אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

הפקולטה למדעי ההנדסה

המחלקה להנדסת מכונות

קורס המיקרו-מחשב במערכות מכניות 362-1-4201

דוח מסכם לפרויקט המסכם:

תכנות רובוט משובץ מחשב
לעקיבה אחר מסלול וביצוע חניה אוטונומית

מגישים:

עדי רון, ת"ז: 209715952

יובל מרמור, ת"ז: 209191568

מרצה הקורס: ד"ר שי ארוגטי

עוזר הוראה: עמית גדג'

אדר תשפה

מרץ 2025

תקציר

דוח פרויקט זה מתאר את מימושה של מערכת רובוטית אוטונומית המבצעת עקיבה אחרי מסלול וחניה אוטונומית. הרובוט מבוסס על פלטפורמה דיפרנציאלית עם שני מנועי DC ומערכת היגוי המאפשרת תנועה מדויקת. המערכת כוללת גם מערך חיישנים המאפשר זיהוי של קו שחור על משטח לבן וביצוע עקיבה מדויקת אחרי מסלול. הרובוט מצויד גם בחיישן מרחק המאפשר לו לזהות חניות פוטנציאליות ולהחליט האם הן תפוסות או פנויות, תוך שהוא מבצע חניה ברוורס במקרה של חניה פנויה.

בכדי לנהל את תנועת הרובוט, פותח אלגוריתם לניהול עקיבה אחרי המסלול באמצעות חיישני החזר אור, בשילוב עם אלגוריתם PID לכיוון המנועים, שמאפשר תנועה יציבה ומדויקת. מערכת החניה מבוססת על זיהוי קו המסמן מקום חניה, ועל חיישן מרחק המוודא כי מקום החניה פנוי. לאחר זיהוי חניה פנויה, הרובוט מבצע תמרון חניה אוטונומי, מחכה 5 שניות ואז חוזר למסלול.

הפרויקט מציג את השימוש בעקרונות בקרה, עיבוד אותות מחיישנים בזמן אמת ואלגוריתמים לניווט אוטונומי. תוצאות המערכת מראות שהרובוט מסוגל לנוע בצורה יציבה על המסלול ולבצע חניה אוטונומית עם תגובה מהירה לשינויים בסביבה. המסקנה העיקרית היא שהמערכת יכולה להיות יעילה ויציבה גם במצבים משתנים, עם מקום לשיפורים נוספים במערכת החיישנים ובדיוק החניה.

תוכן העניינים

רשימת איורים

רשימת טבלאות

רשימת סימנים

סימן	יחידות	תיאור
k	-	קבוע המכפיל את ערך התיקון של הרובוט בזמן העקיבה אחר המסלול.
K_p	-	מקדם פרופורציונלי בבקר PID
K_i	-	מקדם אינטגרלי בבקר PID
K_d	-	מקדם דיפרנציאלי בבקר PID
$X_{[n]}$	-	ערכי המדידה הנוכחית (לפני סינון)
$Y_{[n]}$		ערכי המדידה הקודמת (אחרי סינון)
α	-	קבוע המסנן, אומר כמה משקל יש למדידה הנוכחית לעומת המדידה הקודמת (המסוננת)

טבלה 1: רשימת סימנים

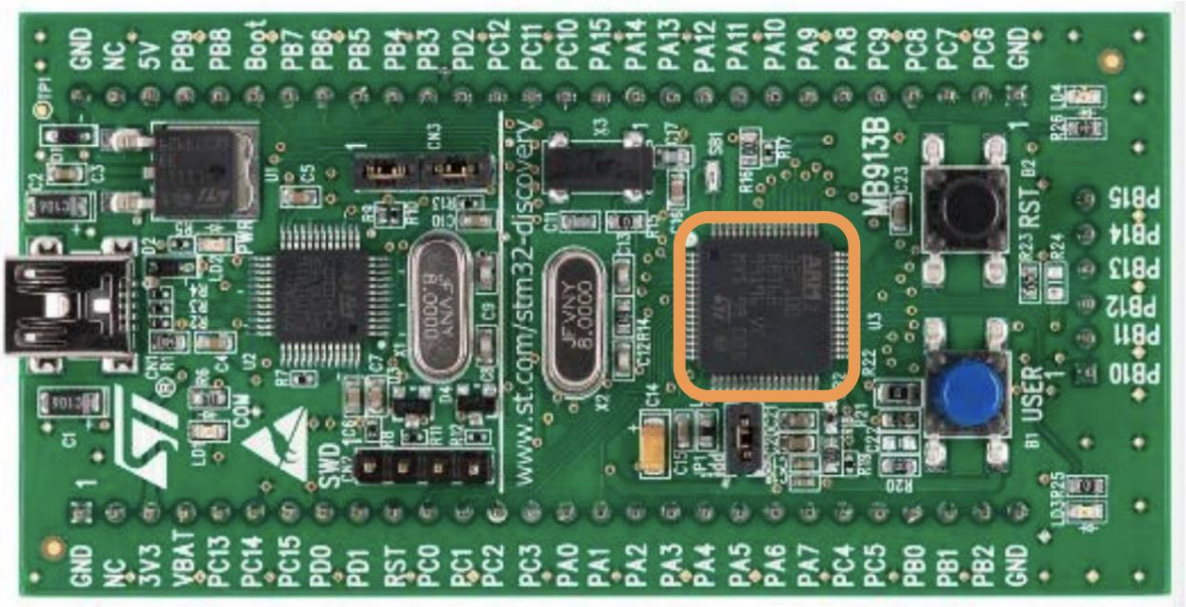
1. מבוא

1.1. תיאור המשימה

הפרויקט המתואר מציג מימוש של מערכת רובוטית אוטונומית המשלבת בקרת מעקב מסלול עם יכולות ניווט וחניה חכמה. המערכת מבוססת על פלטפורמה רובוטית דיפרנציאלית הכוללת שני מנועי DC עם מערכת היגוי המאפשרת תנועה מדויקת במישור. הרובוט מצויד במערך חיישנים מתקדם הכולל סרגל של שמונה חיישני החזר אור בתצורה לינארית בתחתית הרובוט, וחיישן מרחק אופטי בחזית. מערך החיישנים מאפשר לרובוט לזהות ולעקוב אחר מסלול המסומן באמצעות קו שחור על משטח לבן, תוך שמירה על מהירות נסיעה קבועה הניתנת לשינוי על ידי המשתמש. המערכת כוללת אלגוריתם מתקדם לזיהוי נקודות חניה פוטנציאליות, המסומנות באמצעות קווים ניצבים למסלול הנסיעה. בעת זיהוי סימון חניה, הרובוט מפעיל אלגוריתם לבדיקת תפוסת החניה באמצעות חיישן המרחק האופטי. במקרה של זיהוי חניה פנויה, המערכת מבצעת תמרון חניה אוטונומי, הכולל את היכולת לבצע חניה ברוורס בהתאם לתכנון מראש. לאחר השלמת תמרון החניה, המערכת ממתינה פרק זמן קצוב של 5 שניות, ולאחר מכן מבצעת תמרון יציאה וחזרה למסלול המקורי להמשך נסיעה. הפרויקט מדגים יישום מעשי של עקרונות בקרה, עיבוד אותות מחיישנים בזמן אמת, ותכנון אלגוריתמים לניווט אוטונומי.

1.2. המיקרו-בקר

המיקרו-בקר בו נעשה שימוש בפרויקט הינו מסוג STM32F100RBT6B, אשר מותקן על גבי ערכת הפיתוח STM32VLDISCOVERY כפי שניתן לראות באיור 1.

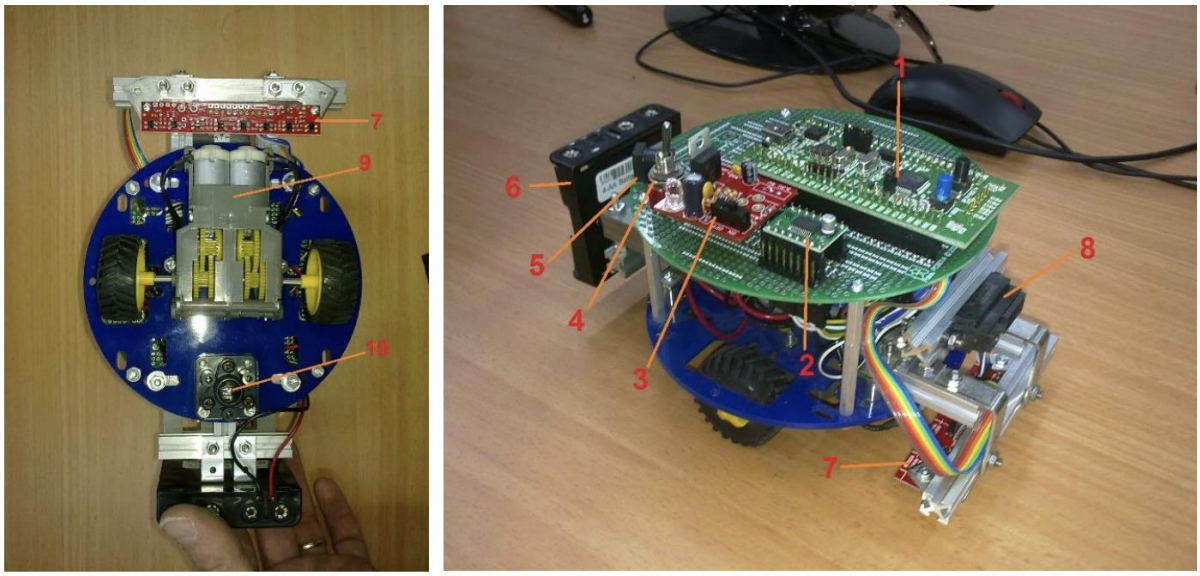


איור 1: ערכת הפיתוח. בריבוע הכתום מסומן המיקרו-בקר מסוג STM32F100RBT6B.

במסגרת מימוש האלגוריתם הנדרש עבור הרובוט נעשה שימוש במספר יחידות ואפשרויות הפעלה שונות של המיקרו-בקר. בין היתר נעשה שימוש ביחידת ה-RCC של המיקרו-בקר. יחידה זו כוללת בקר אתחול ואותות שעון המסופקים מהליבה לרכיבים הפריפריאליים שהוגדרו. כמו כן, נעשה שימוש בבקר ניהול הפסיקות של המעבד (NVIC). בקר זה מאפשר לנהל את הפסיקות המתרחשות במהלך התוכנית, גם עבור אירועים הקשורים במעבד עצמו וגם עבור פסיקות חיצוניות שמגיעות מכניסות GPIO השונות. כניסות אלו הן רכיבים המאפשרים להגדיר את הפינים של ערכת הפיתוח ככניסה או יציאה למטרה כללית או עבור רכיבים פריפריאליים מסוימים (לדוגמה הפינים המשמשים עבור הלדים והכפתורים הכחול והשחור). בנוסף על כך, נעשה שימוש משמעותי בקוד האלגוריתם בטיימרים של המיקרו-מחשב וזאת בעבור תזמון הפסיקות המנוהלות על ידי ה-NVIC ויצירת אותות PWM המשמשים להפעלת מנועי הגלגלים של הרובוט.

1.3. הרובוט

הרובוט בו נעשה שימוש בפרויקט, כפי שניתן לראות באיור 2, הינו רובוט דמוי רכב חשמלי בעל שני מנועים חשמליים, כאשר כל מנוע מסובב את אחד מגלגלי הרובוט. הרובוט כולל מספר רכיבים אלקטרוניים התורמים לפעילותו. ישנו מייצב מתח הינתן להזנה על ידי ספק חיצוני או סוללות ובורר כניסת המתח המאפשר חיבור וניתוק של הספק החיצוני. המערכת בנויה כך שהחיישנים וערכת הפיתוח מקבלים מתח דרך חיבור ה-USB או דרך ספק חיצוני ואילו המנועים מקבלים מתח אך ורק מספק חיצוני. בצורה זו ניתן להפעיל את הערכה והחיישנים ולבצע דיבוג מבלי שהמנועים פועלים.



איור 2 הרובוט בו נעשה שימוש בפרויקט

כפי שצוין, מערכת ההינע של הרובוט מורכבת מתמסורת ושני מנועי DC הנשלטים בעזרת דרייבר שמחובר אליהם. הפעלת הדרייבר מתבצעת על ידי אות PWM מערכת הפיתוח. בקרת התנועה של הרכב מתבססת על תשעה חיישנים המותקנים ברובוט. ישנו סרגל בעל שמונה חיישני החזר אור המורכב בחלקה התחתון של חזית הרובוט ומיועד לאיתור המסלול שהרובוט נדרש לעקוב אחריו. רמות המתח של החיישנים הן בטווח של בין 0 ל-3 וולט, בהתאם לכמות האור המוחזרת אליהם מהמשטח. בנוסף ישנו חיישן המרחק המורכב

בחלקו האמצעי של הרובוט ופונה קדימה. ברובוט בו עשינו שימוש טווח המדידה של חיישן המרחק הינו בין 4 ל-30 ס״מ, כאשר ערכי המתח המתקבלים מהחיישן נעים בין 3 ל-0.3 וולט בהתאמה.

2. גישת הפתרון

2.1. עקיבה אחר מסלול

הרובוט משתמש במערך של 8 חיישנים המסודרים בפריסה ישרה בחזיתו על מנת לזהות את קו המסלול. כל חיישן מחזיר ערך אנלוגי המייצג את עוצמת ההשתקפות של המשטח שמתחתיו. כדי להפחית רעשים ושינויים פתאומיים במדידות, הערכים עוברים סינון באמצעות מסנן מעבר נמוך תוכנתי. המסנן מחושב באמצעות ממוצע משוקלל בין הערך הנוכחי לערך הקודם, בהתאם למשוואה הבאה: :

$$newVal = a \cdot newVal + (1 - \alpha) \cdot oldVal$$

כאשר $0 \leq \alpha \leq 1$ הוא קבוע המסנן, $newVal$ הוא ערך המדידה הנוכחית ו $oldVal$ הוא ערך המדידה הקודם לאחר סינון. לאחר תהליך של ניסוי וטעיה נמצא כי הערך האופטימלי הוא $\alpha = 0.8$

אלגוריתם המעקב אחר הקו מבוסס על חישוב תיקון מסלול (correction value) אשר מחושב באמצעות שילוב ערכי החיישנים עם סט משקלים קבועים מראש:

$$[-2, -2, -1.5, -1, 1, 1.5, 2, 2]$$

המשקלים נבחרו כך שכאשר הרובוט ממורכז על הקו, התרומה מהחיישנים הימניים והשמאליים מאזנת זו את זו, והתיקון המתקבל שווה לאפס. ערכים גבוהים יותר בקצוות נמצאו לא יציבים בניסויים, ולכן נבחרה מערכת משקלים זו.

ערך התיקון המחושב משמש להכוונת מהירות המנועים באופן הבא:

- כאשר הרובוט נוטה שמאלה (החיישנים הימניים מזהים את הקו), המנוע הימני יוגבר והמנוע השמאלי יואט.
- כאשר הרובוט נוטה ימינה (החיישנים השמאליים מזהים את הקו), המנוע השמאלי יוגבר והמנוע הימני יואט.

על מנת למנוע תיקונים חדים ולשמור על תנועה יציבה וחלקה יותר, ערך התיקון מועבר לבקר PID (פרופורציונאלי, נגזרת ואינטגרלי). השימוש ברכיב הגוזר (D) מאפשר לרובוט להפחית את התנודות ולמנוע תיקונים מיותרים, והחלק האינטגרלי (I) עוזר בהפחתת טעויות ממושכות בתהליך. עם זאת, התגובה למידע מהחיישנים היא מיידית כך שהרובוט מגיב במהירות לשינויים במסלול. המשקלים והפרמטרים של בקר ה-PID כוונו באמצעות ניסוי וטעיה תוך ביצוע ניסויים מעשיים והתאמת הערכים עד להשגת נסיעה יציבה ומדויקת, הסבר מפורט יותר על מימוש בקרת הPID בקוד מופיע בפרק 3.

לבסוף, על מנת לשלוט במהירות המנועים הועברה פקודה המבוססת על חיבור של מהירות בסיס(basespeed) עם ערך התיקון (correction value) המוכפל בקבוע k, הפקודה עבור המנועים ניתנת על פי המשוואה הבאה:

$$MotorSpeed = basespeed \pm (correctionValue \cdot k)$$

כאשר כל מנוע מחבר/מחסר את correctionValue המתוקן בהתאם לכיוון הנסיעה הרצוי. השימוש במשוואה זו מאפשר לרובוט להתאים את המהירות של כל מנוע באופן דינמי על פי המידע שהתקבל מהחיישניים ובקרת הPID ובכך להבטיח תנועה חלקה ומדויקת על גבי הקו.

2.2. חניה אוטונומית

2.2.1. זיהוי האם החניה פנויה ואופן התמודדות עם המקרים

השלב הראשוני בתהליך החניה הוא זיהוי של קו החניה. הרובוט מבצע את הזיהוי באמצעות חיישנים הממוקמים בקצוות המערך, כלומר ארבעת החיישנים הימניים וארבעת השמאליים ביותר. כאשר שלושה מתוכם מזהים פס שחור הרובוט מסיק כי מדובר בקו חניה .לאחר זיהוי קו החניה הרובוט משנה את מצבו למצב חניה (state=p) ומאחסן את כיוון החניה בתוך המשתנה parkingState לצורך המשך הפעולה:

- חניה ימנית נשמרת בתור parkingState=r
- חניה שמאלית נשמרת בתור parkingState=l
- אם הרובוט לא זיהה חניה אז parkingState=0

לאחר מכן, הרובוט מבצע סיבוב כולל של 90 מעלות עם הפנים לכיוון החניה ובודק את זמינותה באמצעות חיישן המרחק. הסיבוב מחולק לסיבובי ביניים של 20+20+50 מעלות כאשר לאחר כל סיבוב הרובוט בודק האם קיים מכשול בדרך. אם המרחק שנמדד קטן מהסף שנקבע, הרובוט מסיק כי החניה חסומה, מבצע סיבוב נוסף של 90 מעלות וחוזר לנסיעה רגילה.

בדיקה זו מתבצעת באמצעות מנגנון פסיקה (Interrupt) כדי לזהות מכשולים. הפסיקה מופעלת על ידי Timer7 של הבקר ומתרחשת בכל איפוס של הטיימר (Update Event), כלומר מספר פעמים בשנייה. במהלך כל הפעלת הפסיקה אם חיישן המרחק מזהה אובייקט בטווח הקטן מהסף שהוגדר, הבקר מעדכן את המשתנה obstacleFlag. ערך זה נבדק בקוד במהלך סיבובי הביניים כך שאם obstacleFlag=1 הרובוט עוצר את תהליך החניה, מסתובב חזרה לכיוון המסלול הראשי, נוסע מעט לאחור כדי לחזור לקו המסלול ומבצע מעבר למצב נסיעה.(state='d')

2.2.2. תהליך ביצוע החניה

במידה והחניה אכן פנויה, הרובוט עובר לתהליך ביצוע החניה. כל השלבים בתהליך זה מתבצעים לפי סדר קבוע, והשינויים בין מצבים שונים (ימין או שמאל) מבוצעים בעיקר בכיוון הסיבוב של הרובוט:

תחילה , הרובוט מבצע סיבוב של 180 מעלות כך שיוכל לבצע חניה ברוורס. הרובוט מבצע את הסיבוב בכיוון ההפוך לסיבוב בדיקת החניה שהתבצע קודם לכן, זאת על מנת לצמצם את סיכויי השגיאה במיקום ובזווית.

לאחר הסיבוב, הרובוט נוסע לאחור למשך זמן קבוע מראש, זמן אשר תלוי במרחק שנקבע על מנת שהרובוט יכנס בצורה מדויקת. על הרובוט להישאר במקום במשך 5 שניות בדיוק, כפי שנדרש בתהליך.

2.2.3. תהליך השתלבות חזרה במסלול

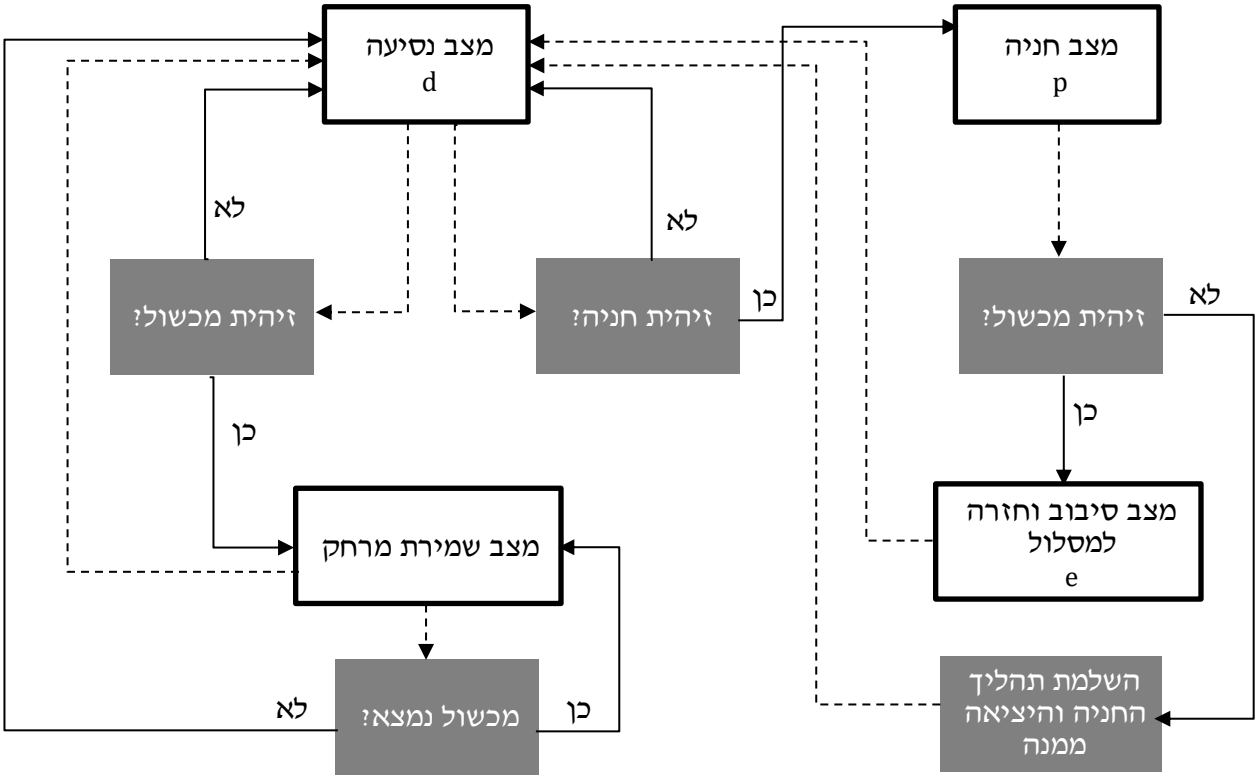
לאחר ביצוע החניה הרובוט נדרש להשתלב חזרה במסלול הנסיעה הקבוע. הרובוט מתחיל בנסיעה קדימה במשך זמן קבוע עד שהוא מגיע למיקום היציאה מהחניה. לאחר שהרובוט יצא מהחניה, הוא מבצע סיבוב של 90 מעלות לכיוון המסלול המקורי בו נסע קודם. בסיום סיבוב זה, הרובוט נוסע מעט לאחור על מנת להחזיר את עצמו למסלול ומבצע מעבר למצב נסיעה (state='d'). ברגע זה הרובוט שב לפעול על פי אלגוריתם המעקב אחרי הקו.

2.3. פונקציית MyDelay

בקוד נעשה שימוש בטיימר 2 ליצירת דיליי מדויק. הטיימר מוגדר עם prescaler כך שפעימת השעון מתרחשת כל 1ms. בדרך זו ניתן לספור את פעימות השעון של הטיימר ולדעת כמה זמן עבר. פונקציה זו שימשה בין היתר להמתנה של 5 שניות במהלך החניה לפני היציאה ממנה.

2.4. תרשים זרימה– מצבי הפעולה של הרובוט

באיור 3 מוצג תרשים זרימה המתאר את מצבי הפעולה של הרובוט והמעבר בניהם:



איור 3 – תרשים זרימה של מצבי הפעולה של הרובוט

2.5. בונוס – מצב שמירת מרחק

בנוסף לדרישות הרובוט כולל גם מצב שמירת מרחק. בזכות הפסיקה של טיימר 7 הרובוט יודע לזהות מכשולים גם במצב נסיעה d ולכן הוחלט לנצל זאת ולהוסיף פיצ'ר חדש לרובוט:

כאשר הרובוט רואה מכשול באמצע הדרך הוא נעצר ומחכה שיעבור. אם מדובר למשל ברכב איטי, הרובוט שומר ממנו מרחק קבוע. אם מכשול מתקרב לכיוון הרובוט, הרובוט זז לאחור וממשיך לשמור ממנו מרחק קבוע. לאחר שהמכשול נעלם הרובוט חוזר למצב נסיעה רגילה.

3. מימוש ודוגמאות קוד

3.1. הסבר הלולאה הראשית של התכנית

באיורים 4-6 מוצגת הלולאה הראשית המתנהלת בפונקציית ה- main של אלגוריתם הפעולה של הרובוט. לולאה זו מתחילה לרוץ לאחר ההגדרות הראשוניות של המערכת (אתחול המנועים, החיישנים, הפינים ובקר הפסיקות).

```
while (1)
{
  //שמירת ערך המדידה הקודם
  m0_old=m0;
  //קריאת החיישנים וביצוע סינון לוו פס
  m0=Read_Sensors();
  lowPassFilter((int*)m0.array_sensor,(int*)m0_old.array_sensor, 8);
  lowPassFilter((int*)&m0.distance,(int*)&m0_old.distance, 1);
  switch(state)
  {
    case 'd':  // מצב נסיעה רגיל
      if(obstacleFlag) //אם היתה מכשול תשמור על מרחק ביטחון מחנו
      {
        escapeSpeed = PID(m0.distance, obstacleThreshold, 1, 1, 0.01, &old_error_distance, &i_Temp_distance);
        Motor_Drive(MOTOR1,escapeSpeed);
        Motor_Drive(MOTOR2,escapeSpeed);
      }
      else //אחרת עקוב אחרי הקו כרגיל
      {
        //בדיקת קו חנית
        parkingState=detectParkingLine();
        if (parkingState!=0)
          state = 'p';
        else
        {
          //חישוב ערך תיקון
          correction_val = getCorrectionValue();
          correction_val=PID(correction_val,  0,  1 , 2, 0.01, &old_error_direction, &i_Temp_direction) *-1;
          //נתן פקודות למנועים
          //motor1=שמאל מנוע, motor2 = ימין מנוע
          Motor_Drive(MOTOR1,basespeed+correction_val*k);
          Motor_Drive(MOTOR2,basespeed-correction_val*k);
        }
      }
      break;
```

איור 4: מבנה הלולאה הראשית במצב נסיעה רגיל של הרובוט

```
        //חניה רצף תתחיל מכשול אין אם
        //מעלות 180 סיבוב
        if(parkingState=='r')
          turnByAngle('l',180);
        else if(parkingState=='l')
          turnByAngle('r',180);
        MyDelay(500);
        //לאחור סע
        Motor_Drive(MOTOR1,-basespeed);
        Motor_Drive(MOTOR2,-basespeed);
        MyDelay(2000);
        //שניות 5 חכה
        Motor_Stop(MOTOR1);
        Motor_Stop(MOTOR2);
        MyDelay(5000);
        //מהחניה צא
        Motor_Drive(MOTOR1,basespeed);
        Motor_Drive(MOTOR2,basespeed);
        MyDelay(2000);
        turnByAngle(parkingState,90);
        MyDelay(500);
        //לאחור קלה נסיעה
        Motor_Drive(MOTOR1,-basespeed);
        Motor_Drive(MOTOR2,-basespeed);
        MyDelay(800);
        Motor_Stop(MOTOR1);
        Motor_Stop(MOTOR2);
        MyDelay(100);
        //למצב נסיעה חזרה
        state='d';
        break;
    }
```

איור 5: תהליך בדיקת החניה וזיהוי מכשולים במצב חניה

```
    case 'p':  // חניה מצב
      //נסיעה קצרה קדימה
      Motor_Drive(MOTOR1,basespeed);
      Motor_Drive(MOTOR2,basespeed);
      MyDelay(1000);

      // סיבוב כולל של 90 מעלות שבמהלכו הרובוט בודק אם קיימים מכשולים
      turnByAngle(parkingState,50);
      MyDelay(500);
      for(int i=0;i<2;i++)
      {
        Motor_Stop(MOTOR1);
        Motor_Stop(MOTOR2);
        MyDelay(100);
        m0=Read_Sensors();
        MyDelay(500);
        //ציאה למצב לעבור מכשול מצאת אם (exit)
        if(obstacleFlag)
          state='e';
        turnByAngle(parkingState,20);
        MyDelay(500);
      }
      Motor_Stop(MOTOR1);
      Motor_Stop(MOTOR2);

     //בנסיעה ולהמשיך מהחניה לצאת יש מכשול נקלט אם
      if(state=='e')
      {
        //מעלות 90 סיבוב
        if(parkingState=='r')
          turnByAngle('l',90);
        else if(parkingState=='l')
          turnByAngle('r',90);
        //לאחור קלה נסיעה
        Motor_Drive(MOTOR1,-basespeed);
        Motor_Drive(MOTOR2,-basespeed);
        MyDelay(800);
        //נסיעה למצב חזרה
        state='d';
        break;
      }
```

איור 6: ביצוע חניה ברוורס ויציאה מהחניה חזרה למסלול

ניתן לראות בשורות 62-66 את שמירת ערך המדידה הקודם, קריאת החיישנים וביצוע פילטור על ידי פונקציית lowPassFilter. הפילטור מופעל על מערך חיישני הקו (עם 8 חיישנים) ועל ערך חיישן המרחק (ערך בודד). לאחר קריאת החיישנים וסינון הרעשים, מתבצעת ה- State Machine (שורה 67) באמצעות פקודת switch על המשתנה state שמגדיר את מצב הפעולה הנוכחי של הרובוט.

במצב נסיעה רגיל (case 'd') בשורות 69-93 הרובוט בודק תחילה אם זוהה מכשול (שורה 70). במידה וכן, הרובוט מפעיל בקרת PID לשמירת מרחק בטוח מהמכשול ושולח את אותה מהירות לשני המנועים כדי לנסוע אחורה בקו ישר (שורות 72-74). אם לא זוהה מכשול, הרובוט בודק האם יש קו חניה באמצעות הפונקציה detectParkingLine (שורה 79). אם אותר קו חניה, הרובוט עובר למצב חניה (state = 'p'), אחרת ממשיך במעקב אחרי הקו השחור באמצעות חישוב ערך תיקון (שורות 85-86). ערך התיקון מחושב תחילה על-פי מפת החיישנים בפונקציה getCorrectionValue ולאחר מכן מועבר דרך בקר PID. הערך המתקבל משמש לתיקון מהירויות המנועים (שורות 89-90) - תוספת למנוע שמאל (MOTOR1) וחיסור ממנוע ימין (MOTOR2) או להיפך, בהתאם לכיוון הסטייה מהקו.

במצב חניה (case 'p') בשורות 95-168 הרובוט מבצע סדרת פעולות: נסיעה קצרה קדימה (שורות 97-99). סיבוב כולל של 90 מעלות בכיוון שבו התגלה קו החניה (שורות 101-102). בדיקה אם החניה פנויה (שורות 104-116) על ידי סיבובים קטנים ובדיקת מכשולים. אם החניה תפוסה (זוהה מכשול), הרובוט עובר למצב יציאה ('e') מבצע סיבוב 90 מעלות בכיוון ההפוך, נסיעה קלה לאחור וחוזר למצב נסיעה רגילה (שורות 120-134) אם החניה פנויה, הרובוט מבצע את תמרון החניה: סיבוב 180 מעלות, נסיעה לאחור, המתנה של 5 שניות, יציאה מהחניה וחזרה למצב נסיעה רגילה (שורות 137-166). בכל מצב, הרובוט משתמש בפונקציות עזר כמו turnByAngle לביצוע סיבובים מדויקים ו- MyDelay להמתנה מבוקרת. הלולאה הראשית מבטיחה שהרובוט פועל באופן רציף, מגיב לשינויים בסביבה ומבצע את משימותיו בהתאם למצבו הנוכחי.

3.2. שימוש בטיימרים

בפרויקט נעשה שימוש במספר טיימרים למשימות שונות שמאפשרות פעולה מדויקת של הרובוט. הטיימרים העיקריים בשימוש הם טיימר 2 , טיימר 4 וטיימר 7 .

```
//הדלייי פונקציית לטובת 2 טיימר אתחול
void Delay_Init()
{
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
  time2.TIM_ClockDivision = 0;
  time2.TIM_CounterMode = TIM_CounterMode_Up;
  time2.TIM_Period=0xFFFF;
  time2.TIM_Prescaler = 24000-1;
  TIM_TimeBaseInit(TIM2,&time2);
  TIM_Cmd(TIM2,ENABLE);
}

//waits milisec[ms] before continues
void MyDelay(uint16_t milisec)
{
  TIM_SetCounter(TIM2, 0);
  while(TIM_GetCounter(TIM2)<milisec);
}
```

איור 7: פונקציות אתחול וניהול זמן באמצעות טיימר 2

באיור 7 ניתן לראות כי טיימר 2 משמש ליצירת השהיות מדויקות באמצעות פונקציית MyDelay (שורות 244-248). טיימר זה מאותחל בפונקציית Delay_Init (שורות 232-241) מה שמאפשר ליצור השהיות. פונקציית MyDelay מפעילה את הטיימר, מאפסת את ערך המונה ומחכה עד שהמונה יגיע לערך המבוקש. בעזרת פונקציה זו, הרובוט מבצע פעולות למשך זמן מוגדר וקבוע, כגון נסיעה קדימה, סיבוב בזווית מסוימת או המתנה בחניה.

```
42  Blue_Led_Init(); //Initialize timer4-OC1  to synchronize blue led
43  TIM_Cmd(TIM4, ENABLE); //Turn blue led ON
44  Delay_Init();
45  Timer7_Init();
46
47  /******* Stuck here until button is pushed *******************/
48  while (!GPIO_ReadInputDataBit (GPIOA, GPIO_Pin_0));
49  while (GPIO_ReadInputDataBit (GPIOA, GPIO_Pin_0));
50  /*********************************************************/
51  TIM_Cmd(TIM4, DISABLE); //Turn blue led OFF
```

איור 8: אתחול המערכת והמתנה ללחיצת כפתור ההפעלה באמצעות טיימר 4

ניתן לראות באיור 8 כי טיימר 4 משמש לבקרת הלד הכחול על הרובוט. הוא מאותחל בפונקציית Blue_Led_Init (שורה 42) ומשמש לסנכרון הלד. בתחילת התוכנית הלד הכחול דולק (שורה 43) ומכבים אותו לאחר לחיצה על הכפתור (שורה 51).

```
293  //7 אתחול טיימר
294  void Timer7_Init()
295  {
296    RCC_APB1PeriphClockCmd  (RCC_APB1Periph_TIM7,ENABLE);
297
298    NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
299    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
300    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
301    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
302    NVIC_Init(&NVIC_InitStructure);
303
304    time7.TIM_Period = 2000;
305    time7.TIM_Prescaler = 5000;
306    time7.TIM_ClockDivision = TIM_CKD_DIV1;
307    time7.TIM_CounterMode = TIM_CounterMode_Up;
308    TIM_TimeBaseInit(TIM7, &time7);
309
310    TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);
```

איור 9: אתחול טיימר 7 לזיהוי מכשולים

טיימר 7 משמש למטרת זיהוי מכשולים ופועל במצב פסיקה. טיימר זה מאותחל בפונקציית Timer7_Init (שורות 296,304-310) ומופעל בתחילת התוכנית. הוא מוגדר עם Period של 2000 ו- Prescaler של 5000, מה שקובע את קצב הבדיקה של חיישן המרחק.

השימוש בטיימרים מאפשר תזמון מדויק ומחזורי של פעולות שונות בו-זמנית, ומהווה חלק מרכזי בפעולתו התקינה של הרובוט.

3.3. שימוש בפסיקות

בפרויקט נעשה שימוש במנגנון פסיקות לביצוע בדיקות מחזוריות של חיישן המרחק לצורך זיהוי מכשולים. הפסיקה מיושמת באמצעות טיימר 7 (TIM7) והיא מאפשרת לתוכנה לבצע בדיקות בזמן אמת ללא תלות בלולאה הראשית.

```
274  void TIM7_IRQHandler(void)
275  {
276    if (TIM_GetITStatus(TIM7,TIM_FLAG_Update)==SET)
277    {
278      //אם זוהה מכשול
279      if(m0.distance > obstacleThreshold)
280      {
281        obstacleFlag=1;
282        GPIO_WriteBit(GPIOC,GPIO_Pin_8,(BitAction) 1) ;
283      }
284      else //אחרת
285      {
286        obstacleFlag=0;
287        GPIO_WriteBit(GPIOC,GPIO_Pin_8,(BitAction) 0) ;
288      }
289      TIM_ClearITPendingBit( TIM7, TIM_FLAG_Update);
290    }
291  }
292
293  //אתחול טיימר 7
294  void Timer7_Init()
295  {
296    RCC_APB1PeriphClockCmd  (RCC_APB1Periph_TIM7,ENABLE);
297
298    NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
299    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1
300    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
301    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
302    NVIC_Init(&NVIC_InitStructure);
303
304    time7.TIM_Period = 2000;
305    time7.TIM_Prescaler = 5000;
306    time7.TIM_ClockDivision = TIM_CKD_DIV1;
307    time7.TIM_CounterMode = TIM_CounterMode_Up;
308    TIM_TimeBaseInit(TIM7, &time7);
309
310    TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);
```

איור 10: פונקציית טיפול בפסיקה ואתחול מערכת הפסיקות

פונקציית האתחול Timer7_Init (שורות 294-311) מגדירה את כל הפרמטרים הדרושים להפעלת הפסיקה. תחילה מופעל שעון הפריפריה של טיימר 7 (שורה 296). לאחר מכן מוגדר ערוץ הפסיקה NVIC_InitStructure עם עדיפות מתאימה (שורות 298-302). לבסוף מופעלת הפסיקה באמצעות TIM_ITConfig (שורה 310).

פונקציית הטיפול בפסיקה TIM7_IRQHandler (שורות 274-291) מופעלת בכל פעם שטיימר 7 מגיע לסוף הספירה. הפונקציה בודקת תחילה אם אכן התקבלה פסיקת עדכון (שורה 276). אם התנאי מתקיים, נבדק ערך חיישן המרחק מול הסף שהוגדר (obstacleThreshold). כאשר המרחק גדול מהסף (כלומר, זוהה מכשול), הדגל obstacleFlag מוגדר ל-1 והלד הכחול נדלק כאינדיקציה ויזואלית (שורות 281-282). אם לא זוהה מכשול, הדגל מאופס והלד נכבה (שורות 286-287). בסיום הטיפול בפסיקה, הדגל הממתין מנוקה באמצעות TIM_ClearITPendingBit (שורה 289), מה שמאפשר לפסיקה להתרחש שוב בהמשך.

השימוש במנגנון הפסיקות מספק יתרונות משמעותיים בפרויקט זה: בדיקת מכשולים בתדירות קבועה וצפויה, תגובה מהירה לנוכחות מכשולים ללא תלות בשלב הביצוע של הלולאה הראשית, ניצול יעיל של

משאבי המעבד- אין צורך לבדוק את חיישן המרחק באופן רציף, פישוט הקוד בלולאה הראשית, שיכולה להסתמך על הדגל obstacleFlag לקבלת החלטות. בלולאה הראשית, התוכנה משתמשת בערך הדגל obstacleFlag שעודכן על ידי מנגנון הפסיקה כדי להחליט אם לעבור למצב בריחה ממכשול או להמשיך במעקב אחר קו.

3.4. עקרונות שנלמדו במעבדות ויושמו בפרויקט

במהלך פרויקט זה נלמדו מספר עקרונות שנלמדו במעבדות ויושמו בפרויקט, עקרונות אלה יופיעו בתת חלק זה.

3.4.1. סינון רעשים באמצעות מסנן מעביר נמוך

לאחר קבלת קריאת החיישנים השונים מומש low pass filter על מנת לסנן רעשים מהמדידות שהתקבלו. המסנן מומש לפי הנוסחה הבאה:

$$Y_{[n]} = \alpha \cdot X_{[n]} + (1 - \alpha) \cdot X_{[n-1]}$$

כאשר α היא קבוע המסנן, $X_{[n]}$ הם ערכי המדידה ו- $Y_{[n]}$ הערכים המסוננים. פונקציית המסנן כפי שמומשה בקוד מוצגת באיור 11.

```
//אורך למערך נמוך מעביר מסנן length
void lowPassFilter(int *newVal, int *oldVal,int length)
{
  float a=0.8;
  for (int i = 0; i < length; i++)
  {
    newVal[i]=a*newVal[i]+(1-a)*oldVal[i];
  }
  return;
}
```

איור 11: פונקציית הסינון רעשים

ניתן לראות כי קבוע המסנן בקוד שלנו הוגדר להיות 0.8. ערך גבוה יותר של α נותן משקל רב יותר למדידה הנוכחית ופחות למדידות קודמות מה שמאפשר תגובה מהירה יותר לשינויים בקריאות החיישנים. הפונקציה מקבלת שלושה פרמטרים: מצביע למערך הערכים החדשים (newVal) , מצביע למערך הערכים הקודמים (oldVal) ואורך המערך (length) . הפונקציה עוברת על כל איבר במערך ומחשבת את הערך המסונן לפי הנוסחה.

השימוש במסנן מעביר נמוך חיוני במיוחד במערכת כמו הרובוט שלנו, שבה קריאות החיישנים יכולות להיות מושפעות מרעשים חשמליים, שינויי תאורה, או רעידות מכניות. הסינון מאפשר קבלת החלטות יציבות יותר ומדויקות יותר לגבי מיקום הרובוט ביחס לקו המסלול ולמכשולים.

3.4.2. בקרת ה- PID

בפרויקט זה נעשה שימוש בבקר PID לבקרת מסלול הרובוט ולשמירת מרחק בטוח ממכשולים. הבקר PID מאפשר בקרה מדויקת של המערכת באמצעות שילוב של שלושה מרכיבים: חלק פרופורציונלי (P), חלק אינטגרלי (I) וחלק נגזרת (D). מימוש פונקציית ה- PID בקוד מוצג באיור 12.

```
// [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].
float PID(int signal, int des_signal, float Kp, float Kd, float Ki, float *old_error ,float* i_Temp)
{
  //Local variables for PID
  float P,I,D,iMax = 7, iMin = -7 , error;

  //[illegible] [illegible]
  error = (des_signal - signal);
  //P
  P = Kp * error;
  //I
  *i_Temp += error;
  if (*i_Temp > iMax)
    *i_Temp = iMax;
  else if (*i_Temp < iMin)
    *i_Temp = iMin;
  I = Ki * *i_Temp;
  //D
  D = Kd * (*old_error - error);
  *old_error = error;
  //[illegible] [illegible] [illegible] [illegible]
  return P + I + D;
}
```

איור 12: מימוש פונקציית PID בקוד

הפונקציה מקבלת את האות הנמדד (signal), האות הרצוי (des_signal), קבועי הבקר (Kp, Kd, Ki), וכן מצביעים לערכי השגיאה הקודמת (old_error) והאינטגרל המצטבר (i_Temp). תחילה מחושבת השגיאה כהפרש בין האות הרצוי לאות הנמדד. לאחר מכן מחושב הרכיב הפרופורציונלי (P) על ידי הכפלת השגיאה במקדם Kp. הרכיב האינטגרלי (I) מחושב על ידי הוספת השגיאה הנוכחית לערך האינטגרל המצטבר ומוגבל לטווח שבין 7- עד 7 כדי שרכיב האינטגרל לא ימשיך להצטבר ללא הגבלה. ערך זה מוכפל במקדם Ki לקבלת רכיב ה- I. הרכיב הנגזרת (D) מחושב על ידי הכפלת ההפרש בין השגיאה הקודמת לשגיאה הנוכחית במקדם Kd. שגיאה זו שימושית לריסון תגובות מהירות של המערכת. לאחר חישוב הרכיב D השגיאה הנוכחית נשמרת עבור החישוב הבא. לבסוף, הפונקציה מחזירה את סכום שלושת הרכיבים (P+I+D) כערך הבקרה שיופעל על המערכת.

ערכי הבקר השונים (Kp, Ki, Kd) הוגדרו בהתאם למשימה הספציפית על סמך ניסוי וטעיה. הערכים שנקבעו הם דלהלן:

בזמן מצב נסיעה:

$$K_p = 1 \ , \qquad K_d = 2 \ , \qquad K_i = 0.01$$

בזמן מצב שמירת מרחק:

$$K_p = 1 \ , \qquad K_d = 1 \ , \qquad K_i = 0.01$$

4. ביצועים

הרובוט עליו עבדנו היה רובוט מספר 1. הקוד מאפשר לרובוט לנסוע במעקב אחר מסלול במהירות השווה לאות PWM של 3000. בנסיעה חופשית במישור הרובוט מסוגל לעלות על מסלולים ישרים וברוב הפעמים גם אנכים לכיוון תנועתו, להתיישר עליהם ולעקוב אחריהם, אולם הוא לא תמיד יצליח לעשות זאת אם המסלול מעוקל. אם זווית כניסתו למסלול היא לא 90 מעלות או קרוב הוא יצליח גם במסלולים מעוקלים. הרובוט, תיאורטית, מסוגל לעקוב אחרי המסלול גם כאשר מהירות המנועים (PWM) עולה ל-4000 או יותר. עם זאת, במהירויות גבוהות אלו, הוא מאבד באופן משמעותי את היכולת לשמור על דיוק במסלול. לכן, הוחלט להשאיר את PWM על ערך של 3000, שנקבע להיות ברירת מחדל. הפונקציה לזיהוי המכשולים עובדת באופן תקין כאשר המכשול מול הרובוט נמצא במרחק מספיק, אך אם המכשול מתקרב במהירות, לעתים הרובוט לא מצליח להגיב במהירות מספקת כדי לברוח ממנו. ביצועי החניה של הרובוט גם הם משתנים בהתאם לתנאי הסביבה. כאשר הרובוט מזהה קו חניה, במקרים רבים הוא מצליח לחנות במדויק בין קווי החניה, אך לעיתים, במיוחד כאשר החניה נמצאת על מסלול מעוקל, הרובוט חונה בצורה פחות מדויקת. בנוסף, בזמן בדיקת החניה, הסיבובים לפעמים אינם מדויקים כאשר החניה ממוקמת על מסלול מעוקל, מה שיכול להשפיע על יכולת זיהוי החניה הפנויה. חיישן המרחק מספק קריאות אמינות ברוב המקרים, אך כאשר המכשול מול הרובוט נמצא במרחק של 4 ס"מ ומטה, לעיתים מתקבלת קריאה לא אמינה שיכולה להתפרש בטעות כמרחק לא נכון מהמכשול או להתייחס לכך כאילו המכשול אינו קיים.

5. דיון ומסקנות

באופן כללי, הרובוט הצליח לענות על כל הדרישות שהוגדרו במפרט הטכני: מעקב אחר קו שחור על משטח לבן, זיהוי קווי חניה, בדיקת פנוי/תפוס, ביצוע חניה והמשך נסיעה לאחר זמן המתנה. בחינת הביצועים מראה כי האלגוריתמים שפיתחנו התמודדו היטב עם המורכבות של המשימה, אף שבמצבים מסוימים היו אתגרים שדרשו התמודדות ייחודית.

אחד האתגרים היה כיול מדויק של פרמטרי בקר ה- PID. הבקר שימש הן למעקב אחר הקו והן לשמירת מרחק ממכשולים, עם פרמטרים שונים (Kp, Ki, Kd) לכל משימה. תהליך הכיול של המקדמים דרש ניסויים חוזרים עד להשגת התוצאות הרצויות. לבקר היו ביצועים טובים במיוחד במעקב אחר קו במסלולים ישרים, אך התמודד פחות טוב עם עקומות חדות. הבחירה במקדם Kp גבוה יחסית אפשרה תגובה מהירה לסטיות מהקו, אך הדבר הוביל לעתים לתנודות קטנות סביב הקו. הניסיון שלנו מצביע על כך שלמרות ההצלחה הכללית של הרובוט במילוי המשימות, יש צורך בשיפור האלגוריתמים כדי להתמודד טוב יותר עם תנאי שטח משתנים. שילוב של בקרת PID עם שיטות בקרה מתקדמות נוספות יכול לשפר את היכולת של הרובוט להתמודד עם מצבים מורכבים ולא צפויים במסלול.

כמו כן, הניסויים שבוצעו הראו כי הרובוט מסוגל להכנס למעקב אחר המסלול במהירות המתאימה לערך PWM של 3000, תוך יכולת להתיישר ולעקוב אחר מסלולים ישרים וברוב הפעמים במסלולים המאונכים לכיוון תנועתו. עם זאת, במסלולים מעוקלים הצלחת המעקב תלויה בזווית הכניסה, כאשר בזוויות מסוימות הרובוט מתקשה לשמור על המסלול. נמצא כי העלאת ערך ה- PWM משפרת את מהירות התנועה, אך פוגעת

ביכולת העלייה על מסלול והתיישרות עליו. לאור זאת, הוחלט להגדיר PWM של 3000 כברירת מחדל, מאחר שהוא מספק איזון מיטבי בין יציבות המעקב לבין מהירות הנסיעה.

אתגר נוסף היה דיוק הסיבוב והשפעתו על ביצועי החניה ברוורס. פונקציית turnByAngle שימשה לביצוע סיבובים בזוויות מוגדרות, אך הדיוק שלה הושפע ממספר גורמים כמו שינויים במתח הכניסה, תנאי המשטח, ומיקום החניה על מסלול ישר או מעוקל. כאשר החניה הייתה על מסלול מעוקל, דיוק הסיבוב נפגע ולעיתים הרובוט לא התמקם בצורה אופטימלית לביצוע החניה ברוורס. למרות זאת, ברוב המקרים הרובוט הצליח לחנות בין פסי החניה ולצאת בחזרה למסלול לאחר ההמתנה הנדרשת בהתאם לדרישות המשימה. למרות האתגרים בדיוק סיבוב הרובוט, המערכת הפגינה יכולת התאוששות טובה והצליחה להשלים את משימות החניה והיציאה, מה שמעיד על עמידות המערכת והיכולת שלה לתפקד גם בתנאים שאינם אידיאליים.

יתר על כן המערכת הצליחה לזהות מכשולים במרחק יעיל. במשימת הבונוס ברוב המקרים המערכת איפשרה לרובוט לשמור על מרחק בטוח באמצעות בקר PID. עם זאת, חיישן המרחק הראה מגבלות מסוימות - בטווחים קרובים מאוד הקריאות היו לא אמינות לעיתים, ובמקרים של התקרבות מהירה של מכשול, זמן התגובה של המערכת לא היה מספיק מהיר כדי להימנע מהתנגשות. מהתנסותנו עם מערכת זיהוי המכשולים למדנו שאפילו עם חיישן בודד ניתן להשיג יכולת סבירה של זיהוי והימנעות ממכשולים, אך חשוב לקחת בחשבון את מגבלות החיישן ולהבטיח זמן תגובה מספק כדי לאפשר לבקר לפעול ביעילות.

6. סיכום

בפרויקט זה פותחה מערכת רובוטית אוטונומית המונעת באמצעות זוג מנועי DC הנשלטים באמצעות מיקרו-בקר STM32 אשר מאפשרת מעקב אחר מסלול וביצוע חניה אוטונומית. הרובוט עושה שימוש במערך חיישני החזר אור לזיהוי קווי המסלול ובחיישן מרחק לאיתור מכשולים וזיהוי חניות פנויות. הניסויים שבוצעו הראו כי הרובוט מצליח לעקוב אחר קו מסלול במהירות אופטימלית של PWM = 3000, כאשר במסלולים ישרים ומאונכים לתנועתו הוא מציג יכולת עקיבה גבוהה. עם זאת, במסלולים מעוקלים ביצועיו משתנים בהתאם לזווית הכניסה למסלול. נמצא כי העלאת ערך ה- PWM משפרת את מהירות התנועה אך פוגעת ביכולת ההתיישרות על המסלול. במהלך הפרויקט אותגרנו בכיול פרמטרי ה- PID והתמודדות עם דיוק תמרוני הסיבוב של הרובוט, במיוחד בעת חניה ברוורס. כמו כן, נמצא כי חיישן המרחק מספק קריאות אמינות בטווחים מסוימים, אך במקרים של התקרבות מהירה של מכשול זמן התגובה של הרובוט אינו תמיד מספיק למניעת התנגשויות. בסיכומו של דבר, המערכת הצליחה לעמוד בדרישות שהוגדרו, תוך הפגנת יכולת מעקב אחר מסלול, זיהוי חניות וביצוע חניה מדויקת במרבית המקרים. עם זאת, קיימת אפשרות לשיפור ביצועי הרובוט על ידי שימוש בשיטות בקרה מתקדמות יותר, הוספת חיישנים לשיפור הזיהוי, ושיפור האלגוריתמים להתמודדות עם מסלולים מעוקלים ומכשולים בתנועה.

7. הצעות לשיפור

כדי להקל על השליטה במערכת, אפשר להוסיף חיישנים נוספים שיהפכו את המערכת לחכמה יותר ויאפשרו הבנה מעמיקה יותר של הדינמיקה של הרובוט. במצב הקיים הרובוט מבצע חניה בעזרת פקודות בחוג פתוח, אם היה לרובוט סרגל חיישנים נוסף מאחור היה ניתן לבצע את החניה בחוג סגור ובכך להפוך את תהליך ביצוע החניה למדויק יותר.

בנוסף, ניתן לבחון את החומרה הכללית של הרובוט ולשאוף להפוך אותה לאחידה יותר בין כלל הרובוטים במעבדה, באופן שהקוד שקבוצה כותבת יוכל לעבוד לא רק על הרובוט הספציפי שלה, אלא להיות מותאם פחות או יותר לרובוטים אחרים במעבדה. גישה זו תאפשר בדיקה מקיפה יותר של הפונקציונליות של הקוד ותוריד את התלות בזמינות של רובוט ספציפי.

8. הצהרה על שיתופי פעולה

בזיהוי החניה וביצוע הנסיעה ברוורס קיבלנו סיוע מקבוצה 3.

9. רשימת מקורות

1. אתר קורס המיקרו-מחשב במערכות מכניות במודל אוני' בן גוריון.

10. נספחים נוספים

10.1 פונקציית חישוב ערך התיקון לבקרת מסלול

```
//חיישנים תיקון ערך חישוב
int getCorrectionValue()
{
  int val=0;
  int sensor_map[8]={-2,-2,-1.5,-1,1,1.5,2,2};
  for (int i = 0; i < 8; i++)
  {
      if(m0.array_sensor[i]>SensorThreshold)
        val+=sensor_map[i];
  }
  return val;
```

איור 13 (נספח): פונקציית getCorrectionValue

פונקציית חישוב ערך התיקון לבקרת מסלול, המופעה באיור 13, אחראית על חישוב ערך התיקון הדרוש לשמירה על הרובוט במרכז הקו. הפונקציה מקבלת את קריאות שמונת חיישני שורת הקו, ומחשבת ערך תיקון כולל באמצעות מפת משקולות. לכל חיישן מוקצה ערך במערך sensor_map, כאשר החיישנים

השמאליים מקבלים ערכים שליליים והחיישנים הימניים מקבלים ערכים חיוביים. עבור כל חיישן שקריאתו גבוהה מסף מוגדר (SensorThreshold), הערך המתאים ממפת החיישנים מתווסף לערך התיקון הכולל. ערך שלילי מצביע על סטייה שמאלה, וערך חיובי מצביע על סטייה ימינה מהקו, וכך ניתן לחשב את התיקון הנדרש עבור המנועים.

10.2. פונקציית זיהוי קו חניה

```
//זיהוי קו חניה
char detectParkingLine() {
  // בודק אם יש קו חניה באמצעות חיישני שורת הקו הקיצוניים
  if ((m0.array_sensor[0] > SensorThreshold || m0.array_sensor[1] > SensorThreshold)
    && m0.array_sensor[2] > SensorThreshold && m0.array_sensor[3] > SensorThreshold)
      return 'l';
  if ((m0.array_sensor[6] > SensorThreshold || m0.array_sensor[7] > SensorThreshold)
    && m0.array_sensor[4] > SensorThreshold && m0.array_sensor[5] > SensorThreshold)
      return 'r';
  return 0;
```

איור 14 (נספח): פונקציית detectParkingLine

פונקציה זו אחראית על זיהוי קו החניה באמצעות חיישני שורת הקו של הרובוט. הפונקציה בודקת את קריאות החיישנים הקיצוניים יחד עם החיישנים הסמוכים אליהם כדי לקבוע אם קיים קו חניה ולאיזה צד הוא נמצא. אם החיישנים השמאליים הקיצוניים (0 או 1) יחד עם החיישנים 2 ו-3 מזהים קו (קריאות מעל הסף), הפונקציה מחזירה 'l' המציין שקו החניה נמצא בצד שמאל. באופן דומה, אם החיישנים הימניים הקיצוניים (6 או 7) יחד עם החיישנים 4 ו-5 מזהים קו, הפונקציה מחזירה 'r' המציין שקו החניה נמצא בצד ימין. אם לא זוהה קו חניה, הפונקציה מחזירה 0. זיהוי מדויק של קו החניה הוא קריטי לביצוע נכון של תמרון החניה של הרובוט.

10.3. פונקציית סיבוב הרובוט בזווית מדויקת

```
//מסויים לכיוון בזווית מסתובב
//direction= 'r'/'l', angle>0
void turnByAngle(char direction, int angle)
{
  //תקין קלט בדיקת
  if(angle<0 || (direction!='r' && direction!='l'))
    return;

  //כיוון הגדרות
  int dir=0;
  if(direction=='r')
    dir=1;
  else if(direction=='l')
    dir=-1;

  //מסתובב הרובוט בה הזווית את קובע הדיליי משך ,הנכון בכיוון למנועים פניה הוראת מתן
  Motor_Drive(MOTOR1, dir* basespeed);
  Motor_Drive(MOTOR2, dir* -basespeed);
  MyDelay(5750*angle/360.0); //when voltage is exactly 7.0
  Motor_Stop(MOTOR1);
  Motor_Stop(MOTOR2);
}
```

איור 15 (נספח): פונקציית turnByAngle

פונקציה זו מאפשרת לרובוט להסתובב בזווית מדויקת לכיוון מבוקש. הפונקציה מקבלת שני פרמטרים: כיוון הסיבוב 'r' לימין או 'l' לשמאל וגודל הזווית בדרגות. ראשית, הפונקציה מבצעת בדיקת תקינות של הפרמטרים שהתקבלו, ומוודאת שהזווית חיובית וכיוון הסיבוב תקין. לאחר מכן, בהתאם לכיוון שהתקבל, נקבע משתנה dir שמגדיר את כיוון הסיבוב (1 לימין, 1- לשמאל). הרובוט מסתובב במקום על ידי הפעלת המנועים בכיוונים מנוגדים, כאשר זמן ההשהיה $5750 \cdot angle/360$ מילישניות קובע את מידת הסיבוב שיבצע הרובוט. זמן ההשהיה כויל כך שהוא מתאים למתח של 7.0 וולט בדיוק. בסיום הסיבוב, שני המנועים נעצרים.