצא

https://github.com/IntelRealSense/librealsense

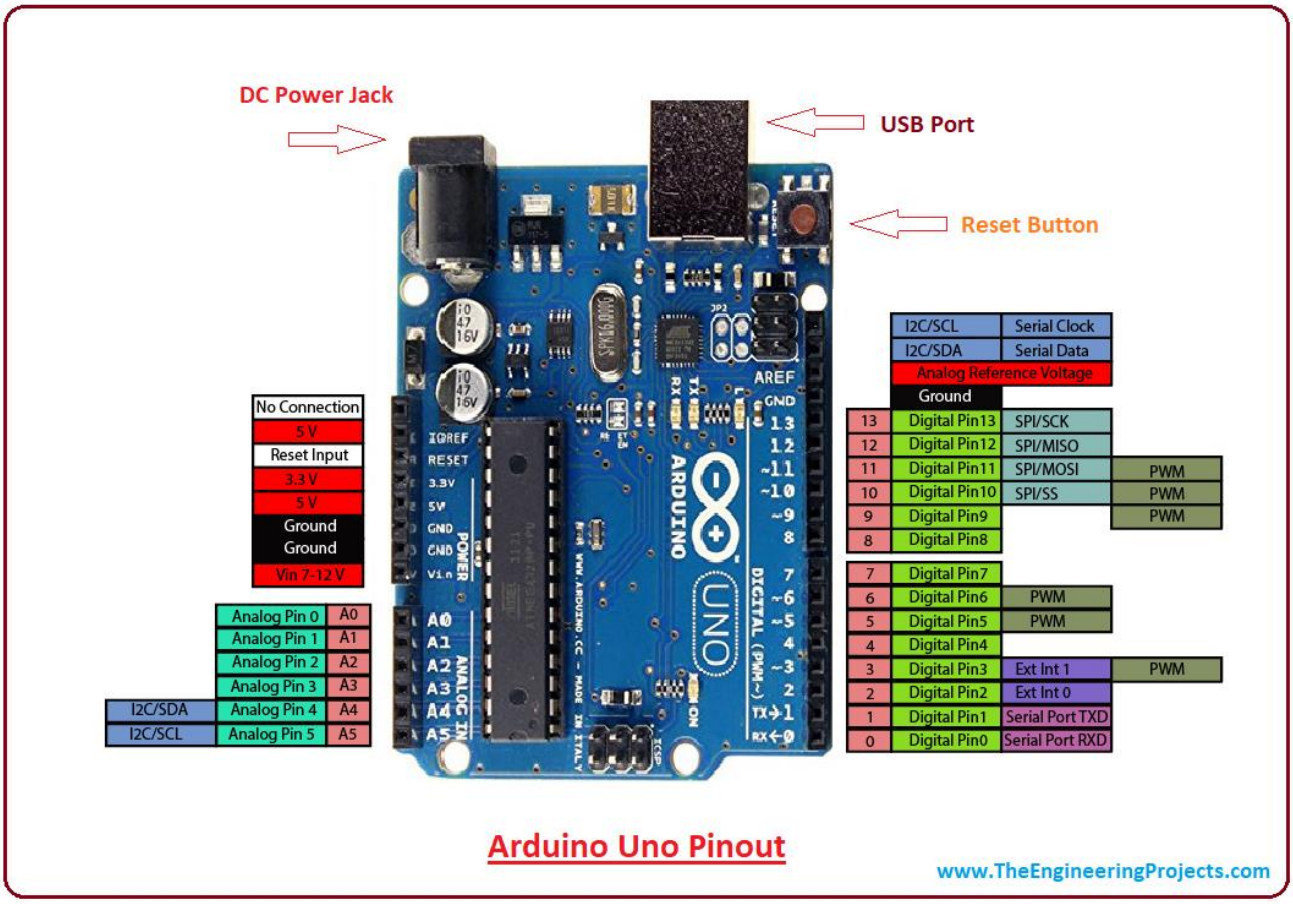
ממליץ לעבור על מסמך בתוך docs שמסביר על הframe-ים ואיך הם שמורים במערכת (frame_lifetime.md)

מפצל USB-HUB



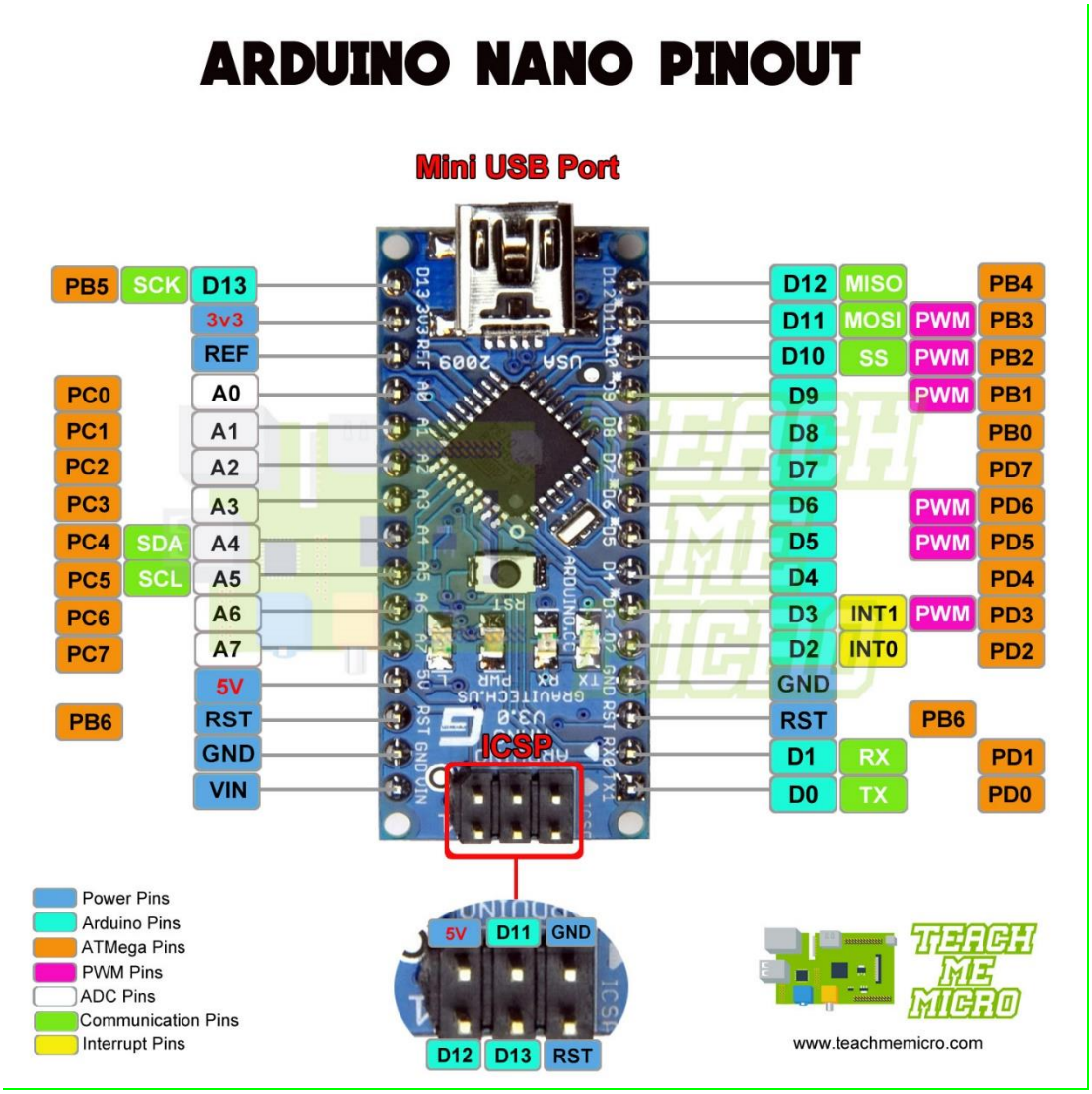
תיאור: המפצל מכיל כניסה אחת ו4 יציאות של usb 2.0/3.0.
הספק: המפצל יכול להיות מוזן ממתח של מחשב המחובר אליו או מספק כוח.
אם מוזן ממחשב אז צורך [V]5 ו- [A]0.9 (המצב הנוכחי).
ניתן גם לחברו ישירות לספק מתח ואז צורך [V]12 ו- [A]1.5.

Arduino uno



צריכת מתח וזרם(5V ו- 0.05A).
השימוש העיקרי בבקר זה הוא עבור שליטה בoptical flow, על הארדואינו צרוב הקוד BitcrazeArd , ויש ממשק חיצוני שיודע לתקשר עם הקוד הצרוב על הארדואינו – לבקש ולקבל מידע מה- optical flow

Arduino nano



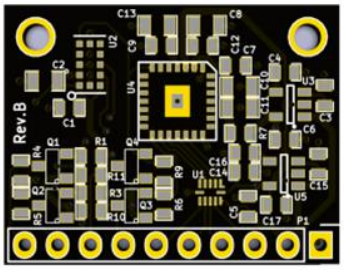
מתח וזרם(5V ו- 0.04A).

השימוש העיקרי בבקר זה הוא עבור שליטה במנוע ובהיגוי, על הארדואינו צרוב הקוד RaceCarArd , ויש ממשק חיצוני שיודע לתקשר עם הקוד הצרוב על הארדואינו , הקוד הצרוב מחכה לקבל פקודות נסיעה מהממשק החיצוני

סוללת מנועים

כמות: 1.
מתח: v7.4
אנרגיה: 5000mAh

חיישן תנועה



תיאור: החיישן הוא חיישן תנועה המכיל בתוכו שני רכיבים: רכיב המודד מרחק (VL5350x) למשטח תחתיו ורכיב המזהה שינוי בתנועה (optical pwm3901mb).

דגם: flow breakout bitcraze

הספק(משולב): 3.6[V] * (0.09 + 0.019)[A] = 0.1[W]

Fov: 42[deg].

Pixels_x: 30[px]

Pixels_y: 30[px]

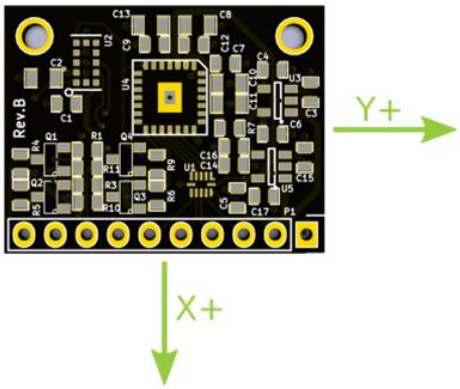
פלט: החיישן פולט שלושה סוגי מידע:

Dx – השינוי בפיקסלים בכיוון x. [px]

Dy – השינוי בפיקסלים בכיוון y. [px]

Range – המרחק מהרכיב למשטח תחתיו כאשר המינימום הוא 8[mm].

כאשר הצירים מוגדרים כך:



חישוב מהירות: ניתן לחשב מהירות מהפלט של החיישן על ידי הנוסחה הבאה:

$$V_x[\frac{mm}{s}] = \frac{range * \text{fov} * \frac{\pi}{180} \text{dx}}{pixels\_x * dt}$$

$$V_y[\frac{mm}{s}] = \frac{range * \text{fov} * \frac{\pi}{180} \text{dy}}{pixels\_\text{y} * dt}$$

הספק ואנרגיה

להלן טבלה המסכמת את ההפסקים של הרכיבים במערכת:

	מתח [V]	זרם [A]	הספק [W]
Jetson	20	1.5	30
מצלמה	5	0.7	3.5
מפצל USB	5	0.9	4.5
Arduino uno	5	0.05	0.25
Arduino nano	5	0.04	0.2
חיישן תנועה	3.6	0.028	0.1

סך האנרגיה שהמערכת צורכת היא: 38.55[W].

הסוללה מספקת 11.1[V] וזרם של 8000[mAh] ולכן סך האנרגיה: 88.8[Wh].

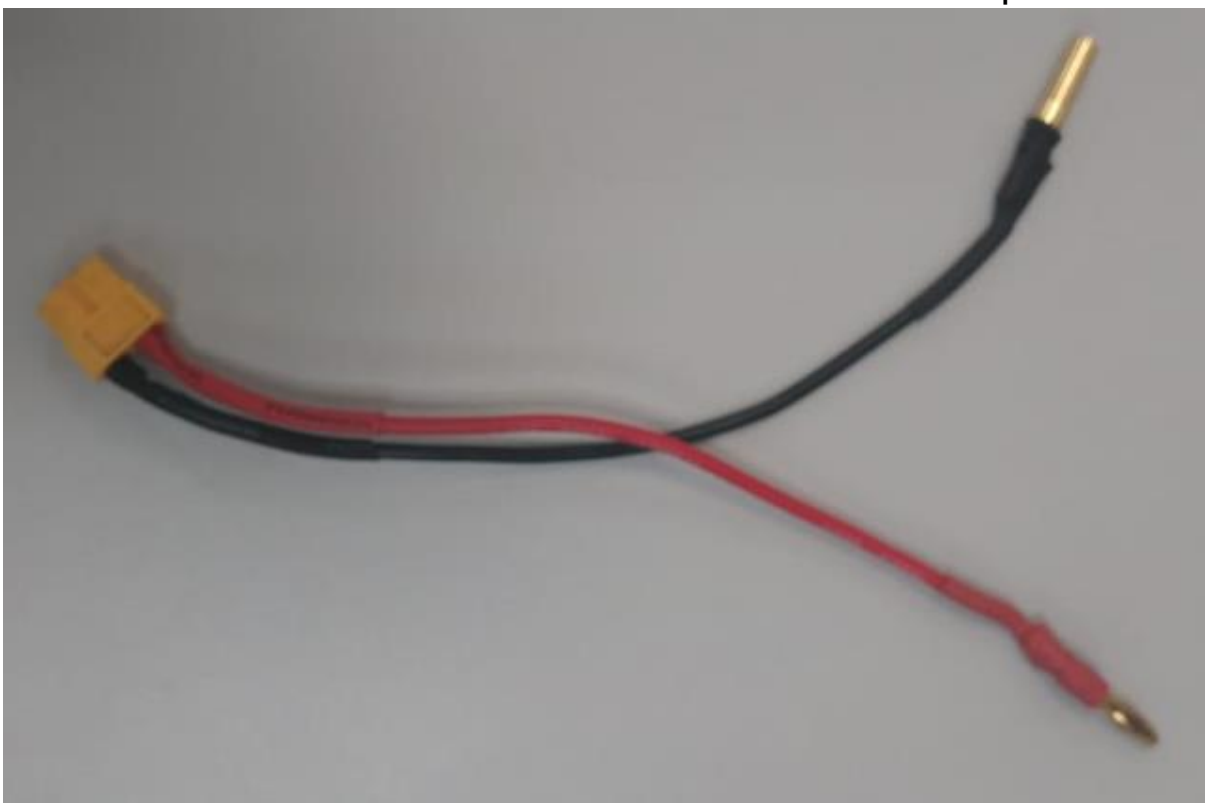
כלומר הסוללה מספיקה לזמן עבודה של $\frac{88.8}{38.55} = 2.3[hours]$.

כבלים והטענה

- הכבל שהוזכר בחלק של הxavier שמאפשר להחליף מתח קיר וסוללה ללא כיבוי הxavier :

- כבל מתאם בין סוללת המנועים למנוע:

- כבל מתאם בין הסוללת המנועים ליחידת הטעינה:



- יחידת הטעינה – מגיע עם ספק שמתחבר למתח קיר המזין את המטען, לכל אחת משתי הסוללות ישנו כבל "כיול" (עם ראש לבן) שאותו יש לחבר למטען כדי שהוא יזהה איזו סוללה מחוברת אליו ויתאים את הגדרות הטעינה בהתאם.

ספריות עזר והתקנות

על מנת שכל התוכן יעבוד כראוי יש להתקין את כל ספריות העזר והתוכנות הנדרשות, נחלק לשתי סביבות עיקריות: סביבת העבודה של ה – xavier המכילה את chaos, וסביבה המכילה את ה- RemoteControl.

ההתקנות עליהן נפרט הן:

JetPack: חבילה של nvidia המותאמת למוצרי ה- jetson השונים, בפרויקט הזה – xavier. מכילה מספר ספריות, בין היתר openCv, Cuda ועוד.

Cmake: ספרייה המאפשרת להגדיר קונפיגורציות כאשר מקמפלים קוד c/c++ בתצורה של cross-platform.

Qt: IDE לכתיבת c/c++ המאפשר פיתוח gui. מכיל המון class-ים וכמובן cross-platform.

OpenGl: ספרייה לתצוגה גרפית, ברוב המערכות הספרייה כבר מותקנת אבל ליתר ביטחון נרחיב גם על ספרייה זו.

Turbo-Jpeg: ספרייה המשמשת לדחיסת תמונות לפי האלגוריתם של jpeg, משתמש לדחיסת התמונות מהמצלמה טרם שליחתם ברשת.

LibRealSense: ספריית הממשק למצלמת ה – realsense.

JetPack

כאשר מתקינים את ה – xavier מההתחלה יש להתקין בנוסף את חבילה זו.
הסבר מפורט ו - וידאו המסביר את אופן ההתקנה אפשר למצוא בלינק:
https://www.google.com/search?q=how+to+install+jetpack+xavier&oq=how+to+install+jetpack+xavier&aqs=chrome..69i57j0.10401j0j4&sourceid=chrome&ie=UTF-8#kpvalbx=_xudzXZSkL8GXkwXC3LqgCg18
או המסמך הרשמי:
https://developer.download.nvidia.com/assets/embedded/secure/jetson/xavier/docs/jetson_agx_xavier_developer_kit_user_guide.pdf?qNc3nfq23E4iiPbU4El5YM-2YhTOMWndsW0fNgBwEJ0DQmh6NrHfYLPAcVH4yKts0BRO8Xltd2lFyQ_paeX7dl4xDqcr_ivi_h876Hw6ZUQsqRspsAaNwpEtWJpkBg9McgR4pwOgJYOUbvCGwG7xrTE-wBi3tGRdxl9LNaf7ffNdOWbzZeUWPMLdhEzChbwHm8Xrd6Qlcw
הכולל הסבר על היכולות של הרכיב.

דגשים:

- את קובץ ההתקנה מורידים למחשב חיצוני אחר(host) **לא על ה – xavier.** ההתקנה מתבצעת על מחשב ה – host עם כבל מחובר ל – xavier.
- יש לשים לב שבזמן ההתקנה עוברים למצב setup mode על ידי לחיצת שני כפתורים בו זמנית(מוסבר בווידאו).
- בשל חוסר זמן, לא חקרנו מספיק את יכולות החבילה, אנו ממליצים לשים דגש על לימוד החבילה לדורות הבאים שכן היא מותאמת לחומרה של ה – xavier.

Cmake

Cmake מאפשר לקמפל קוד c/c++ תחת הגדרות שונות (למשל הכרת נתיבי includes, קישור לקבצי dll/so/a/lib ועוד).
בכל פרויקט open source (ניתן למצוא בשפע ב github) יש קובץ הנקרא CmakeList.txt שמריצים אותו כאשר לאחר ההרצה נוצר קובץ makeFile ואז מקמפלים.
ניתן להתקין את Cmake גם בווינדוס וגם בלינוקס.
בלינוקס זה ייעשה על ידי השורות הבאות:

```
>> sudo apt-get install cmake
```

פרויקט יקומפל בעזרת השורות הבאות:

```
>> cmake .
>> make
```

הפקודה הראשונה(Cmake) מקבל כארגומנט path לתיקייה המכילה את הקובץ CmakeList.txt ומחפשת אותו לבד. במקום לשים path מלא ניתן לרשום נקודה,

הנקודה מציינת את התיקייה הנוכחית, כלומר ניתן לרשום את הפקודה עם כל ה path או לפתוח את הטרמינל בתיקייה הרצויה ולהשתמש בנקודה.

הפקודה השנייה(make) חייבת להיות מופעלת בתוך תיקייה שבה נמצא קובץ בשם makeFile או רק make והיא מריצה אותו לבד, זה למעשה הקימפול.

Qt

Qt(שנהגה כמו cute) הוא cross-platform IDE(כלומר את אותו פרויקט ניתן לקמפל בכל מערכות ההפעלה) ובנוסף ניתן לפתח איתו gui באמצעות c++.
ניתן לפתח בעזרתו תוכנות console רגילות והוא מכיל מאגר אדיר של class-ים מתקדמים(מערכות קבצים, sockets, serialization ועוד).

ניתן להוריד את התוכנה החינמית opensource מהאתר הרשמי של qt לווינדוס או ללינוקס אך ב – xavier צריך להוריד אותו מהטרמינל ולא מהאתר שכן באתר אין הורדה לארכיטקטורת arm.

כדי להוריד יש להריץ:

```
>> sudo apt-get install qt5-default qtcreator
```

OpenGl

OpenGl היא ספרייה גרפית opensource אשר יודעת להשתמש במנועים הגרפיים באמצעות ה – gpu.
נדרש להתקין אותה גם כי הספרייה לדחיסת התמונות דורשת אותה וכמו כן, הדוגמאות לשימושי המצלמה ב – github של realsense משתמש בה.
כדי להתקין את הספרייה יש להריץ:

```
>> sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

Turbo-Jpeg

יש להתקין את הספרייה זו אם נעשה שימוש בדחיסת תמונות באלגוריתם Jpeg.
לצורך פרויקט זה נכתבו שתי מחלקות בשימוש Turbo-Jpeg אשר יפורטו בהמשך.
אחת משמשת לדחיסת התמונה שמתקבלת מהמצלמה ושלחיתה ב – socket והשנייה משמשת לפריסת התמונה במחשב המקבל.

על מנת להתקין את הספרייה בלינוקס יש לבצע את ההוראות הבאות:

1. להוריד את תוכנת האסמבלי מלינק הבא:
   http://www.nasm.us/pub/nasm/releasebuilds/2.14.02/nasm-2.14.02.tar.xz

2. לחלץ את הקבצים לתיקייה הרצויה.
3. לפתוח את הטרמינל בתיקייה הרצויה.
4. להריץ:

```
>> ./configure --prefix=/usr && make
```

5. להריץ:

```
>>  sudo make install
```

6. להוריד את הספרייה של jpeg מהלינק הבא(יש להמתין 5 שניות לפני הורדה):
   https://downloads.sourceforge.net/libjpeg-turbo/libjpeg-turbo-2.0.2.tar.gz

7. לחלץ את הקבצים לתיקייה הרצויה.
8. להריץ:

```
>> mkdir build &&
cd    build &&
cmake -DCMAKE_INSTALL_PREFIX=/usr \
    -DCMAKE_BUILD_TYPE=RELEASE  \
    -DENABLE_STATIC=FALSE       \
    -DCMAKE_INSTALL_DOCDIR=/usr/share/doc/libjpeg-turbo-2.0.2 \
    -DCMAKE_INSTALL_DEFAULT_LIBDIR=lib  \
    .. && make
```

9. להריץ:

```
>> sudo make install
```

LibRealSense

יש להתקין את הספרייה לשימוש בממשק המצלמה, ההתקנה היא על LINUX . (כבר מותקן על הxavier אז אין צורך להתקין שוב אלה אם כן תעשו פורמט לרכיב או שתרצו להתקין את הספרייה על מחשב פיתוח אחר).

1). מורידים את הrepo מהgit מריצים את הפקודות:

```
$ cd $HOME
$ git clone https://github.com/jetsonhacks/buildLibrealsense2Xavier
$ cd buildLibrealsense2Xavier
```

2). נא לוודא שהמצלמה לא מחוברת בזמן ההתקנה ואז להריץ את סקריפט ההתקנה:

```
$ cd buildLibrealsense2Xavier
$ ./installLibrealsense.sh
```

3). הסקריפט מתקין את הספריות:

- The library is installed in /usr/local/lib
- The header files are in /usr/local/include
- The examples and tools are located in /usr/local/bin

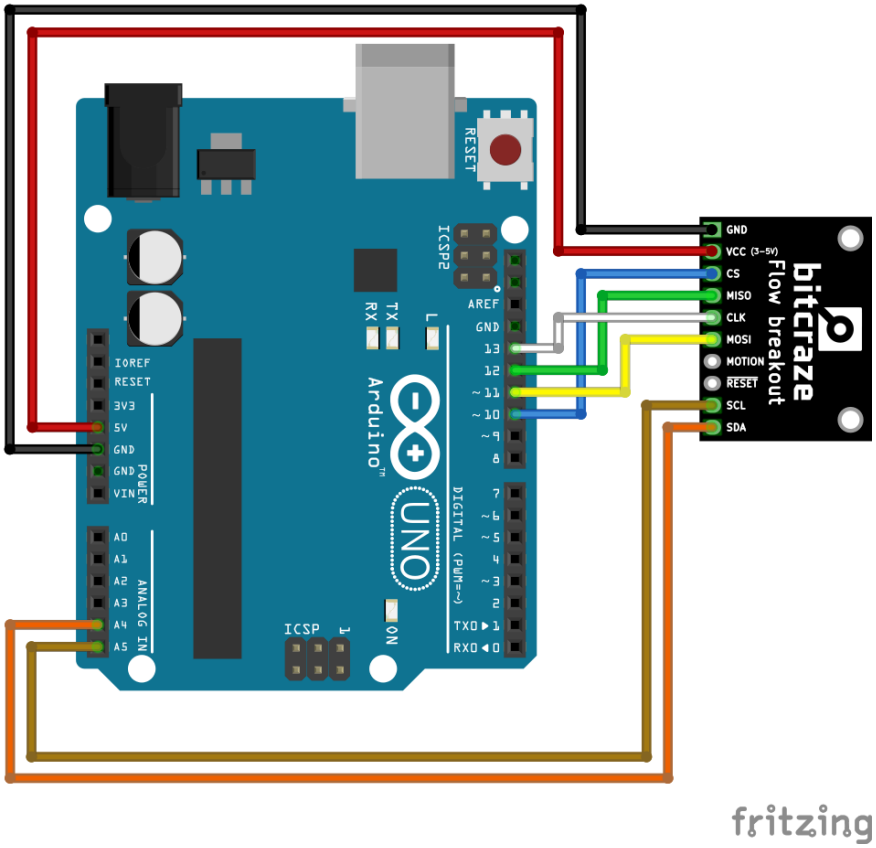
אפשור המצלמה במרחב המשתמש.

4). לאחר שהסקריפט מסיים להתקין יש לחבר את המצלמה וניתן להריץ את ה GUI הרשמי של המצלמה ע"י הפקודות:

```
$ cd /usr/local/bin
$ ./realsense-viewer
```

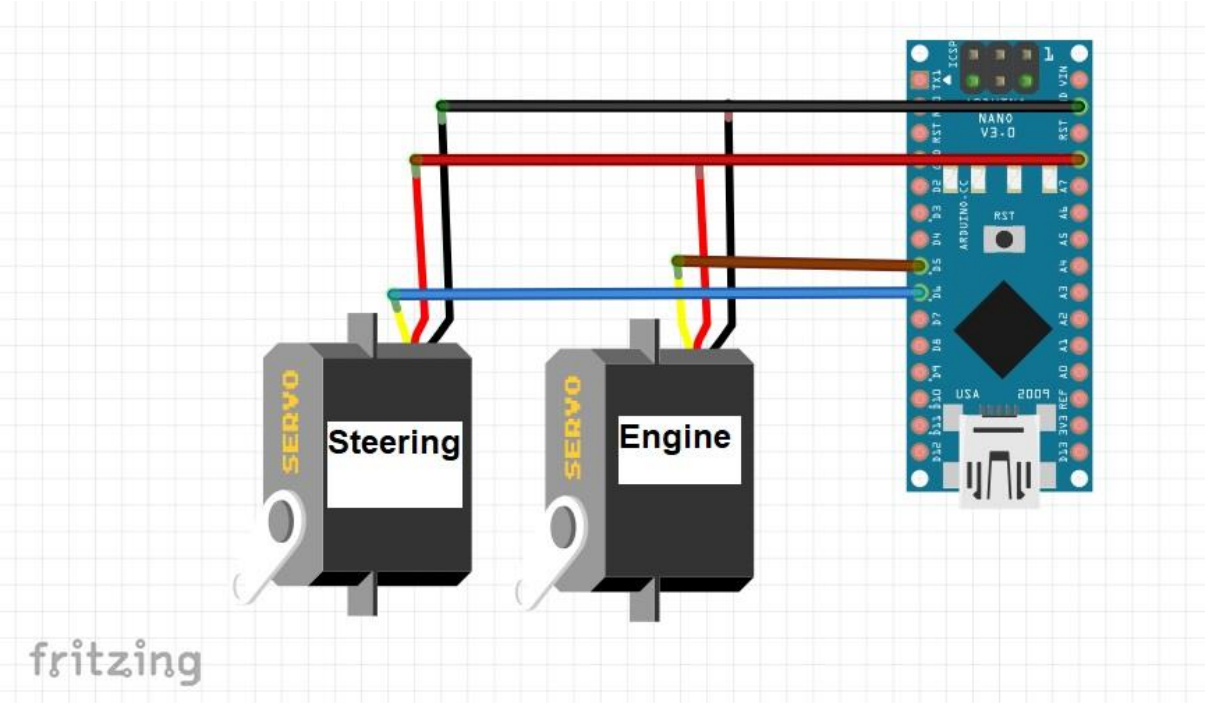
הספרייה כוללת ממשק משתמש למצלמה ופרויקטים לדוגמא לשימוש בממשק
https://github.com/IntelRealSense/librealsense

ארדואינו – חיישן תנועה



החיבור בין הארדואינו לבין הoptic flow תואם לציור זה (גם בצבעים), אל הbitcraze הולחמו פינים (הרכיב מגיע ללא חיבור GPIO נוח) , החוטים הולחמו ליחדות שלמות עבור חיבור וניתוק נוחים של הרכיב.

ארדואינו – מנועים – גלגלים



החיבור בין הארדואינו לבין הoptic flow תואם לציור זה (גם בצבעים).
בעת הדלקה ראשונית של הרכב, חייב ללחוץ לחיצה ארוכה על הכפתור שנמצא על הspeed controller עד שיהבהב בירוק.
על מנת שהמנוע יקבל פקודות נסיעה צריך לשלוח אל הבקר מהירות פקודת ARM, (נעשה על ידי הקוד הרשום בארדואינו)

ממשקים

יעד עיקרי של הפרויקט הוא כתיבת הממשקים מול הרכיבים והחיישנים השונים.
רוב הממשקים נכתבו בתצורה של pure virtual class כדי לבודד את הקבצים כמה שניתן וליצור הפרדה בין ממשק למימוש.
בתצורה זו, מאוד נח ליצור קבצי lib/dll/so ועשות includeרק לקובץ הממשק.

הסבר נוסף לכתיבת ממשקים ב- c++ אפשר למצוא בלינק הבא:
https://en.wikibooks.org/wiki/More_C%2B%2B_Idioms/Interface_Class

ממשק מצלמה:

RealSenseAPI

תיאור המחלקה:
ישנם 3 דרכים לתפעל את המצלמה RealSense.

1). דרך פונקציות ישירות של המכשיר, דוגמא לדרך תפעול זו אפשר לראות בדוגמא של המצלמה בשם api_how_to.h

2). דרך ממשק PIPE , דוגמא לתפעול זה ניתן למצוא גם כן בדוגמאות של המצלמה. (מהrepository של realsense).

3). דרך ממשק זה אשר מתבסס על הממשק דרך PIPE ואמור להיות פשוט יותר ואינטואיטיבי יותר.

תיאור כללי של הממשק וFLOW עבודה:

לאחר חיבור המצלמה פיזית למחשב , יש להתחבר למצלמה.

בתחילת עבודה יש לקנפג את המצלמה לFrameים הרצויים מכל סנסור פנימי של המצלמה (כולל הIMU). לאחר הקינפוג נותנים פקודה למצלמה לתחילת עבודה.
כאשר רוצים לקבל frame ברגע נתון קוראים לפונקציה captureFrame אשר היא שומרת את כל הframeים שקונפגו קודם לכן בתוך set. בעזרת הפונקציות הנתונות שולפים את הFrame הרצוי ועושים עליו את העיבוד הרצוי.

ה-structים שממשק המצלמה מחזיר נמצאים בקובץ Camera_types .

דוגמא בפסאדו קוד לתפעול המצלמה:

```
RealSense camera;
camera.connectCamera();
camera.setupColorImage(ress,fps);
camera.startCamera();
while(true){
        camera.captureFrames();
        DepthImage d_img = camera.getDepthImage();
       …. Process the frame……
}
```

- *שימו לב – אם לא מספיקים לטפל בFrame בפרק זמן של $\frac{1}{fps}$ התמונה תידרס ע"י הframe אחר.
- בממשק בחרנו להשתמש בEnum-ים של הקלאס כדי למנוע, כמה שניתן, הכנסת פרמטרים שגויים לפונקציות ולאפשר שימוש נוח בIDE .
- שימו לב – אם לא מחוברים למצלמה עם כבל שהוא USB3 יתכן וחלק מהרזולוציות שנתמכות ע"י הממשק לא יעבוד והמצלמה תקרוס. המצלמה יודעת לזהות לבד מה התעבורה שניתן להעביר בקו ועם כבל לא טוב מספיק המצלמה חוסמת חלק מהאפשרויות העבודה שלה.
- מומלץ לקורא את ה-docs של המצלמה מתוך הספרייה של realsense להבנת כל המגבלות של המצלמה.

https://github.com/IntelRealSense/librealsense/tree/master/doc

<u>פונקציות הממשק:</u>

bool connectCamera();

תיאור הפונקציה:

הממשק מניח שעובדים רק עם מצלמת realsense אחת, (למרות שאין בעיה לחבר כמה מצלמות לHOST עם שינויים קטנים בממשק)

בעזרת פונקציה זו הHOST מאתר את המצלמה המחוברת אליו ושולפת מהמצלמה את הסנסורים שלה (נכון לגבי realsense – 3 סנסורים) :

1. StrereoModule – בעזרתו מוצאים את תמונות העומק
2. RGBCamera – מצלמה רגילה
3. MotionModule – מוציא נתוני gyro וacceleration .

פרמטרים:

אין.

ערך החזרה:

הפונקציה מחזירה TRUE אם הצלחנו להתחבר למצלמה ולקרוא ממנה את שלושת הסנסורים שלה.

bool isConnect();

תיאור הפונקציה:

בפונקציה בודקת אם המצלמה מחוברת לHOST**.**

שימו לב* - אין לקרוא לפונקציה בתוך לולאה אין סופית כי היא פותחת thread נפרד במהלך הבדיקה. (יתכן ישנה דרך לבדוק אם המצלמה מחוברת בצורה שונה)

פרמטרים:

אין

ערך החזרה:

מחזיר TRUE אם המצלמה מחוברת.

void resetCamera();

תיאור הפונקציה:

מבצע אתחול חומרתי למצלמה. מומלץ לחכות קצת אחרי אתחול כזה ולא ישר לקרוא לפונקציות של המצלמה כדי לתת זמן לחומרה לעלות.

פרמטרים:

אין

ערך החזרה:

אין

void setupColorImage(RealSense::ColorFrameFormat format, RealSense::ColorReassolution resolution, RealSense::ColorCamFps fps);

תיאור הפונקציה:

קינפוג מצלמה לFrameים רצויים.

פרמטרים:

- פורמט רצוי (מרשימת הפורמטים בתוך הCLASS)

```
YUYV            , /**< 32-bit y0, u, y1, v data for every two pixels. Similar to
YUV422 but packed in a different order - https://en.wikipedia.org/wiki/YUV */
RGB8            , /**< 8-bit red, green and blue channels */
BGR8            , /**< 8-bit blue, green, and red channels -- suitable for OpenCV
*/
RGBA8           , /**< 8-bit red, green and blue channels + constant alpha channel
equal to FF */
BGRA8           , /**< 8-bit blue, green, and red channels + constant alpha channel
equal to FF */
Y16             , /**< 16-bit per-pixel grayscale image */
```

- רזולוציה רצויה
- FPS רצוי

*שימו לב שישנם שילובים של רזולוציה וFPS שלא יעבדו ביחד (המגבלות רשומות בקוד).

ערך החזרה:

אין

void setupInfraredImage(RealSense::InfrarFrameFormat format,
RealSense::InfrarRessolution resolution,
RealSense::InfrarCamFps fps,
RealSense::InfrarCamera side);

תיאור הפונקציה:

מקנפג את מצלמות האינפרה אדום (חלק מסנסורי ה StereoModule) .
ניתן להוציא מידע על כל אחת מ2 המצלמות.

פרמטרים:

- פורמט רצוי

```
Y8, /**< 8-bit per-pixel grayscale image */
Y16 /**< 16-bit per-pixel grayscale image */
```

- רזולוציה רצויה
- FPS רצוי
- בחירת אחת מ2 המצלמות

*שימו לב שישנם שילובים של רזולוציה וFPS שלא יעבדו ביחד (המגבלות רשומות בקוד).

ערך החזרה:

אין

void setupDepthImage(RealSense::DepthRessolution resolution,
RealSense::DepthCamFps fps);

תיאור הפונקציה:

מקנפג את תמונת העומק. הפורמט קבוע – Z16

פרמטרים:

- רזולוציה רצויה
- FPS רצוי

ערך החזרה:

אין

void setupGyro();

תיאור הפונקציה:

קינפוג הGYRO

פרמטרים:

אין

ערך החזרה:

אין

void setupAccel();

תיאור הפונקציה:

קינפוג הaccelometer

פרמטרים:

אין

ערך החזרה:

אין

void startCamera();

תיאור הפונקציה:

אחרי כל הקינפוגים קוראים לפונקציה כדי לגרום למצלמה לעבוד. מרגע זה לא ניתן לקנפג streamים חדשים.

פרמטרים:

אין

ערך החזרה:

אין

void captureFrame();

תיאור הפונקציה:

כאשר רוצים לקבל FRAME מהמצלמה קוראים לפונקציה זאת, עבור כל קינפוג שהגדרנו קודם לכן נקבל FRAME שונה אשר כולם יכנסו לSET פנימי.

פרמטרים:

אין

ערך החזרה:

אין

Camera::ColorImage getColorImage();

תיאור הפונקציה:

שליפת תמונת COLOR מהSET

פרמטרים:

אין

ערך החזרה:

FRAME שמכיל את התמונה ופרמטרים שקשורים לStruct

```
struct ColorImage
{
   uint64 frame_num;
   uint64 size;
   int32 bytes_per_pixel;
   int64 host_ts_ms;
   real64 camera_ts_ms;
   uint32 width;
   uint32 height;
   const unsigned char *data;
};
```

Camera::ColorImage getInfraredImage();

תיאור הפונקציה:

שליפת תמונת אינפרה אדום (אם קונפגה כזאת קודם לכן) מהSET

פרמטרים:

אין

ערך החזרה:

Struct שמכיל את התמונה ופרמטרים שקשורים לFRAME

Camera::DepthImage getDepthImage();

תיאור הפונקציה:

שליפת תמונת עומק מהSET

פרמטרים:

אין

ערך החזרה:

Struct שמכיל את התמונה ופרמטרים שקשורים לFRAME

```
struct DepthImage
{
   uint64 frame_num;
   uint64 size;
   int32 bytes_per_pixel;
   int64 host_ts_ms;
   real64 camera_ts_ms;
   uint32 width;
   uint32 height;
   real32 depth_scale;
   const unsigned char *data;
};
```

float getDepthUnits();

תיאור הפונקציה:

על מנת לקבל בתמונת העומק יחידות של מטרים עבור כל פיקסל יש לכפול את אותה בערך זה.

פרמטרים:

אין

ערך החזרה:

מספר בו יש לכפול את תמונת העומק לקבלת יחידות של מטרים.

Camera::Intrinsics getDepthCamIntrinsics();

תיאור הפונקציה: מחזירה את קבועי הintrinsic של מצלמת העומק

פרמטרים:

אין

ערך החזרה: המבנה הבא:

```
struct Intrinsics
{
   real32        ppx;      /**< Horizontal coordinate of the principal point of the
image, as a pixel offset from the left edge */
   real32        ppy;      /**< Vertical coordinate of the principal point of the
image, as a pixel offset from the top edge */
   real32        fx;       /**< Focal length of the image plane, as a multiple of pixel
width */
   real32        fy;       /**< Focal length of the image plane, as a multiple of pixel
height */
//    rs2_distortion model;    /**< Distortion model of the image */
   real32        coeffs[5]; /**< Distortion coefficients, order: k1, k2, p1, p2, k3 */
};
```

Camera::Intrinsics getColorCamIntrinsics();

תיאור הפונקציה : מחזירה את קבועי הintrinsic של מצלמת הצבע

פרמטרים:

אין

ערך החזרה:

מבנה ה Intrinsics .

Camera::Intrinsics getIfraRedCamIntrinsics();

תיאור הפונקציה: מחזירה את קבועי הintrinsic של מצלמת האינפרה אדום

פרמטרים:

אין

ערך החזרה:

מבנה ה Intrinsics .

Camera::MotionIntrinsics getMotionCamIntrinsics();

תיאור הפונקציה: מחזירה את ערכי ה Intrinsic עבור הIMU

פרמטרים:

אין

ערך החזרה: המבנה MotionIntrinsics :

```
struct MotionIntrinsics
{
   /* \internal
   * Scale X    ,   cross axis  , cross axis ,  Bias X \n
   * cross axis ,   Scale Y  ,   cross axis  Bias Y \n
   * cross axis  ,  cross axis , Scale Z  ,   Bias Z */
   float data[3][4];         /**< Interpret data array values */

   float noise_variances[3];  /**< Variance of noise for X, Y, and Z axis */
   float bias_variances[3];   /**< Variance of bias for X, Y, and Z axis */
};
```

Camera::Extrinsics getExtrinsics(RealSense::Stream from_stream, RealSense::Stream to_stream);

תיאור הפונקציה: מחזירה את מטריצת ההמרה בין 2 הstreamים הרצויים

פרמטרים:

מאיזה stream לאיזה stream אנחנו רוצים את הטרנספורמציה

ערך החזרה:

```
/** \brief Cross-stream extrinsics: encode the topology describing how the
different devices are connected. */
struct Extrinsics
{
   float rotation[9];    /**< Column-major 3x3 rotation matrix */
   float translation[3]; /**< Three-element translation vector, in meters */
};
```

Camera::EulerAngles getAngularVelocities();

תיאור הפונקציה: מחזירה את המהירויות הזוויתיות בכל ציר מהgyro

פרמטרים:

אין

ערך החזרה:

```
struct AngularVelocities
{
   real32 x_pitch;
   real32 y_yaw;
   real32 z_roll;
   int64 host_ts_ms;
   real64 camera_ts_ms;
};
```

Camera::AccelData getAccelData();

תיאור הפונקציה: מחזירה את הנתונים מהaccelerator

פרמטרים:

אין

ערך החזרה:

```
struct AccelData
{
   real32 x;
   real32 y;
   real32 z;
   int64 host_ts_ms;
   real64 camera_ts_ms;
};
```

ממשק מנוע:

Arduino

תיאור המחלקה:

Arduino היא מחלקה היורשת מהמחלקה האבסטרקטית MotorAPI – מחלקה המגדירה את פונקציות הממשק הדרושות (למקרה של שינוי עתידי מארדואינו) המחלקה מכילה בתוכה פונקציות ממשק עבור כל הרכיבים המחוברים לארדואינו , המנוע וההיגוי

פרמטרים עבור המחלקה:

speed: מהירות ניתנת במספר שלם חסר יחידות הנע בין [-500,500]
מספר זה מייצג את הערך הנשלח אל הPWM של הארדואינו , כאשר pwm=1500 הוא האפס מבחינת מהירות (ומשמש גם להדלקת המנוע) והמספר הממשי שנשלח הוא 1500+speed

Angle : הזווית נשלחת במעלות , הערכים האפשריים [0,1000], זווית של 500 מייצגת גלגלים ישרים

לשים לב בעת שימוש ראשוני יש צורך לתת הרשאות קריאה כתיבה עבור קובץ הסריאל , ניתן להשתמש בפקודה :

"chmod 777 /dev/ttyUSB0"

פונקציות הממשק:

bool Arduino::connect()

תיאור הפונקציה:

הפונקציה מתחברת לתקשורת סריאלית אל מול הארדואינו לממשק.

פרמטרים:

אין.

ערך החזרה:

מצביע לממשק.

int Arduino::getAngle()

תיאור הפונקציה:
מחזירה את הזווית הנוכחית (שמורה במשתנה פנימי במחלקה).

פרמטרים:
אין.

ערך החזרה:
הזווית הנוכחית.

int Arduino::getSpeed()

תיאור הפונקציה:
מחזירה את המהירות הנוכחית ביחידות PWM (שמורה במשתנה פנימי במחלקה).

פרמטרים:
אין.

ערך החזרה:
המהירות הנוכחית.

Arduino &Arduino::drive(const int &wanted_speed, const int &wanted_angle)

תיאור הפונקציה:
הפונקציה שולחת פקודת נסיעה למנוע והגלגלים וגורמת לו לנסוע במהירות והזווית הנתונות עד לפקודה חדשה

פרמטרים:
wanted_speed – המהירות הרצויה
wanted_angle – הזווית הרצויה

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino &Arduino::changeSpeed(const int &wanted_speed)

תיאור הפונקציה:
הפונקציה שולחת פקודת נסיעה למנוע וגורמת לו לנסוע במהירות הנתונה עד לפקודה חדשה

פרמטרים:
wanted_speed – המהירות הרצויה

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino& Arduino::changeSpeedBy(const int &delta)

תיאור הפונקציה:
הפונקציה שולחת פקודה למנוע לשנות את המהירות במספר הנתון

פרמטרים:
delta – בכמה יחידות לשנות את המנוע מהמצב הנוכחי

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino &Arduino::changeAngle(const int &wanted_angle)

תיאור הפונקציה:
הפונקציה שולחת פקודה להיגוי וגורמת לו לשנות את הגלגלים בזווית הנתונה עד לפקודה חדשה

פרמטרים:
wanted_angle – הזווית הרצויה

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino& Arduino::changeAngleBy(const int &delta)

תיאור הפונקציה:
הפונקציה שולחת פקודה להיגוי וגורמת לו להוסיף או להחסיר מעלות מזווית הגלגלים

פרמטרים:
delta – בכמה מעלות לשנות את הזווית מהמצב הנתון

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino &Arduino::stop()

תיאור הפונקציה:
הפונקציה שולחת פקודת עצירה, עוצרת את המנוע ומיישרת את הגלגלים

פרמטרים:
אין

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

Arduino &Arduino::driveCurrentState(){

תיאור הפונקציה:
הפונקציה שולחת פקודה להמשיך לנסוע במצב הנוכחי

פרמטרים:
אין

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

ממשק תקשורת סריאלית

תיאור:
ISerial הוא ממשק הנועד לממש תקשורת סריאלית בין מכשירים.
השימוש העיקרי בפרויקט שלנו הוא ליצור תקשורת בין המחשב (במקרה הנ"ל xavier) ל Arduino nano ולהעביר פקודות לבקר מנוע דרך תקשורת זו.
המחלקה נתמכת על ידי לינוקס בלבד וחלק מהשדות שלה הוא נתיב לקובץ סריאלי (ברגע שמחברים arduino למחשב נפתח קובץ file descriptor או בקיצור fd).
דרך קובץ זה ניתן לבצע פקודות קריאה וכתיבה כפי שיוסבר בהמשך.

בנוסף קיימת מחלקה שנקראת Serial אשר יורשת ממחלקה זו שבה נמצא כל המימוש, הפונקציות המפורטות בהמשך מוגדרות בממשק ISerial וממומשות במחלקה Serial.

פונקציות הממשק:

```
static std::shared_ptr<ISerial> create()
```

תיאור הפונקציה:
הפונקציה יוצרת מופע של הממשק ומחזירה מצביע לממשק.

פרמטרים:
אין.

ערך החזרה:
מצביע לממשק.

```
virtual ISerial &connect(const string &path = "/dev/ttyACM0")
```

תיאור הפונקציה:
הפונקציה פותחת ערוץ תקשורת סריאלי לקובץ שמועבר כפרמטר.

פרמטרים:
path – הקובץ שנפתח עבורו ערוץ סריאלי, הנתיב הדיפולטיבי הוא נתיב אוטומטי שנפתח כאשר מחברים את ה- arduino.

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

virtual void flush()

תיאור הפונקציה:
הפונקציה מרוקנת את הקובץ הסריאלי.

פרמטרים:
אין

ערך החזרה:
אין

virtual ISerial &write(const string &msg)

תיאור הפונקציה:
הפונקציה כותבת לערוץ התקשורת מחרוזת מטיפוס std::string.

פרמטרים:
msg – המחרוזת שנשלחת לערוץ התקשורת.

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

virtual ISerial &write(const char &msg)

תיאור הפונקציה:
הפונקציה כותבת לערוץ התקשורת תו יחיד.

פרמטרים:
msg – התו שנשלח לערוץ התקשורת.

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

virtual string read(const uint &len)

תיאור הפונקציה:
הפונקציה קוראת מחרוזת מטיפוס std::string מערוץ התקשורת.
פונקציה חוסמת.

פרמטרים:
len – אורך המחרוזת שיש לקרוא.

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

virtual void read(char *dst, const uint &len)
תיאור הפונקציה:
הפונקציה קוראת מידע מערוץ התקשורת.
פונקציה חוסמת.

פרמטרים:
dst – היעד שאליו יש לקרוא את המידע.
len – אורך המחרוזת שיש לקרוא.

ערך החזרה:
מחזיר את הממשק חזרה לצורך שרשור פונקציות.

virtual bool isConnected() const
תיאור הפונקציה:
הפונקציה בודקת האם יש חיבור סריאלי או לא.

פרמטרים:
אין

ערך החזרה:
מחזירה true אם יש חיבור, אחרת מחזירה false.

virtual ~ISerial()

תיאור הפונקציה:
הפונקציה משחררת את כל המשאבים של התקשורת הסריאלית.

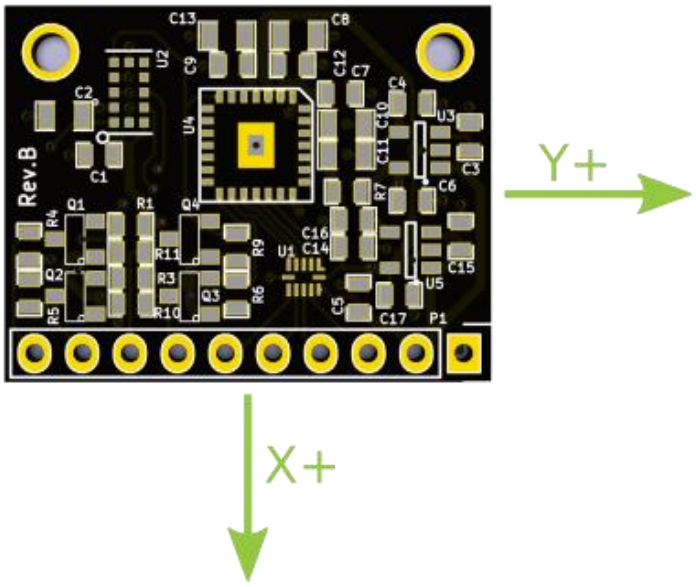
פרמטרים:
אין.

ערך החזרה:
אין.

ממשק חיישן תנועה:

Bitcraze

מערכת הצירים של הרכיב :



pixel resolution is 30x30
FOV is 42°

More data on : https://wiki.bitcraze.io/breakout:flow

אנו משתמשים ברכיב על מנת לקבל מבנה בשם Flow שמכיל 4 שדות
Dx -תזוזה ביחידות פיקסלים בציר x בין דגימות
Dy -תזוזה ביחידות פיקסלים בציר y בין דגימות
Dt - הזמן במילי שניות בין דגימות
Range – המרחק מהרצפה במילימטרים

<u>תיאור המחלקה:</u>
המחלקה עוטפת את התקשורת מול רכיב ה –optic sensor ומעבירה ממנו מידע אל המשתמש

לשים לב בעת שימוש ראשוני יש צורך לתת הרשאות קריאה כתיבה עבור קובץ הסריאל , ניתן להשתמש בפקודה :
"chmod 777 /dev/ttyACM0"

<u>פונקציות הממשק:</u>

void bitcraze::setup()

תיאור הפונקציה:
הפונקציה מתחברת אל רכיב ה optic sensor (מצורף ציור של החיבור מול הארדואינו)

פרמטרים:
ללא

ערך החזרה:
ללא

bool isConnected()

תיאור הפונקציה:
הפונקציה בודקת אם אנו מחוברים אל הארדואינו

פרמטרים:
ללא

ערך החזרה:
True במידה ואנו מחוברים.

Bitcraze &requestFlowData();

תיאור הפונקציה:
הפונקציה שולחת הוראה לארדואינו להתחיל לשלוח מידע מהחיישן

פרמטרים:
ללא

ערך החזרה:
מצביע לממשק.

Bitcraze &stopStream();

תיאור הפונקציה:
הפונקציה שולחת הוראה לארדואינו להפסיק לשלוח מידע מהחיישן

פרמטרים:
ללא

ערך החזרה:
מצביע לממשק.

Flow getFlowOutput() ;

תיאור הפונקציה:
הפונקציה קוראת את המידע שהחיישן שולח דרך הארדואינו
פרמטרים:
ללא

ערך החזרה:
Flow – מבנה המכיל את המידע מהחיישן

ממשק TcpClient

תיאור המחלקה:
ITcpClient הוא ממשק הנועד ליצור client והתחבר לשרת בתקשורת TCP.
בנוסף קיימת מחלקה שנקראת TcpClient אשר יורשת ממחלקה זו שבה נמצא כל המימוש, הפונקציות המפורטות בהמשך מוגדרות בממשק ITcpClient וממומשות במחלקה TcpClient.

פונקציות הממשק:

```
static std::shared_ptr<ITcpClient> create()
```

תיאור הפונקציה:
הפונקציה יוצרת client חדש אשר יכול להתחבר לשרת ומחזירה מצביע לממשק.

פרמטרים:
אין.

ערך החזרה:
אין.

```
virtual void connect(const string& ip, const unsigned short& port) const
```

תיאור הפונקציה:
הפונקציה מתחברת לשרת קיים באמצעות ה – ip וה – port של השרת.

פרמטרים:
Ip – כתובת ה- ip של השרת.
Port – מספר הפורט של השרת.

ערך החזרה:
אין.

virtual bool isConnected() const

תיאור הפונקציה:
הפונקציה מחזירה אם קיימת תקשורת עם השרת או לא.

פרמטרים:
אין.

ערך החזרה:
מחזירה true אם יש תקשורת, אחרת מחזירה false.

virtual void disconnect()

תיאור הפונקציה:
הפונקציה מנתקת את התקשורת מהשרת.

פרמטרים:
אין.

ערך החזרה:
אין.

virtual void receive(char *dst, const uint &len, const uint &timeout_sec = 3)

תיאור הפונקציה:
הפונקציה קוראת מידע מהשרת.

פרמטרים:
Dst – היעד אליו יקרא המידע.
Len – אורך המידע בבתים.
Timeout_sec – זמן ההמתנה המקסימלי עד שהתכנית תמשיך, אם הזמן שווה ל – 0 התכנית תמשיך מיד אם אין מידע לקרוא.

ערך החזרה:
וקטור של בתים המכיל את המידע שנקרא.

virtual void send(const std::vector<char>& data) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מידע לשרת.

פרמטרים:
Data – המידע שיש לשלוח.

ערך החזרה:
אין.

virtual void send(const string& message) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מחרוזת לשרת.

פרמטרים:
message – המחרוזת שיש לשלוח.

ערך החזרה:
אין.

virtual void send(const char* data, const uint &len) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מידע לשרת.

פרמטרים:
data – מצביע למידע שיש לשלוח.
Len – אורך המידע בבתים.

ערך החזרה:
אין.

virtual ~ITcpClient() noexcept

תיאור הפונקציה:
שחרור כל המשאבים.

פרמטרים:
אין.

ערך החזרה:
אין.

ממשק TcpServer

תיאור המחלקה:
ITcpServer הוא ממשק הנועד ליצור שרת ולפתוח תקשורת TCP בין מחשבים.
בנוסף קיימת מחלקה שנקראת TcpServer אשר יורשת ממחלקה זו שבה נמצא כל המימוש, הפונקציות המפורטות בהמשך מוגדרות בממשק ITcpServer וממומשות במחלקה TcpServer.

פונקציות הממשק:

```
static std::shared_ptr<ITcpServer> create()
```

תיאור הפונקציה:
הפונקציה יוצרת Server חדש ומחזירה מצביע לממשק.

פרמטרים:
אין.

ערך החזרה:
מצביע לממשק.

```
virtual void bind(const string &ip, const unsigned short &port,
                  const int &max_num_of_clients) noexcept
```

תיאור הפונקציה:
הפונקציה פותחת תקשורת TCP ומתחברת ל – Ip כשרת.

פרמטרים:
Ip – מחרוזת המכילה את כתובת ה – ip של מחשב ה – host שבו יפתח השרת.
Port – מספר פורט שרירותי שמוגדר בין המשתמשים שבניהם נפתחת התקשורת.
Max_num_of_client – מספר המשתמשים המקסימלי שיכולים להתחבר לשרת.

ערך החזרה:
אין.

virtual bool isBind() const noexcept

תיאור הפונקציה:
הפונקציה בודקת אם קיימת תקשורת בין השרת למחשב ה – host.

פרמטרים:
אין.

ערך החזרה:
מחזירה true אם נפתחה תקשורת, אחרת מחזירה false.

virtual bool hasConnectionWithSocket(const Socket &socket)

תיאור הפונקציה:
הפונקציה בודקת אם קיימת תקשורת בין השרת ל – client מסוים.

פרמטרים:
Socket – המזהה של ה - client שאיתו יש לבדוק את התקשורת.

ערך החזרה:
מחזירה true אם קיימת תקשורת, אחרת מחזירה false.

virtual Socket waitForConnections(const uint &timeout_sec)

תיאור הפונקציה:
הפונקציה מחכה להתחברות client חדש ומחזירה Socket(שם אחר ל int) אשר בעזרתו אפשר לגשת ל - client שהתחבר.

פרמטרים:
Timeout_sec – הזמן המקסימלי שהפונקציה תמתין לבקשת התחברות חדשה.

ערך החזרה:
Socket ל - client שהתחבר.

virtual unsigned long getNumOfConnectedClients() const

תיאור הפונקציה:
מחזירה את מספר ה – clients המחוברים לשרת.

פרמטרים:
אין.

ערך החזרה:
מספר ה – clients המחוברים לשרת.

virtual void receive(const Socket &socket, char *dst, const uint &len, const uint &timeout_sec = 3)

תיאור הפונקציה:
הפונקציה קוראת מידע מ - client מסוים שמחובר לשרת.

פרמטרים:
Socket – המזהה של ה - client שממנו יש לקרוא את המידע.
Len – אורך המידע בבתים.
Timeout_sec – זמן מקסימלי להמתנה למידע, אם הזמן שווה ל – 0, התכנית תמשיך מיד.

ערך החזרה:
אין.

virtual void send(const Socket& socket, const std::vector<char>& data) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מידע ל - client מסוים שמחובר לשרת.

פרמטרים:
Socket – המזהה של ה - client שאליו יש לשלוח את המידע.
Data – המידע שיש לשלוח.

ערך החזרה:
אין.

virtual void send(const Socket& socket, const string& message) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מחרוזת ל - client מסוים שמחובר לשרת.

פרמטרים:
Socket – המזהה של ה - client שאליו יש לשלוח את המידע.
message – המחרוזת שיש לשלוח.

ערך החזרה:
אין.

virtual void send(const Socket& socket, const char *data, const uint &len) const noexcept

תיאור הפונקציה:
הפונקציה שולחת מידע ל - client מסוים שמחובר לשרת.

פרמטרים:
Socket – המזהה של ה - client שאליו יש לשלוח את המידע.
data – מצביע למידע שיש לשלוח.
Len – אורך המידע בבתים.

ערך החזרה:
אין.

virtual ~ITcpServer() noexcept

תיאור הפונקציה:
שחרור כל המשאבים.

פרמטרים:
אין.

ערך החזרה:
אין.

מחלקות נוספות

מלבד הממשקים מול החיישנים נכתבו מחלקות נוספות אשר משלבות בין הממשקים השונים:

RemoteControl: gui אשר מופעל במחשב מרוחק ולו 2 מטרות:

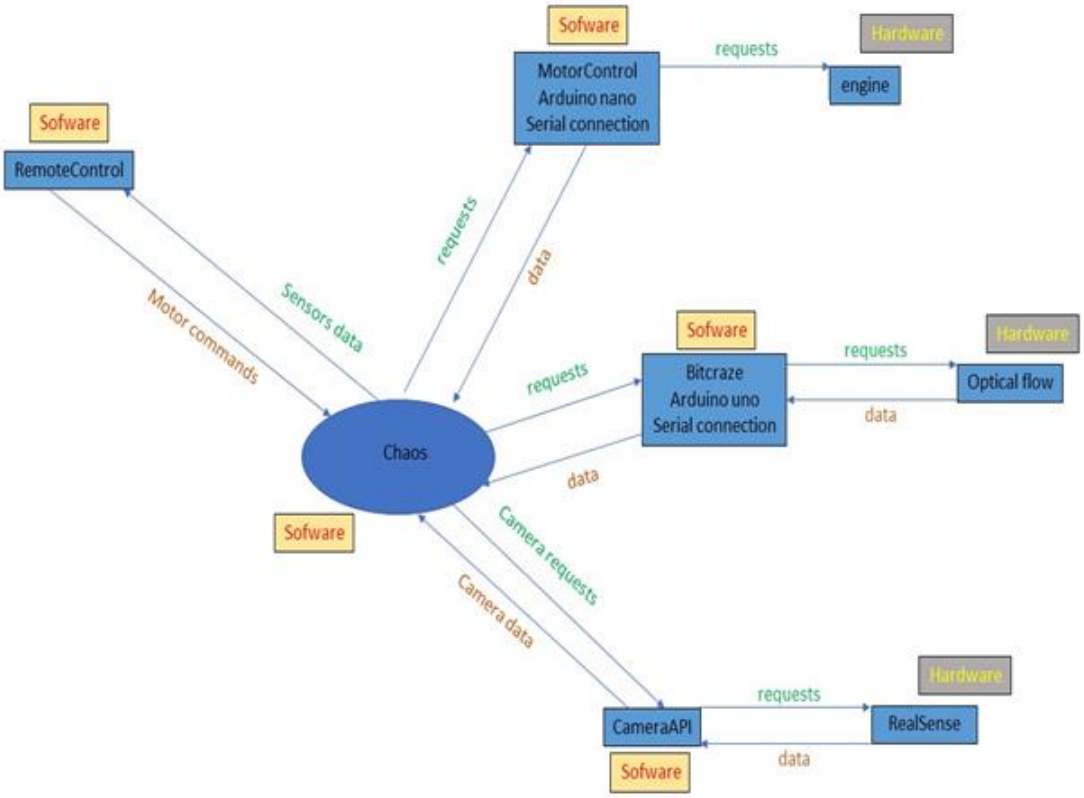
- הצגת המידע מהחיישנים.
- שליטה ברכב באמצעות החיצים במקלדת.

JpegCompressor: מחלקה המשתמש לדחיסת תמונות.

JpegDecompressor: מחלקה המשמשת לפריסת תמונות דחוסות.

Chaos: הפרויקט העוטף.

תיאור כללי של מערכת התוכנה:



RemoteControl – Gui

תיאור:
ה – RemoteControl הוא כלי שפותח באמצעות Qt והוא ממשק גרפי.
הפרויקט משמש כ – TcpServer עבור ה – Chaos לצורך קבלת תמונות.
הפרויקט משתמש כ – TcpClient עבור ה – Chaos לצורך שליחת פקודות למנועים.
הפרויקט מציג נתונים מהמצלמה: תמונת צבע, אינפרה אדום או מפת עומק ויודע לבקש מה – Chaos את סוג המידע בזמן ריצה.
הפרויקט מציג נתוני תנועה בזמן אמת כגון: נתוני gyro, נתוני accelometer ונתוני optical flow.

*כל המידע המוצג הוא מידע גולמי (raw data) ולא נעשה עליו שום עיבוד.

אופן פעולה:
הפרויקט מריץ מספר threads, אלו מנוהלים על ידי Qt לא באופן רגיל, לימוד על כך הוא נושא בפני עצמו ולכן נסביר את הרעיון הכללי.
כאשר הפרויקט מופעל נפתח ה – gui ורואים את כל התצוגה.
לאחר מכן יש להריץ את Chaos ולקבל חיווי ש – Chaos ו – RemoteControl מחוברים.(ה – RemoteControl מתייחס ל – Chaos כאל camera כי הוא ממתין ממנו לנתוני מצלמה).
כעת מוזרמים מה – Chaos ל – RemoteControl נתוני המצלמה וחיישן התנועה זמן אמת.
בנוסף בזמן ריצה ניתן להזין את ה – ip וה – port של המחשב שעליו מורץ ה – Chaos ולהתחבר אליו כ – client.
כאשר פעולה זו מצליחה מתקבל חיווי הנקרא Controller.
במצב זה ניתן להשתמש בחיצים כדי לשלוח פקודות ל – Chaos שישלח פקודות לארדואינו שישלח פקודות למנועים.

למעשה יש thread שמאזין ל – Chaos לקבל נתונים ו – thread נוסף שמאזין ללחיצת הכפתורים ושולח את הפקודות ל – Chaos.

כמו כן לפרויקט יש שדה מטיפוס JpegDecompressor הפורס את התמונה שהגיעה ומציג אותה.

JpegCompressor

תיאור המחלקה:
JpegCompressor היא מחלקה העוטפת את הספרייה של Turbo-jpeg לשימוש מצומצם של דחיסת תמונה על פי האלגוריתם של jpeg.

פונקציות הממשק:

JpegCompressor()

תיאור הפונקציה:
הפונקציה יוצרת מופע חדש של המחלקה.

פרמטרים:
אין.

ערך החזרה:
אין.

```
JpegCompressor(const uint32 &width, const uint32 &height,
               const JpegCompressor::Format &format,
               const uint32 &quality_percent)
```

תיאור הפונקציה:
הפונקציה יוצרת מופע חדש של המחלקה.

פרמטרים:
Width – רוחב התמונה בפיקסלים.
Height – גובה התמונה בפיקסלים.
Format – בחירה בין RGB ל – GREY_SCALE

ערך החזרה:
אין.

```
void setParams(const uint32 &width, const uint32 &height,
                const JpegCompressor::Format &format,
               const uint32 &quality_percent);
```

תיאור הפונקציה:
הפונקציה מעדכנת את הפרמטרים אם יש צורך בשינוי בזמן ריצה.

פרמטרים:
Width – רוחב התמונה בפיקסלים.
Height – גובה התמונה בפיקסלים.
Format – בחירה בין RGB ל – GREY_SCALE

ערך החזרה:
אין.

```
void compress(const uint8 *input)
```

תיאור הפונקציה:
הפונקציה דוחסת את התמונה ושומרת את הפלט.

פרמטרים:
Input – מצביע לתמונה, גודל תמונת ה – Input כבר נקבע ב – constructor או על ידי setParams, כמו כן הפלט נשמר בתוך המחלקה וניתן לקבלו על ידי getOutput.

ערך החזרה:
אין.

```
uint64 getCompressedSize()
```

תיאור הפונקציה:
הפונקציה מחזירה את גודל התמונה לאחר דחיסה.

פרמטרים:
אין.

ערך החזרה:
גודל התמונה בבתים של התמונה שנדחסה, אם לא נדחסה תמונה יוחזר 0.

uint8 *getOutput()

תיאור הפונקציה:
הפונקציה מחזירה את התמונה שנדחסה.

פרמטרים:
אין.

ערך החזרה:
התמונה הדחוסה, אין צורך לדאוג לניהול זיכרון ואת הגודל ניתן לקבל באמצעות getCompressedSize.

~JpegCompressor()

תיאור הפונקציה:
משחרר את כל המשאבים.

פרמטרים:
אין.

ערך החזרה:
אין.

JpegDecompressor

תיאור המחלקה:
JpegDecompressor היא מחלקה העוטפת את הספרייה של Turbo-jpeg לשימוש מצומצם של פריסת תמונה על פי האלגוריתם של jpeg.

פונקציות הממשק:

JpegDecompressor()

תיאור הפונקציה:
הפונקציה יוצרת מופע חדש של המחלקה.

פרמטרים:
אין.

ערך החזרה:
אין.

JpegDecompressor(const uint32 &width, const uint32 &height,
const JpegDecompressor::Format &format)

תיאור הפונקציה:
הפונקציה יוצרת מופע חדש של המחלקה.

פרמטרים:
Width – רוחב התמונה בפיקסלים.
Height – גובה התמונה בפיקסלים.
Format – בחירה בין RGB ל – GREY_SCALE

ערך החזרה:
אין.

JpegDecompressor &setParams(const uint32 &width, const uint32 &height, const JpegDecompressor::Format &format);

תיאור הפונקציה:
הפונקציה מעדכנת את הפרמטרים אם יש צורך בשינוי בזמן ריצה.

פרמטרים:
Width – רוחב התמונה בפיקסלים.
Height – גובה התמונה בפיקסלים.
Format – בחירה בין RGB ל – GREY_SCALE

ערך החזרה:
אין.

void decompress(uint8 *input, const uint64 &compressed_size)
תיאור הפונקציה:
הפונקציה פורסת את התמונה ושומרת את הפלט.

פרמטרים:
Input – מצביע לתמונה הדחוסה.
Compressed_size – גודל התמונה הדחוסה בבתים, הפלט נשמר במחלקה וניתן לקבלו על ידי שימוש ב – getOutput.

ערך החזרה:
אין.

int32 getBytesPerPixel()

תיאור הפונקציה:
הפונקציה מחזירה כמה בתים מוקצה לכל פיקסל.

פרמטרים:
אין.

ערך החזרה:
מספר הבתים המוקצה לכל פיקסל(3 בתים עבור RGB ו – 1 בתים עבור GREY_SCALE).

uint8 *getOutput()

תיאור הפונקציה:
הפונקציה מחזירה את התמונה שנפרסה.

פרמטרים:
אין.

ערך החזרה:
התמונה הפרוסה בגודל המקורי שלה, אין צורך לדאוג לניהול זיכרון.

~JpegDecompressor()

תיאור הפונקציה:
משחרר את כל המשאבים.

פרמטרים:
אין.

ערך החזרה:
אין.

Chaos

Chaos זהו שם הפרויקט שמשמש ביחד עם RemoteControl ככלי debug למערכת וכקוד דוגמא לשליפת נתונים מהסנסורים השונים.
הפרויקט Chaos משתמש במחלקה Racecar שבא ממומשות פונקציות המטפלות בדברים הבאים:

- באתחול כל הסנסורים
- שליפת המידע מהסנסורים והעברת פקודות מנוע למכונית
- הכנת חבילה לשליחה שכוללת נתונים מהמצלמה ומהחיישן מהירות
- שליחת המידע למחשב המרוחק (דרך sockets)

במחשב המרוחק תרוץ האפליקציה RemoteControl והיא תפענח את החבילה שנשלחה ותציג את הנתונים למשתמש (כמו כן פקודות הנסיעה מגיעות מאפליקציה זו).
אופן פעולה וזרימת התוכנית: (להלן הסבר על שלבי הריצה ב main של chaos לכן מומלץ לעבור על הקוד ביחד עם הסבר זה)

- אחרי שהכרזנו על טיפוס RaceCar יש להתחבר (דרך connect) לכל הסנסורים וליצור את כל אמצעי התקשורת שקיימים בין הסנסורים למחשב המרוחק שזה כולל תקשורת Serial ל2 הארדואינו (Serial נפרד לכל ארדואינו) וsocketים לעמדה המרוחקת.
  לכן בהתחברות יש להזין את הIP והPORT של המחשב המרוחק שאליו רוצים להתחבר ולשלוח לו מידע בהמשך. במקרה זה הHOST (שזה המחשב שמריץ את הפרויקט Chaos) משמש כclient אל מול הRemoteControl לשליחת מידע מהחיישנים.
  בנוסף יש להזין את הIP של המחשב עליו רץ הHOST וזה כדי לאפשר למחשבים אחרים להתחבר למחשב זה (bind). במקרה זה ה HOST משמש כserver מול הRemoteControl שמצפה לקבל פקודות למנועים.
- לאחר שהוגדרו כל דרכי התקשורת הדרושות, מריצים את התוכנית (דרך run). כעת, עבור כל סנסור יפתח thread נפרד שיטפל במידע מהסנסור. במקרה שלנו אנחנו פותחים שלושה:
  1 - שולף מידע מהמצלמה, מכין חבילה לשליחה לRemoteControl **ושולח** אותה דרך הsocket המתאים.
  2 – מחכה שמשתמש מרוחק יתחבר לHOST. ברגע **קבלת** (מהsocket) פקודות הנסיעה, שיגיעו מה RemoteControl , **יעביר** אותם דרך ה serial לארדואינו שאחראי על המנועים.
  3 – **מקבל** מידע דרך הserial מהארדואינו שאחראי על ה- bitcraze (חיישן המהירות). שומר את הנתונים בצד (משתמשים במנעול בזמן שמירת הנתונים כי הthread הראשון שמכין את החבילה לשליחה קורא את הנתונים האלה כדי להכניסם לחבילה).

❖ הסיבה שאנחנו משתמשים ב-2 ארדואינו נפרדים, אחד למנועים ואחד לחיישן מהירות, היא של-serial קשה להתמודד על כתיבה וקריאה בקצב שונה מאותו הfd. במיוחד שאחרי זמן מאוד קצר יחסית הfile מתפוצץ ובצורה ציקלית נכתב מחדש.

- בChaos נעשה שימוש בכל המחלקות שהוסברו קודם וזו דוגמא מאוד טובה לאופן התפעול שלהם.
- אחרי שב main שיגרנו את התוכנית (3 thread'ים כמעט בלתי תלויים רצים) הוספנו אופציה לעצור את כל התוכנית אשר עוצרת גם את כל הthread'ים הרצים (ע"י לחיצה על q במחשב הHOST). כמו כן אם לא הצלחנו להתחבר למחשב המרוחק או אם אבדה התקשורת עם המחשב המרוחק, כל התוכנית לא תמשיך לרוץ וכל הthread'ים יסיימו את פעילותם.
- במהלך הChaos ישנם הדפסות חיווי למשתמש שמראות באיזו שלב התוכנית ואם ישנם חיישן או תקשורת שלא מתחברים כראוי.
- חשוב לציין שעסקנו בעיקר בפיתוח ולא בהסעת הרכב לכן הפעלנו את התוכנית ידנית. כלומר כאשר הxavier מחובר למסך ואחרי הפעלת הchaos ניתקנו מהמסך ושמנו את האוטו על הרצפה. אם מעוניינים אפשר לעשות שהתוכנית תופעל אוטומטית בעליית מערכת.

פערים והמלצות

- יש לנהל את הפרויקט מהתחלת דרכו ועד סופו בגיט! גם לצרכי ניהול גרסאות, גם לצרכי תמיכה וגם לצרכי גיבוי. אין מה להתחיל לכתוב קוד אם לא יודעים להשתמש בפונקציונליות הבסיסית של הגיט.
- המצלמה וה – xavier מופעלות על ידי חברות מתחרות ולכן התאימות בניהן נמוכה. חלק מהתקלות הצלחנו לפתור אך אם אפשרי, אנו ממליצים להחליף את המצלמה.

  אחת הבעיות היא שלפעמים היא נתקעת בשידור מפת העומק, דבר הקורה גם ב – gui הרשמי שלהם.

  תקלה דומה קרתה גם בשידור תמונת הצבע אך זו נפתרה על ידי הגדלת ה – buffer של ה – usb ובכלל, על מנת שהספרייה תתמוך נאלצנו לעדכן kernel ב – xavier.
- ה – speed controller המובנה ברכב פחות נח, יש להדליק אותו בכל הפעלה מחדש והתצורה שבא מוצאים את ה – trigger שלו דרך הארדואינו בעייתית. ישנם בקרים אוטומטיים ודיגטליים שמציעים ממשקים נוחים יותר.
- כרגע יש סוללת מערכת אחת וסוללת מנועים אחת, בסביבת הפיתוח משתמשים לרוב בחשמל אבל כאשר תסיעו את הרכב הרבה מומלץ שיהיה עוד סוללה אחת מכל סוג – כאשר אחת בפעולה, השנייה בטעינה.
- אמנם הסענו את הרכב אך לא הרצנו אלגוריתמיקה ולא דחפנו את ה – xavier לקצה היכולת, במצב זה ההספק המקסימלי שלו הוא 30[W], אל אף שבחישוב ההספק התחשבנו בהספק המקסימלי מומלץ לעבור לסוללה של 20[V] על מנת לעלות את זמן הפעולה.
- המפצל HUB-USB מוזן כרגע על ידי ה – xavier. לאופי הנוכחי של בניית המערכת זה מספיק בהחלט. למצב שבו יתחברו חיישנים נוספים למפצל יש להזינו באמצעות ספק מתח.
- אם אכן יתחברו חיישנים נוספים למפצל, אז כדאי להגדיל את כמות היציאות באמצעות מפצל אחר כיוון שמומלץ להשאיר תמיד 2 יציאות פנויות מחיישנים (למקלדת ולעכבר).
- רוב הפרויקט עסק בכתיבת ממשקים ובהכנת פלטפורמת פיתוח לרכב אוטונומי ועל כן, התייחסנו אל ה – xavier כאל עוד מחשב ולא חקרנו מספיק או ניצלנו את יכולות החומרה שלו.