אוניברסיטת בן גוריון

בית ספר להנדסת חשמל ומחשבים

דו"ח תיעוד פרויקט מסכם קורס "מבנה מחשבים ספרתיים" 361-1-4191

מערכת לגילוי מקורות אור וניטור אובייקטים במרחב

רון בניטה 314882317

תמיר יעקב 207351974

תאריך הגשה:20.10.2025

מטרת הפרויקט

הפרויקט הינו פרויקט הסיכום של הקורס "מבנה מחשבים ספרתיים" אשר מאחד על גבי פרויקט יחיד מגוון משימות שמתבצעות על ידי שימוש בידע נרחב שנצבר במהלך המעבדות בקורס ובמהלך העבודה על הפרויקט.

מטרת הפרויקט היא לתכנון ולממש מערכת לגילוי מקורות אור ואובייקטים במרחב.המערכת מבוססת על בקר MSP430FG4619 וכוללת חיישני מרחק אולטראסוני וחיישני אור(LDR) בנוסף, משולב במערכת מנוע סרבו המאפשר סריקת המרחב.המערכת נשלטת ומציגה את הנתונים באמצעות GUI מצד המחשב, שמתקשר עם הבקר דרך ערוץ התקשורת UART.

תיאור הפרויקט

בפרויקט זה מומשה מערכת משובצת לזיהוי מקורות אור ולניטור אובייקטים במרחב. הסריקה מתבצעת באמצעות סיבוב חיישנים על גבי מנוע סרוו, וקריאת מידע משני סוגי חיישנים משלימים: אולטרסוני (מרחק) ואופטי (עוצמת הארה).

רכיבי החישה וההנעה

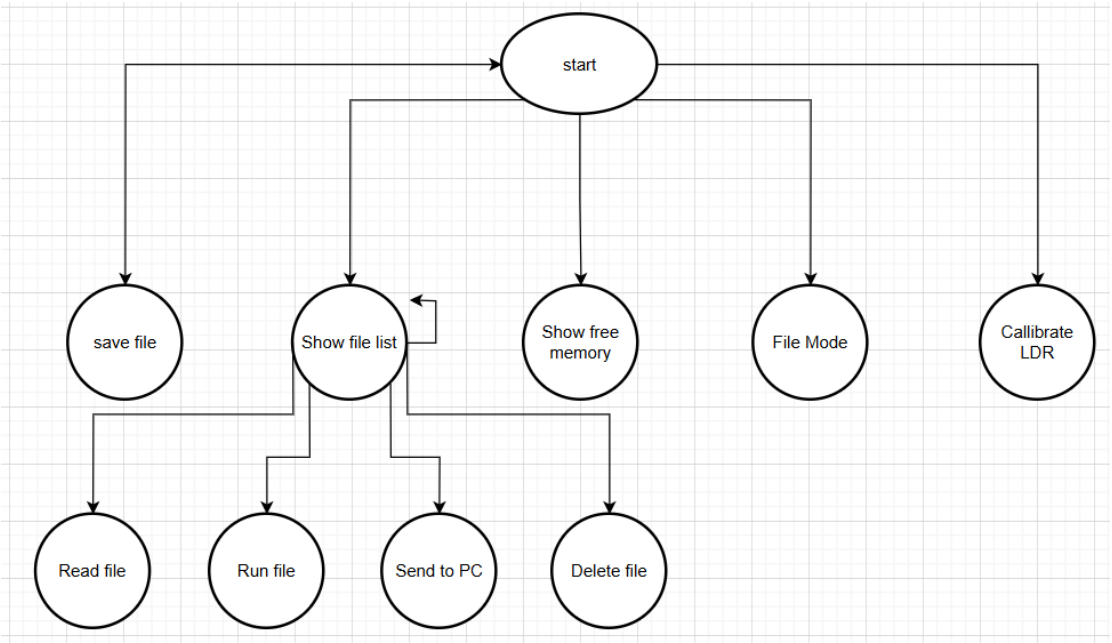
1. חיישן אולטרסוני (40 kHz)-
החיישן משדר מנה קצרה בת 8 מחזורים בתדר $40\ kHz$, ומודד את זמן החזרה (Echo) מהאובייקט. המרחק מחושב מזמן המעבר הלוך־ושוב:

$$d = \frac{v_{sound} \cdot t_{echo}}{2}$$

כאשר v_{sound} מהירות הקול (מותאמת תנאי סביבה). כך מתקבלת הערכת מרחק מדויקת לאובייקט.

2. זוג חיישני LDR (זיהוי כיוון מקור אור)-
נגדי LDR משנים את התנגדותם עם עוצמת ההארה. במימוש הפרויקט הם משולבים כמחלקי מתח, כך שמתקבל מתח מדידה פרופורציונלי לעוצמת האור על כל חיישן. השוואת המתחים בין שני ה־LDR מאפשרת לאתר את הזווית שבה מקור האור מצוי בניצב להרכבה.

3. מנוע סרוו (הכוונת זווית 0° – 180°)
הסרוו משמש לכוונת החיישנים לזוויות שונות לצורך סריקה. השליטה מתבצעת באמצעות אות PWM בתדר $\sim 50Hz$, עם רוחב פולס טיפוסי של $\sim 1.0 - 2.0\ ms$ (כ־5%–10% Duty), הממפה את התחום 0° – 180°. כך ניתן לבצע סריקה סדורה של המרחב, ולחבר את תוצאות המרחק (אולטרסוני) עם כיוון מקור האור (LDR).



תיאור פרוטוקול התקשורת :

לצורך תקשורת בין תחנת הבסיס (המחשב) לבקר (MCU), תוכנן ויושם **פרוטוקול תקשורת בינארי פשוט, אמין וגמיש**. מטרת הפרוטוקול היא לאפשר שליחה של פקודות ונתונים באופן עקבי, כולל אפשרות לבקרה על תהליך הסריקה, הפעלת סקריפטים, שליחת קבצים, כיול ועוד.

מבנה הודעה

כל הודעה בפרוטוקול בנויה במבנה הבא:

'@' | command_id | length | payload[] | crc | '#'

שדה	גודל	תיאור
'@'	1 byte	תו פתיחה של הודעה (Start Byte)
command_id	1 byte (uint8_t)	מזהה פקודה(Command ID)
length	1 byte (uint8_t)	מספר בתים במטען(payload)
payload[]	משתנה	תוכן ההודעה בהתאם לפקודה
crc	1 byte (uint8_t)	סכום בדיקת תקינות פשוט(CRC)
'#'	1 byte	תו סיום של הודעה(End Byte)

כיווני תקשורת

- **מהמחשב → לבקר (פקודות):**

תיאור	מזהה	פקודה
מגדיל את ערך התצוגה ב־MCU	0x01	RADAR_CMD_INC_LCD
מקטין את ערך התצוגה	0x02	RADAR_CMD_DEC_LCD
סיבוב ימינה של התצוגה (RRA = Rotate Right)	0x03	RADAR_CMD_RRA_LCD
סיבוב שמאלה של התצוגה (LRA = Rotate Left)	0x04	RADAR_CMD_LRA_LCD
קובע השהייה (delay) לסריקה, בפרמטר של מילי־שניות /10	0x05	RADAR_CMD_SET_DELAY
מאפס את התצוגה	0x06	RADAR_CMD_CLEAR_LCD
מסובב את הסרבו לזווית נתונה	0x07	RADAR_CMD_SERVO_DEG
מפעיל סריקה בטווח זוויות(start, end)	0x08	RADAR_CMD_SERVO_SCAN
מעביר את הבקר למצב שינה	0x09	RADAR_CMD_SLEEP
שולח פקודה כסקריפט (פקודה אחת בשורה)	0x0A	RADAR_CMD_SCRIPT_CMD
התחלת שליחה של סקריפט מרובה שורות	0x0B	RADAR_CMD_SCRIPT_START
מסמן סיום סקריפט	0x0C	RADAR_CMD_SCRIPT_STOP
שורת סקריפט בודדת בתוך רצף	0x0D	RADAR_CMD_SCRIPT_LINE
מודיע שסקריפט הסתיים ונשמר	0x0E	RADAR_CMD_SCRIPT_FILE_DONE
שליחת קובץ בחלקים)שם הקובץ(data +	0x0F	RADAR_CMD_FILE_PART
בקשת סטטוס מהבקר	0x10	RADAR_CMD_GET_STATUS
שליחת ערכי הכיול מה־PC) עבור LDR1 ו־LDR2)	0x11	RADAR_CMD_CALIBRATION

סריקה מלאה על כל הזוויות ומדידת LDR +מרחק	0x12	RADAR_CMD_FULL_SCAN

- **מהבקר → למחשב (תגובות):**

תיאור	מזהה	פקודה
החזרת ערכי הכיול מהבקר (לשימוש /אימות)	0x80	RADAR_MSG_LDR_CALIB
נתוני סריקה מלאים שנשלחו לאחר FULL_SCAN	0x81	RADAR_MSG_SCAN_DATA
אישור שהקובץ התקבל	0x82	RADAR_MSG_FILE_RECEIVED
הודעת לוג (טקסט חופשי)	0x83	RADAR_MSG_LOG
אישור שסקריפט הסתיים ורץ	0x84	RADAR_MSG_SCRIPT_DONE
נתוני מצב עכשוויים של המערכת	0x85	RADAR_MSG_STATUS

מנגנוני בטיחות

- **CRC**פשוט (סכום בתים) משמש לזיהוי שגיאות בקבלת ההודעה.
- הודעות שגויות CRC) לא תואם / מבנה לא חוקי (נדחות מיידית.
- מערכת ACK/NACK מאפשרת ווידוא קבלת פקודות קריטיות.

תמיכה בקבצים וסקריפטים

- הודעות מיוחדות מאפשרות **שליחת קבצים בפורמט בינארי** ,כאשר כל חלק נשלח בהודעה נפרדת מסוג .RADAR_CMD_FILE_DATA
- סקריפטים להפעלה אוטומטית נשלחים בשורת פקודות (טקסט פשוט) דרך .RADAR_CMD_SCRIPT

יתרונות הפרוטוקול

- **פשטות** :מתאים למשאבים מוגבלים של בקר.
- **גמישות** :ניתן להרחיב בקלות ע"י הוספת מזהי פקודות חדשים.
- **אמינות** :כולל בדיקת CRC והפרדה ברורה בין הודעות.
- **תמיכה דו-כיוונית מלאה** :כל פעולה מקבלת תגובה / ACK) נתונים.(

פענוח הודעות -
בנוסף לסט הפקודות , בנינו פונקציית עזר גם לצד מחשב וגם לMCU שהיא למעשה בונה הודעה , בכל פעם שאנחנו מקבלים תו אנחנו שולחים אותה לפירסור ושם נבנת ההודעה לפי מכונת מצבים :

מצבי המכונה

- STATE_MAGIC (איפוס/סנכרון):
  כל פעם שנקלט התו המיוחד RADAR_MAGIC, המכונה מתאפסת למצב ראשוני. שדות העבודה מתאפסים (index=0, payload_index=0) ונתוני ההודעה החדשה מתחילים להיבנות.

- STATE_CMD (שלב הפקודה):
  התו הבא אחרי MAGIC נחשב כ־ command_id. נשמר בבאפר. לאחר מכן המכונה עוברת למצב STATE_LEN.

- STATE_LEN (שדה אורך):
  התו המתאים משמש כאורך המטען.(payload_len)
  - אם האורך גדול מהמותר (> RADAR_MAX_PAYLOAD_LEN), יש reset חזרה ל־ (STATE_MAGIC) והפונקציה מחזירה שגיאה.(-1)
  - אחרת, אם האורך אפס – המכונה ממשיכה ישר למצב STATE_CRC.
  - אם האורך חיובי – היא עוברת ל־STATE_PAYLOAD.

- STATE_PAYLOAD (טעינת הפיילוד):
  תווים נכנסים נטענים למערך payload. כאשר payload_index מגיע לערך payload_len, המכונה מתקדמת ל־STATE_CRC.

- STATE_CRC (בדיקת תקינות) :
  התו נקלט כשדה CRC. הפונקציה מחשבת CRC חדש (computed_crc) ומשווה.
  - אם תקין – ההודעה מוכרזת כשלמה msg_out, מתעדכן, ומוחזר ערך 1.
  - אם השוואת CRC נכשלה – מוחזר 1- (שגיאה).
  בשני המקרים יוצאים חזרה ל־ STATE_MAGIC לצורך סנכרון חבילה חדשה.

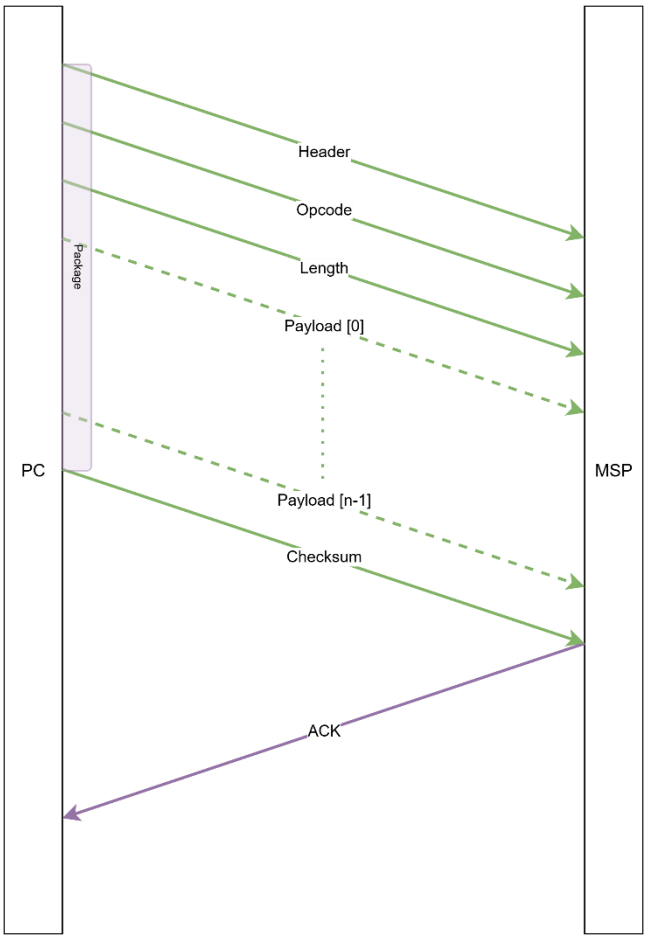
מעבר מצבים

- כל MAGIC חדש STATE_CMD → איפוס וסנכרון, בלי קשר למצב הנוכחי.
- STATE_CMD → STATE_LEN אחרי קריאת command.
- STATE_LEN → STATE_PAYLOAD אם payload_len > 0.
- STATE_LEN → STATE_CRC אם payload_len == 0.
- STATE_PAYLOAD → STATE_CRC לאחר קריאת כל המטען.
- STATE_CRC → STATE_MAGIC תמיד (אחרי בדיקה).

עקרון הפעולה

המכונה בעצם בונה פרוטוקול חבילות:

- בכל תו נכנס מתבצע קידום שלב מתאים.
- שגיאות או MAGIC חדש איפוסי מחזירים את ה־ FSM לפתיחה מחדש.



תקשורת בתצורת file mode

פרוטוקול התקשורת שלנו מבוסס על פרוטוקול Stop and Wait, כאשר אנחנו שולחים נתונים גדולים מהמחשב אנחנו נחלק את ההודעה לחבילות.
עבור קבצים גדולים אנחנו קודם נחלק את הקובץ למספר חלקים מכיוון שאנחנו מוגבלים בגודל הקבצים שהבקר יכול לקבל בכל שליחה, לאחר שחילקנו אנחנו נשלח הודעה שתודיע לבקר שאנחנו מתחילים שליחת קבצים גדולים או פקודה שמתעסקת בקבצים שכבר שלחנו.

לאחר מכן נשלח את חלקי הקבצים, עבור כל חלק אנחנו מצפים לקבל ACK כאשר בדיקת תקינות הקובץ מתבסס על parity check ובנוסף כל חלק מוסיף לבדיקת ה-CRC הכללי של הקובץ הגדול.

לאחר ששלחנו את כל החלקים המחשב שולח הודעה שסיימנו לשלוח את כל החלקים ומסיימים תקשורת.

שכבת מסגור(Frame)

כל הודעה עטופה במסגרת קבועה:

- **SOF**: '@' (0x40)
- **CMD**: בית פקודה ASCII) לאותיות, או מזהה בינארי בפקודות רדאר(
- **LEN**: אורך המטען (Payload) בבייטים
- **PAYLOAD**: 0..255 בייט
- **CRC8**: XOR על CMD + LEN + PAYLOAD
- **EOF**: '#' (0x23)

דוגמה כללית (הקס):
40 | <CMD> | <LEN> | ...payload... | <CRC8> | 23

ACK/NACK

הבקר מחזיר **ACK** קצר באותו מסגור:
@ A <code> # ⇒ 40 41 <code> 23

- **0x06** = ACK_OK
- **0x07** = ACK_NO_SPACE
- **0x09** = ACK_NOT_FOUND
- **0x0B** = ACK_BAD_SEQ

שכבת פרוטוקול יישום - S/C/E (Stop-and-Wait)

למשלוח נתונים גדולים (קבצים) ולפעולות מערכת מאוגדות, משתמשים בשלוש פקודות־על:

1) START – 'S' (0x53)

<u>Payload:</u>

$$[\,OP \mid FLAGS \mid TOTAL\_HI \mid TOTAL\_LO \mid CRC16\_LO \mid CRC16\_HI \mid \ldots OP - SPEC \ldots]$$

- OP - קוד פעולה לוגי:
  - 0x01 UPLOAD
  - 0x02 LIST
  - 0x04 DELETE
  - 0x05 REPACK
  - 0x06 LDR (כיול)

2) CHUNK – 'C' (0x43)

<u>Payload:</u>

$$[\,SEQ \mid N \mid DATA\ldots\,]$$

- SEQ – מספר החלק מתוך ההודעה השלמה.
- N – אורך המקטע (למשל 24B).
- DATA – תוכן המקטע.

התנהגות:

אם ה־SEQ הצפוי נכתב לפלאש, מצטברים CRC-16 ו־CRC-8 (הקובץ) ומחזירים ACK_OK. אם התקבל דופליקט (SEQ הקודם) מחזירים ACK_OK בלי לכתוב שוב. אם SEQ שגוי ACK_BAD_SEQ. אם קיים חריגת אורך / ERR_SIZE ERR_FORMAT.

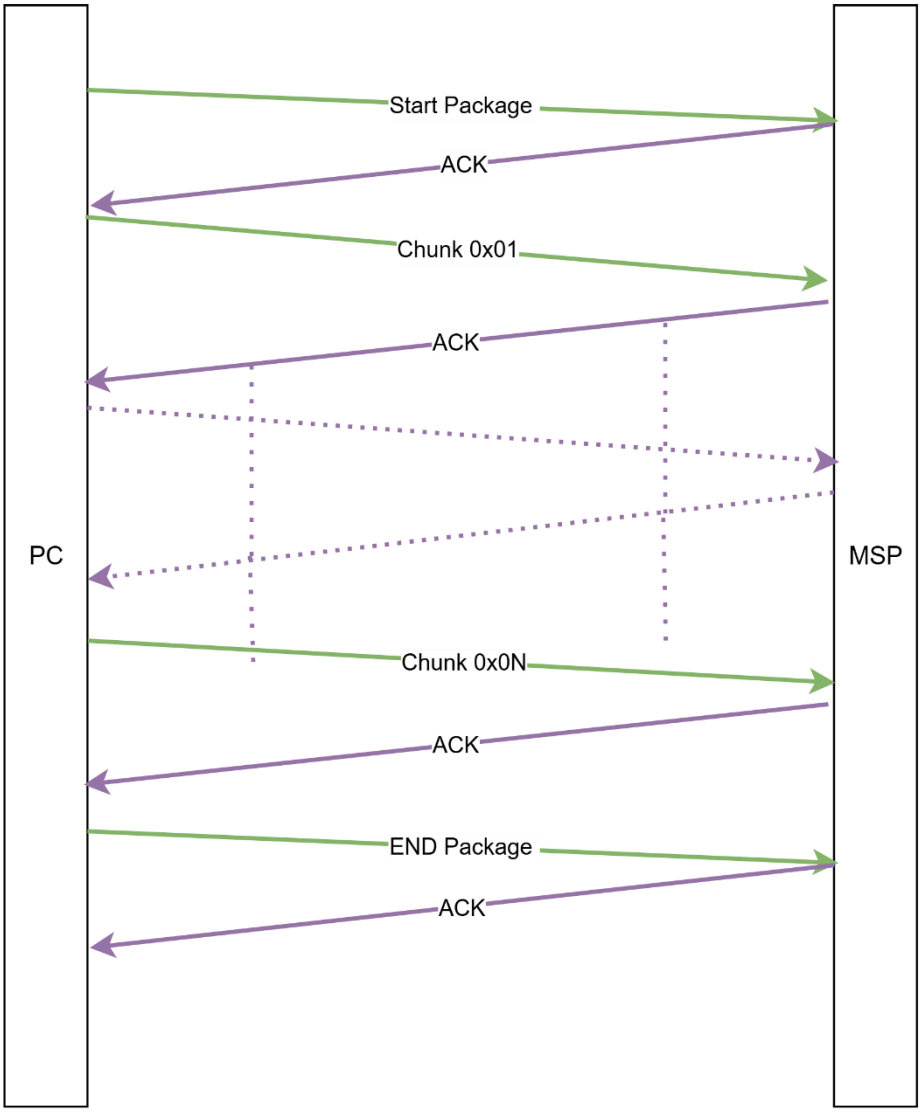
3) END – 'E' (0x45)

בלי Payload. המיקרו מבצע מספר דברים:

- מאמת recv == total ו־CRC16_run == CRC16_goal.
- בונה רשומת FAT ומעדכן file_alloc_ptr.
- מחזיר ACK_OK או שגיאה (ERR_CRC/ERR_SIZE/ERR_STATE).

פעולות לוגיות (OP) טיפוסיות

- **UPLOAD (0x01):** START (עם total+crc16+ftype+meta) → רצף CHUNK-ים. END → בסיום נוצרת רשומה בטבלת FAT.
- **LIST (0x02):** START עם total=0. הבקר שולח לפני ה־ACK את כל רשומות FAT, כל רשומה 8B
  [meta, type, sizeLo, sizeHi, startLo, startHi, crc8, reserved]
- **DELETE (0x04):** START עם payload[6]=name_id מחזיר ACK_OK/ACK_NOT_FOUND.
- **REPACK (0x05):** START ללא Payload נוסף; דוחס קבצים "חיים" לראש האיזור ומחזיר ACK_OK.
- **LDR כיול:** START מסרים לפי מימוש (קריאת/כתיבת דגימות InfoC/InfoD).
-


PC
MSP
Start Package
ACK
Chunk 0x01
ACK
Chunk 0x0N
ACK
END Package
ACK

איך שומרים קבצים ב-Flash

חלוקת הזיכרון

- **אזור קוד** –לא נוגעים.
- **אזור קבצים (File Area)** - ארבעת הסגמנטים 2-5 של ה-Flash (סה״כ 2KB בפרויקט שלנו), בהם נשמר תוכן הקבצים עצמם. הכתיבה מתבצעת מלמעלה-למטה (top-down).
- **אזור מידע (Info,סגמנט B)** כאן נשמרת טבלת FAT קטנה וקבועה, כדי שהמערכת תדע, גם אחרי reset, איפה כל קובץ יושב ומה גודלו.

פריסת הקובץ

כשמעלים קובץ:

1. הבקר מחשב את גודל ההקצאה: גודל הקובץ + ריפוד ל-16 ביט אם צריך (אם הכמות אי-זוגית מוסיפים 0xFF).
2. בודקים שיש מקום:file_alloc_ptr - need >= FILE_AREA_START.
3. מוחקים מראש את הסגמנטים שאליהם נכתוב (מחיקה היא ברמת סגמנט).
4. כותבים את הנתונים בפועל בפעימות (chunks) אל הכתובת שחושבה תוך שמירה על כללי כתיבת Flash של MSP430 (כתיבת 16 ביט, המתנה לסיום וכו׳).
5. בסוף יוצרים רשומה ב-FAT ומעדכנים file_alloc_ptr לכתובת החדשה.

טיפ טכני: אם הגודל אי-זוגי, אנחנו כותבים בייט אחרון ואז "מרפדים" את המילה העליונה ב-0xFF כדי לא לשבור זוגיות כתיבה.

טבלת ה FAT-(ב-Info B)

טבלת FAT היא מבנה קטן וקבוע שנשמר בסגמנט ה-(Info (B, ולכן לא "נמחק" ב-reset. היא מכילה:

- **magic-** בייט זיהוי (0x5A) כדי לדעת שהטבלה תקינה ולא "זבל".
- **num**– מספר הקבצים בטבלה.
- **crc16-** תקינות הטבלה (ראו בהמשך).
- **[]entries-** מערך רשומות; לכל קובץ רשומה בגודל 6B:

- start- מיקום הקובץ ב-File Area במונחי **מילים** (כלומר כתובת בסיס = FILE_AREA_START + start*2).
- size- גודל הקובץ בבתים.
- type- סוג (טקסט/סקריפט וכו').
- meta- 4 סיביות **flags** + 4 סיביות **name_id**(מזהה קובץ לוגי 0...15).
- crc8- תקינות מהירה של תוכן הקובץ עצמו.

ה-FAT כולו נשמר בפעולת **fat_save()**:

1. מעדכנים magic.
2. מחשבים crc16 של כותרת+רשומות(ללא שדה ה crc-עצמו).
3. מוחקים את סגמנט ה-Info (B) וכותבים את 64 הבייט של הטבלה מחדש.

ב-**fs_init()** בעת אתחול:

- קוראים את הטבלה מה-Info.
- בודקים magic ו-crc16. אם משהו לא תקין – מאפסים לטבלת ריקה ושומרים אותה.
- מחשבים מחדש את file_alloc_ptr ע"פ הרשומות (מחפשים את הכתובת הנמוכה ביותר שאליה מגיע קובץ חי).

<u>למה צריך CRC16 ולמה גם CRC8?</u>

CRC16(על כל הקובץ)

- אנחנו משתמשים ב-CRC-16/IBM (פולינום 0xA001, LSB-first) כל צ'אנק שנכנס מעדכן את ה-CRC במקום לחכות לסוף.
- זה מאפשר לנו לגלות שגיאה בקובץ כולו (סיביות שנפגמו בדרך או בטעות שידור) בלי לשמור זמנית את כל הקובץ ב-RAM.

CRC8(XOR פשוט)

- זהו XOR של כל בתי הקובץ. הוא מהיר מאוד.
- משתמשים בו לשני מצבים טיפוסיים:

1. **שדה ב-FAT**: מאפשר בדיקה מהירה שהקובץ בפלאש כצפוי (לדוגמה, אחרי REPACK) בלי לחשב CRC16 מלא כל פעם.
2. **בפריימים קצרים**: לפעמים הפרוטוקול עצמו משתמש ב-XOR על שדות הפריים (Cmd+Len+Payload) כבדיקת תקינות מהירה.

חשוב :CRC8 לא מחליף CRC16. הוא "בדיקה מהירה" – זול לסריקות/אימותים מהירים, CRC16 הוא הבדיקה התקנית לתוכן הקובץ.

בפרויקט ביצענו אינטגרציה בין הרכיבים כדי ליצור פעולות מסוימות, סריקה פריטים בין זוויות, סריקת מקורות אור וכו'. נסקור מספר פעולות שהמערכת שלנו מבצעת.

<u>טלמטר-</u>

בפעולה זו נדרשנו למקם את מנוע הסרוו בזווית נתונה לבחירה דרך ממשק למשתמש ולהציג על מסך ה- PC את המרחק הנמדד מחיישן המרחק בזמן אמת באופן דינאמי ברזולוציה של cm (ללא רישום היסטוריית מדידות).

המימוש של חיישן המרחק בפרויקט הוא דו-שלבי:

1. <u>שידור (Trigger):</u> הבקר שולח פולס קצר של 10 מיקרו-שניות לפין ה-TRIGשל החיישן, כדי להורות לו לשדר גל קול.
2. <u>קליטה (Echo):</u> הבקר מתחיל למדוד זמן מרגע שהפולס נשלח, ומפסיק את המדידה ברגע שמתקבל פולס בחזרה בפין ה .ECHO-זמן זה הוא הזמן שלקח לגל הקול להגיע לאובייקט ולחזור, והוא פרופורציונלי למרחק.

המערכת יכולה להשתמש בחיישן בשני מצבים: במצב סריקה, בו היא מודדת מרחק למספר נקודות במרחב; ובמצב טלמטר, בו היא מבצעת מדידה רציפה של המרחק.

1. <u>כיצד ממומשת פונקציונליות המדידה בחיישן?</u>
המדידה מבוצעת על ידי שליחת פולס קצר מפין ה TRIG-של הבקר לחיישן, והפעלת טיימר המודד את הזמן עד לקבלת פולס בחזרה בפין ה .ECHO-הבקר מחשב את המרחק על בסיס הזמן הזה, בהתאם למהירות הקול.

2. <u>מי אחראי על הטיפול בנתונים וחישוב המרחק ?</u>

הבקר (MCU) אחראי על כל תהליך: הוא מודד את הזמן שחלף מרגע השידור ועד קבלת ההד, ומחשב את המרחק על בסיס מדידה זו.

3. <u>מהם הנתונים הנשלחים מהבקר למחשב ?</u>

הבקר שולח למחשב נתונים המייצגים את רוחב הפולס, שממנו ניתן לחשב את המרחק, יחד עם הזווית שבה המדידה בוצעה.

4. <u>האם יש צורך בכיול של החיישן ?</u>

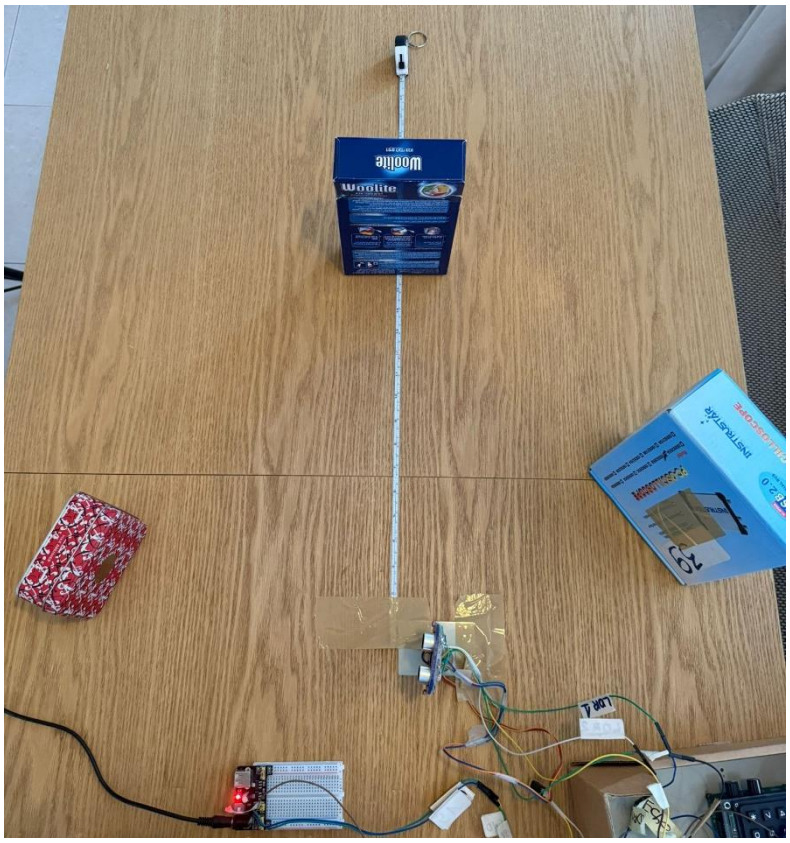
אין צורך בכיול של החיישן. הוא מבוסס על עקרון פיזי קבוע של מהירות הקול, ולכן אינו דורש התאמות נוספות כדי לעבוד באופן מדויק.

כאשר אנחנו רוצים למדוד את המרחק הבקר מבצע את המדידה 10 פעמים באופן רצוף ושולח את הדגימות כמערך לצד מחשב. המחשב מבצע שערוך למרחק באמצעות שימוש באלגוריתם לפי המדידה הכי נפוצה . השערוך נועד לייצב את המדידות ולהימנע בקפיצות וסטיות בעקבות החזרים .

סריקת עצמים במרחב-

בפעולה זו אנחנו מכניסים לפונקציה שתי זוויות שביניהן הוא יבצע סריקה ויזהה עצמים במרחב. המנוע ינוע בין הזוויות כאשר בין כל שינוי זווית קטן הוא ינוח באמצעות טיימר ובזמן הזה במקביל אנחנו מבצעים דגימה של עצמים במרחב באמצעות חיישן אולטרסוני.

ניסוי – איתור 3 גופים –



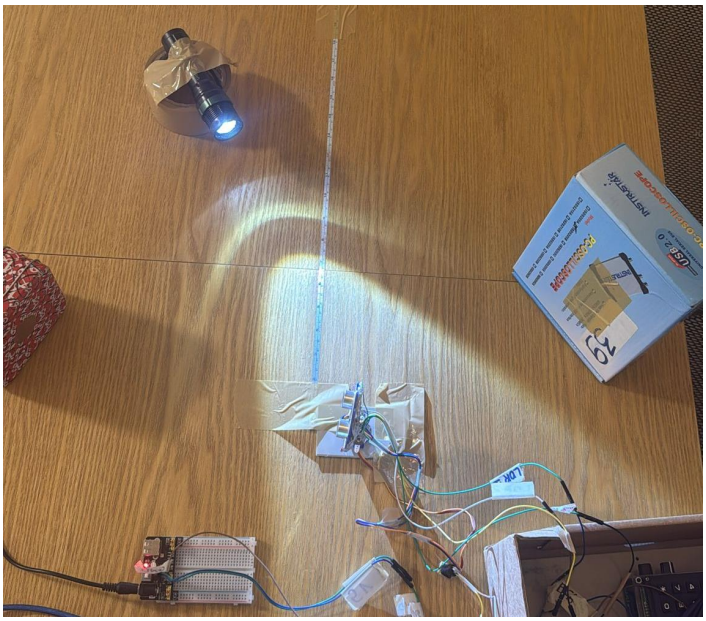
תוצאות הסריקה לאיתור הגופים

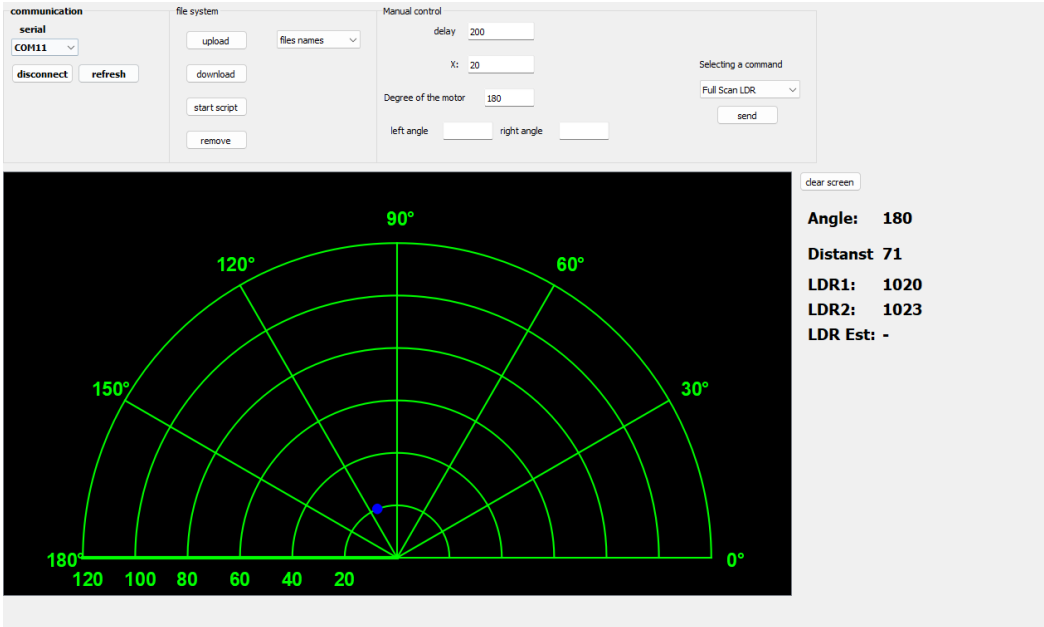
תיאור האלגוריתם -

הפונקציה extractObjectsFromScan נועדה לזהות אובייקטים מתוך נתוני סריקה של חיישן. היא מקבלת מפה שבה כל זווית מקושרת לנקודת סריקה הכוללת מרחק, ובמהלך הריצה היא בודקת אילו נקודות מייצגות אובייקט תקף, כלומר כאלו שנמצאות בתחום מרחק סביר. כאשר מתגלות נקודות רצופות שהמרחק ביניהן משתנה בצורה הדרגתית (ללא "קפיצה" גדולה), הן מצורפות לרשימה זמנית שמייצגת אובייקט פוטנציאלי. אם הרצף נקטע או נמצא קצר מדי, הוא מתאפס. כאשר מצטבר רצף באורך מינימלי (בהתבסס על זווית הסריקה), הוא נשלח לניתוח נוסף באמצעות הפונקציה analyzeObject, שהיא למעשה עושה שיערוך לרוחב של הגוף , ואז שומרים את האובייקט כתוצאה. בסיום המעבר על כל נקודות הסריקה, גם הרצף האחרון שנותר (אם קיים) מנותח ונשמר. בצורה זו, האלגוריתם מזהה בצורה אמינה קבוצות של נקודות שמהוות אובייקטים לפי קריטריונים של מרחק וקפיצה בין דגימות.

זיהוי מקורות אור במרחב-

בפעולה זו נדרשנו לזהות מקורות אור בין 180 – 0 באמצעות חיישני LDR. לפני שאנחנו מתחילים את פעולה זו נדרשנו לבצע כיול לשני חיישני ה-LDR, אנחנו מבצעים מדידה של החיישנים כל 5 ס"מ באמצעות ממשק המשתמש בצד המחשב. אנחנו מבצעים מדידה ראשונה מקבלים את המידע שנשמר בבקר וממשיכים למדידה הבאה לאחר שהרחקנו את מקור האור ב-5 ס"מ. לאחר שביצענו את הכיול אנחנו מבצעים בצד המחשב חישוב כדי לאמוד את המרחק של מקור שנזהה בהמשך באמצעות המדידות.





ניתן לראות שזיהנו רק את המקור אור בזווית הנכונה עם סטיה קלה במרחק .

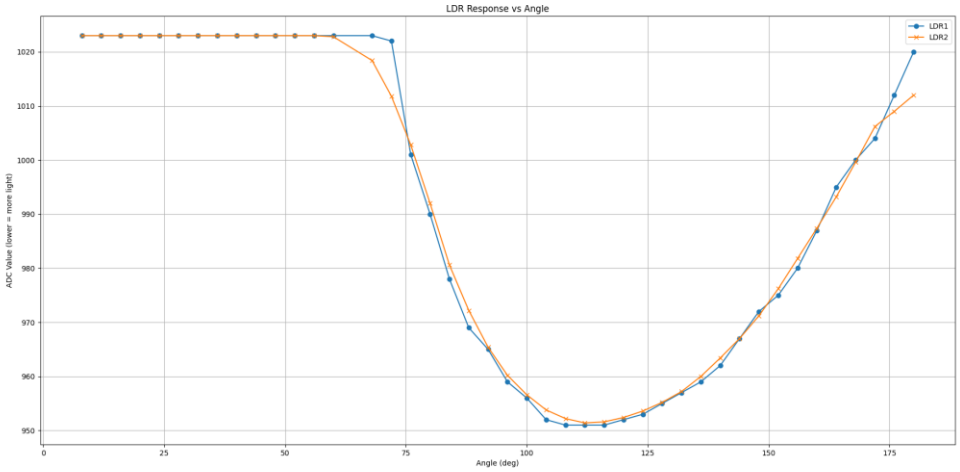
<u>תיאור האלגוריתם :</u>

השתמשנו בשני סוגים של אלגוריתמים הראשון למצוא את הזווית של מקור האור והשנייה לשערך את המרחק של מקור האור לפי הדגימות

<u>מציאת הזווית של מקור האור</u>

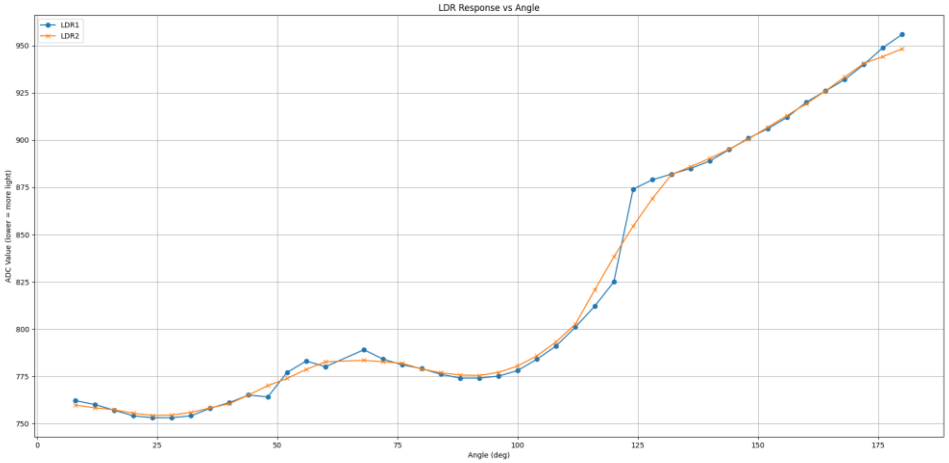
האלגוריתם לזיהוי זוויות של מקורות אור מבוסס על ניתוח נקודות מינימום מקומיות בנתוני ה־ LDRלאחר החלקה. תחילה, הנתונים מתוך הסריקה מומרים למבנה של זוגות) זווית, ערך מינימלי בין שני חיישני ,(LDR וממוינים לפי הזווית. לאחר מכן, מופעלת פונקציית החלקה (moving average) על הערכים, על מנת לסנן רעש ולחדד את המבנה הכללי של הסיגנל. החלקה זו מתבצעת באמצעות ממוצע סימטרי של ערכים סמוכים. לאחר החלקה, האלגוריתם עובר על כל הנקודות ומזהה נקודות מינימום מקומיות לפי שינוי סימן בנגזרת – כאשר הערך קטן מזה שלפניו וגדול מזה שאחריו. בכל גילוי כזה נרשמת זווית מקור האור, וניתן לקבוע סף הפרדה בין זוויות כדי למנוע גילוי חוזר של אותו מקור אור. התוצאה הסופית היא וקטור של מקורות אור מזוהים, כל אחד עם זווית וזיהוי עוצמה (score) לפי ערך ה־ LDRלאחר ההחלקה.

איך הגרף נראה כאשר שמנו מקור אור אחד :



בכתום – ההחלקה של הפונקציה כדי להימנע מסטיות
בכחול – הנקודות עצמן

איך הגרף יראה כאשר שמנו שני מקורות אור :



מציאת מרחק של מקור האור

האלגוריתם למציאת המרחק נועד להעריך את המרחק של מקור אור מהחיישנים, על סמך דגימות משני חיישני LDR ושני מערכי כיול המתארים את תגובת כל חיישן לעוצמות אור במרחקים שונים (מ-5 ס"מ ועד 50 ס"מ בקפיצות של 5 ס"מ).

השלב הראשון מתבצע באמצעות הפונקציה interpolateSingle, אשר מקבלת ערך דגימה ממד ה־ LDR ואחד ממערכי הכיול. היא סורקת את מערך הכיול ומחפשת את שני ערכי הכיול הסמוכים לערך הנמדד (לפי מיקום יחסי בגרף הכיול), ולאחר מכן מבצעת אינטרפולציה לינארית כדי להעריך את המרחק המתאים לערך זה.

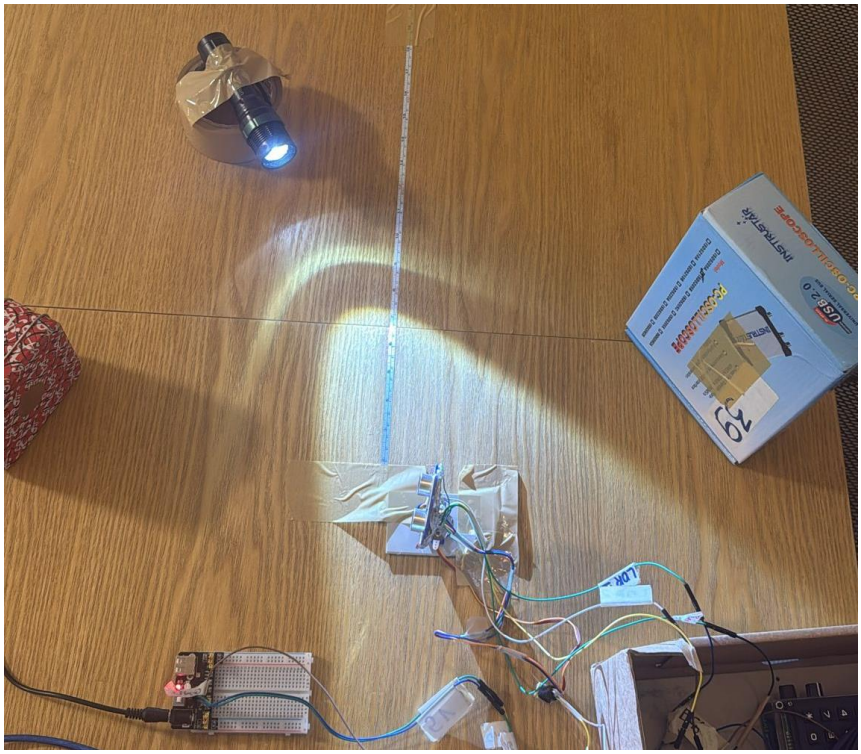
אם הערך נמדד מחוץ לתחום הכיול (גבוה מהנקודה הראשונה או נמוך מהנקודה האחרונה), מוחזר מרחק קצה (5 או 50 ס"מ).

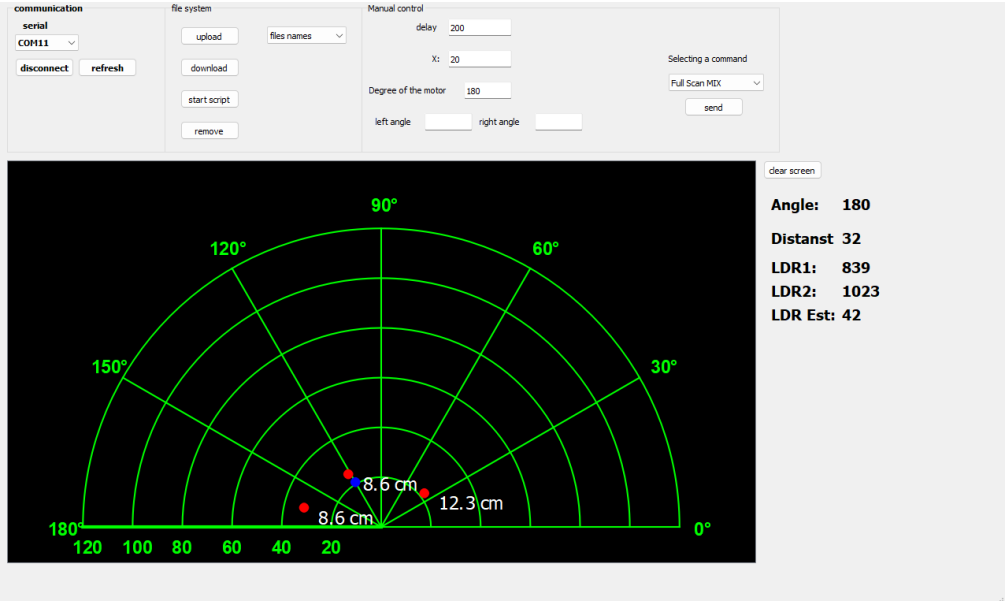
השלב השני מתבצע בפונקציה estimateDistanceFromSamples, אשר מקבלת את שתי הדגימות (LDR1 ו־ LDR2) ושני מערכי הכיול. היא מחשבת מרחק בנפרד לכל חיישן באמצעות interpolateSingle, ואז קובעת את המרחק הסופי לפי ממוצע של שניהם — אלא אם אחד מהם נכשל, ואז משתמשים רק בזה שהצליח. אם שניהם נכשלו, מוחזר ערך שגיאה.(-1.0)

באופן זה מתקבל אומדן מדויק ורובוסטי למרחק מקור האור מהחיישנים על בסיס נתוני הכיול שנאספו מראש.

זיהוי מקורות אור ועצמים במרחב (בונוס)-

בפעולה זו נדרשנו לשלב בין סעיף 1 ו-3 כאשר אנחנו מבצעים סריקה אחת בין הזוויות $0 - 180$ ובין כל תזוזה של זווית אנחנו דוגמים את העצמים במרחב ואת מקורות האור. לאחר שסיימנו את הסריקה אנחנו מציגים בצד המחשב את מיפוי התוצאות שלנו.





הקוד שומר את הערכים שדגמנו בצורה טובה, הדבר היחיד שאני רוצה לסדר זה שרק הערכים יהיו בכל סגמנט ושהם יתחילו מהכתובת ההתחלתית של כל סגמנט. כך כאשר אני ארצה לקרוא את הערכים הם יהיו בתחילת כל סגמנט.