

# תרגיל 1 - רובוטים ניידים (עקיבה אחר קו - תנועה מבוקרת משוב, בקר PID, חישוב מסלול בעזרת אודומטריה)

תאריך: 2.12.25

קורס: רובוטים ניידים

מרצה: ד"ר יורם יקותיאל

מגישים:

בנימין רוטין | ת"ז: 211426598


גיא דנין | ת"ז: 205372105

דוד ושלר | ת"ז: 209736578

קוד בגיטהב:

[https://github.com/DavidWeschler/Mobile\\_Robots](https://github.com/DavidWeschler/Mobile_Robots)

תיקיה בדרייב שמכילה את כל המדיה והקוד:

 ex1\_robots\_media

## הקדמה

בתרגיל זה פיתחנו רובוט LEGO Mindstorms בעל הנעה דיפרנציאלית שמבצע עקיבה אחר קו כהה באמצעות חיישן אור יחיד. את האלגוריתמים והבקרה מימשנו בשפת NXC, ואת ניתוח הנתונים (אודומטריה והצגת מסלולים) ביצענו ב-Python. העבודה כללה שתי משימות עקיבה: אלגוריתם בסיסי ואלגוריתם מתקדם המבוסס על בקר PID, וכן חישוב מסלול מדויק בעזרת אודומטריה והצגתו - הן בעזרת plot כאשר החישוב בוצע "offline", והן על מסך הרובוט.

## משימת עקיבה ראשונה

אלגוריתם ראשון -

בשלב הראשוני התמקדנו בפיתוח אלגוריתם עקיבה פשוט שמבוסס על ערכי חיישן האור. חיישן האור מחזיר ערך מספרי בתחום שבין 0 לבין 100. ערכים נמוכים מעידים על פני שטח כהים כמו הקו השחור או הכחול, וערכים גבוהים מעידים על פני שטח בהירים כגון הלבן. הגדרנו ערך סף ראשוני של 50. כאשר ערך החיישן קטן או שווה ל-50 הרובוט נמצא על הקו הכהה. כאשר הערך גדול מ-50 הרובוט נמצא על האזור הבהיר וסטה מהמסלול.



כאשר ערך החיישן נמצא מתחת לערך הסף הרובוט ממשיך ישר. כאשר ערך החיישן חוצה את הסף כלפי מעלה הרובוט מפסיק להתקדם ישר ועובר למצב סריקה. מצב הסריקה כולל פניות לסירוגין ימינה ושמאלה תוך תנועה לאחור עד שהחיישן חוזר להציג ערך נמוך שמייצג שוב אזור כהה. תדירות הדגימה הראשונה הייתה כל 10 מילישניות. מהירות הנסיעה ההתחלתית הייתה 60.

ביצענו ניסויים גם במהירות גבוהה יותר של 70. בזמן זה עדיין השתמשנו בחיישן במצב דלוק. בתצורה הזאת התצפיות היו פחות יציבות והופיעו סטיות מהמסלול. הסיבה לכך הייתה שינוי גדול בערכי חיישן האור בין רגע לרגע, ככל הנראה בגלל השפעות תאורה חיצונית.

בין הניסויים בדקנו את ביצועי הרובוט כאשר החלון במעבדה פתוח וכאשר הוא סגור. כאשר החלון היה פתוח נכנסה תאורה חיצונית חזקה שהשפיעה מאוד על ערכי החיישן. שינויים קטנים באור נכנס גרמו לשינוי משמעותי בקריאת החיישן ולכן התקבלו תוצאות פחות עקביות. כאשר סגרנו את החלון ורמת האור בחדר התייצבה התוצאות היו יציבות בהרבה.

### ניסוי ראשון:

תוצאות הניסוי הראשוני עם חיישן דלוק, threshold שווה ל-50, דגימה כל 10 מילישניות ומהירות 60:

סיבוב נגד כיוון השעון: 29 שניות

סיבוב עם כיוון השעון: 25 שניות

תקופה זו התאפיינה בזמן ביצוע איטי יחסית ביחס לקבוצות אחרות.

### לצפייה בסרטון

במטרה לנסות לשפר את ביצועי הרובוט, עברנו למצב אור חיישן כבוי. כיבוי האור של החיישן גרם לכך שהקריאה שלו מבוססת יותר על אור חיצוני ופחות על התאורה הפנימית שהוא מקרין. לאחר שסגרנו את החלון, כך שהמעבדה מוארת באופן קבוע וללא שינויים חדים, הורדנו את ערך הסף ל-29. ערך זה נקבע לאחר בחינת ערכי הקריאה על הקו ומחוצה לו. במסגרת זו ערכי חיישן נמוכים מ-29 שייכים לאזור כהה, וערכים גבוהים מ-29 לאזור בהיר.

### ניסוי שני:

תוצאות הניסוי לאחר שינוי מצב החיישן לערך כבוי ושינוי ערך הסף ל-29:

סיבוב נגד כיוון השעון: 29 שניות

סיבוב עם כיוון השעון: 22 שניות

כאן כבר נצפה שיפור מסוים בזמני הסיבוב, בעיקר בכיוון הנגדי לשעון.

אבל - נתקלנו באי יציבות של הרובוט בעיקר בפניות החדות, שם הוא לפעמים סטה מהמסלול.

### לצפייה בסרטון

כדי לייצב את הרובוט ולמנוע סטיות, הדלקנו חזרה את חיישן האור. בשלב הזה הגדלנו את תדירות הדגימה של ערך החיישן מדגימה כל 10 מילישניות לדגימה כל 1 מילישנייה. שינוי זה שיפר את מהירות התגובה של האלגוריתם. ברגע ש-1ms מספיקים כדי לזהות עלייה בערך החיישן, הרובוט מצליח לתקן את המסלול מהר יותר ומבזבז פחות זמן באזור הבהיר.

### ניסוי שלישי:

תוצאות הניסוי לאחר העלאת תדירות הדגימה:

סיבוב נגד כיוון השעון: 24 שניות

סיבוב עם כיוון השעון: 20 שניות

שיפור זה היה משמעותי והפך את האלגוריתם למדויק ומהיר יותר.

### לצפייה בסרטון של נסיעה נגד כיוון השעון

### לצפייה בסרטון של נסיעה עם כיוון השעון

במהלך העבודה שיחקנו עם ערכי המהירויות הבאים: מהירות הנסיעה (speed) ומהירות הסריקה (searching\_speed). בניסויים אלו ראינו בבירור את ה-trade off בין מהירות לבין יציבות. כאשר העלינו את המהירות הרובוט אמנם השלים את ההקפה מהר יותר, אך הופיעו יותר מצבים של סטייה מהמסלול. מצד שני, בחירה במהירות נמוכה יותר שיפרה את היציבות אך האריכה את זמן ההקפה. שינוי במהירות החיפוש יצר השפעה ישירה על מידת

האגרסיביות של תיקוני המסלול בזמן שהחיישן יצא מהאזור הכהה. מהירויות גבוהות מדי גרמו לתנועות חדות ולפעמים לאובדן עקיבה, ומהירויות נמוכות מדי גרמו לרובוט לשהות זמן רב במצב סריקה.

ניסינו גם מספר שיטות לבחירת כיוון הסריקה בעת חריגה מהקו. ניסינו להתחיל תמיד ימינה. ניסינו להתחיל תמיד שמאלה. ניסינו להחליף כיוון בכל פעם ללא קשר לערך האחרון. בנוסף ניסינו לשמור את הכיוון האחרון שבו נמצאה הצלחה ולבסס עליו את הכיוון הבא. בסופו של דבר הגישה הטובה ביותר הייתה שינוי כיוון בכל חריגה. פעולה זו מנעה הצטברות של שגיאות לצד אחד ושיפרה את כיסוי אזור החיפוש.

**[קישור לקוד של אלגוריתם זה](#) (הערה - זהו לא הקוד הסופי של משימת העקיבה הראשונה)**

אלגוריתם שני וסופי -

לאחר שלא הצלחנו לרדת מזמן של 20 שניות לסיבוב באלגוריתם המקורי, עברנו לבחון אלגוריתם אחר שמיועד לעקיבה מהירה יותר על קו מבלי לבצע עצירות במהלך הנסיעה. מתוך אתר [purduesigbots](#) מצאנו אלגוריתם מסוג Bang Bang, והחלטנו ליישם גרסה שלו המבוססת על המשתנים המופיעים בקוד שלנו.

באלגוריתם Bang Bang אין שימוש בערך סף קבוע מראש. במקום זאת ערך הסף THRESHOLD מחושב בתוך התכנית כממוצע בין BLACK לבין WHITE. בקוד שלנו BLACK מוגדר כ-30 והוא מייצג את ערך חיישן האור באזור השחור, ו-WHITE מוגדר כ-60 והוא מייצג את ערך האור באזור הלבן. מכיוון ש-THRESHOLD מחושב כממוצע בין האזור השחור ללבן, מתקבל ערך סף של 45. האלגוריתם מנסה לשמור את ערך החיישן כמה שיותר קרוב לערך זה, מה שממקם את הרובוט על הקו הכחול.

לאחר חישוב הסף התחלנו בהרצות עם BASE\_SPEED מוגדר ל-90. מהירות הפנייה הוגדרה על פי אותו יחס שבו השתמשנו בקוד, כלומר BASE\_SPEED חלקי 2. המשמעות היא שבפנייה מהירות המנוע האיטי הייתה 45. בצורה זו האלגוריתם היה יציב והרובוט השלים סיבוב מלא בזמן של 13 שניות. לא היה הבדל בין נסיעה בכיוון השעון לבין נסיעה נגד כיוון השעון.

לאחר מכן ביצענו סדרת ניסויים כדי לשפר את זמן הסיבוב. בכל הניסויים שמרנו על עיקרון הפנייה מהקוד, כלומר שינוי במהירות אחד המנועים על פי יחס מוגדר של BASE\_SPEED לעומת מהירות מתוקנת. הניסויים שבוצעו היו:

#### **ניסוי ראשון:**

BASE\_SPEED מוגדר ל-100. מהירות הפנייה הוגדרה כ-100 חלקי 2, כלומר 50. בניסוי הזה הרובוט יצא מהמסלול ולא השלים סיבוב.

[לצפייה בסרטון](#)

#### **ניסוי שני:**

BASE\_SPEED מוגדר ל-80. מהירות הפנייה הוגדרה כ-80 חלקי 1.5, כלומר בערך 53. גם בניסוי זה הרובוט סטה מהמסלול.

### ניסוי שלישי:

BASE\_SPEED הוגדר ל-100. מהירות הפנייה הוגדרה כ-100 חלקי 2.5, כלומר 40. בניסוי זה זמן הסיבוב התקבל כ-15 שניות.

### ניסוי רביעי:

BASE\_SPEED הוגדר ל-100. מהירות הפנייה הוגדרה כ-100 חלקי 2.1, כלומר בערך 47.6. בצורה זו הרובוט השלים סיבוב בזמן של כ-9 שניות.

[לצפייה בסרטון נסיעה עם כיוון השעון](#)

[לצפייה בסרטון נסיעה נגד כיוון השעון](#)

מניתוח התוצאות אפשר לראות בבירור שקיים trade off בין יציבות לבין זמן הסיבוב. כאשר BASE\_SPEED היה גבוה מדי הרובוט התחיל לסטות מהקו לפני שהפיק תגובה מתאימה. מצד שני, כאשר מהירות הפנייה לא הייתה ביחס מדויק ל-BASE\_SPEED התיקון היה חד מדי או איטי מדי, וכתוצאה מכך הרובוט או יצא מהמסלול או איבד זמן. יחס המהירויות בין BASE\_SPEED לבין BASE\_SPEED חלקי מקדם הפנייה היה הגורם המרכזי שהשפיע על הצלחת הפנייה ועל זמן ההקפה.

בסופו של דבר השילוב המוצלח ביותר היה BASE\_SPEED שווה ל-100 ומהירות פנייה של בערך 47.6, והוא אפשר זמן הקפה של כ-9 שניות, שהוא הזמן הטוב ביותר שהשגנו עם אלגוריתם Bang Bang.

### [קישור לקוד הסופי של משימת העקיבה הראשונה](#)

[קישור לסרטון מלא של משימת העקיבה הראשונה - שני סיבובים ועצירה בסוף \(עם כיוון השעון\)](#)  
[קישור לסרטון מלא של משימת העקיבה הראשונה - שני סיבובים ועצירה בסוף \(נגד כיוון השעון\)](#)

## **משימת עקיבה שנייה**

במשימה זו השתמשנו בבקר PID כדי לשפר את איכות העקיבה ביחס לאלגוריתמים הפשוטים. המטרה הייתה להשיג תנועה רציפה, מהירה ויציבה יותר, תוך ביצוע תיקוני כיוון חלקים המבוססים על השגיאה בין ערך חישן האור לבין ערך היעד. ביצענו סדרת ניסויים לכיול מקדמי P ו-D עד למציאת שילוב אופטימלי.

### **ניסוי ראשון:**

בחרנו להתחיל עם ערך קטן יחסית של  $k_p$  (0.8) וערך גבוה יותר של  $k_d$  (1.2) כדי לבדוק תגובה רגישה יותר לתיקוני השגיאה המיידית ולהמעט את תגובת ה-P, מתוך תקווה שהתנהגות הרובוט תהיה יציבה ולא תגרום לרעידות יתר.  $k_p$  הוגדר ל-0.8 ו- $k_d$  הוגדר ל-1.2. התנהגות הרובוט הייתה לא טובה, מכיוון שהוא לא עקב אחרי הקו השחור. אמנם הרובוט עשה סיבוב כלשהו, אך לא היה טעם למדוד אותו מכיוון שהוא קיצר הרבה פינות. התוצאה הייתה תנועה עצבנית, והרובוט סטה מהמסלול הרבה.

[לצפייה בסרטון](#)

### **ניסוי שני:**

בחרנו להעלות את ערך  $k_p$  ל-1.5 ולהוריד את  $k_d$  ל-0.5 כדי להגביר את ההשפעה של תיקון הסטייה המיידית ולהפחית את השפעת ה-damping (המאופיין ב- $k_d$ ), במטרה לבדוק האם הרובוט יעקוב טוב יותר אחרי הקו אך בלי תגובות רכות מדי.  $k_p$  הוגדר ל-1.5 ו- $k_d$  הוגדר ל-0.5. בניסוי זה הרובוט הפגין תוצאות טובות כאשר נע נגד כיוון השעון - 9.18 שניות להשלמת סיבוב, אך כאשר נע עם כיוון השעון הרובוט ברח מהמסלול.

[לצפייה בסרטון נסיעה עם כיוון השעון](#)

[לצפייה בסרטון נסיעה נגד כיוון השעון](#)

### **ניסוי שלישי:**

בחרנו לשלב ערך ביניים של  $k_p$  (1.2) עם ערך גבוה יותר של  $k_d$  (1.8) במטרה להחזיר קצת יציבות לתגובות הרובוט ולמנוע סטיות גדולות על ידי הגדלת השפעת רכיב ה-D, בתקווה לשפר את היציבות לאורך המסלול.  $k_p$  הוגדר ל-1.2 ו- $k_d$  הוגדר ל-1.8. התוצאות כאן היו דומות לניסוי השני, עם זמן של 9.51 שניות נגד כיוון השעון ובריחה מהמסלול עם כיוון השעון.

### **ניסוי רביעי וסופי:**

לאחר שניסוינו כמה שילובים שונים, החלטנו לכייל את הבקר לערכי  $k_p$  ו- $k_d$  גבוהים יותר (2.0) כדי לקבל תגובה מהירה יותר ואגרסיבית יותר לתיקון הסטייה, אך גם מאוזנת מספיק כדי לשמור על יציבות בתנועת הרובוט לאורך המסלול הקצר. קבענו מהירות בסיסית של 90, וקבענו את ערכי ה-PID כך ש- $k_p$  יהיה שווה ל-2.0 ו- $k_d$  יהיה שווה

ל-2.0. ערכים אלה נמצאו כנקודת איזון יעילה במיוחד בין תגובה מהירה של הרובוט לבין יציבות בתיקון הסטייה ומהירות הרובוט הייתה כ-9 שניות לסיבוב. במהלך הנסיעה נצפתה התנהגות יציבה: הרובוט שמר על קו עקיבה רציף, ביצע תיקוני כיוון מהירים אך לא אגרסיביים, ולא נרשמו מצבים שבהם הוא סטה מהמסלול. התגובה של חיישן האור לערכי ה-PID הייתה חלקה ולא נצפו תנודות או רטט במסלול. ניסוי זה היה הניסוי המוצלח ביותר מתוך סדרת ניסויי ה-PID, והוא מהווה את נקודת הסיום בתהליך הכיול של הבקר עבור חלק זה של העבודה.

[לצפייה בסרטון נסיעה עם כיוון השעון](#)

[לצפייה בסרטון נסיעה נגד כיוון השעון](#)

### **ניתוח תוצאות ביחס למשימת העקיבה הראשונה:**

ה-PID שיפר משמעותית את הביצועים ביחס לאלגוריתמים של משימה 1. אמנם זמן ההקפה היה דומה לאלגוריתם של משימה 1, כ-9 שניות, אך העקיבה הפכה יציבה יותר וללא איבודי קו. בנוסף, הפערים בין הכיוונים כמעט נעלמו, והרובוט הגיב טוב יותר לשינויים בתאורה. בסך הכול, ה-PID סיפק שילוב טוב יותר של מהירות, דיוק ויציבות.

[הקוד הסופי של משימת העקיבה השנייה PID](#)

[קישור לסרטון מלא של משימת העקיבה השנייה - שני סיבובים ועצירה בסוף \(עם כיוון השעון\)](#)

[קישור לסרטון מלא של משימת העקיבה השנייה - שני סיבובים ועצירה בסוף \(נגד כיוון השעון\)](#)

## משימת חישוב המסלול על ידי אודומטריה

[קישור לקוד של אלגוריתם PID שמחשב את המסלול בעזרת אודומטריה ושומר לקובץ](#)  
[קישור לקוד של אלגוריתם PID שמחשב את המסלול בעזרת אודומטריה ומציג את המסלול על](#)

[המסך](#)

[קישור לקוד פייתון המייצר plot של המסלול על סמך קובץ הפלט](#)

השיטה בה השתמשנו לחישוב מיקום הרובוט מבוססת על קינמטיקה של הינע דיפרנציאלי. בשיטה זו, אנו מניחים שהרובוט נע על משטח דו-ממדי ושולט על כיוון התנועה והמהירות אך ורק באמצעות שינוי המהירות היחסית בין שני הגלגלים המקבילים.

החישוב:

האלגוריתם מבצע סכימה של תנועת הגלגלים בכל מחזור ריצה של הבקר. הנחת היסוד היא שבפרקי זמן קצרים מאוד  $dt \rightarrow 0$ , הרובוט נע בקשת מעגלית או בקו ישר.

החישוב מתבצע בשלושה שלבים עבור כל מחזור לולאה:

1. חישוב המרחק שעבר מרכז הרובוט ( $\Delta Distance$ ):  
אנו קוראים את המידע מהמקודדים (החיישנים של המנוע) ומחשבים ממוצע של המרחק שעבר הגלגל השמאלי והימני. זהו המרחק שעבר מרכז הציר של הרובוט.  
$$\Delta D = (\Delta L + \Delta R)/2$$
2. חישוב השינוי בכיוון ( $\Delta\theta$ ):  
ההפרש בין המרחק שעבר גלגל ימין לגלגל שמאל, כשהוא מחולק ברוחב הרובוט ( $TrackWidth$ ), נותן לנו את הזווית (ברדיאנים) שהרובוט הסתובב באותו מחזור.  
$$\Delta\theta = (\Delta L - \Delta R)/TrackWidth$$
3. עדכון המיקום החדש (אינטגרציה):  
אנו משתמשים בטריגונומטריה כדי להטיל את המרחק ( $\Delta D$ ) על מערכת הצירים הגלובלית ( $X, Y$ ), בהתבסס על הזווית הנוכחית, ומוסיפים זאת למיקום הקודם:  
$$\theta_{new} = \theta_{old} + \Delta\theta$$
$$X_{new} = X_{old} + \Delta D * \cos(\theta_{old} + \Delta\theta/2)$$
$$Y_{new} = Y_{old} + \Delta D * \sin(\theta_{old} + \Delta\theta/2)$$

Trade-offs בין מהירות לדיוק:

קיים מתח בין הרצון לנוע מהר לבין הצורך בדיוק באודומטריה:

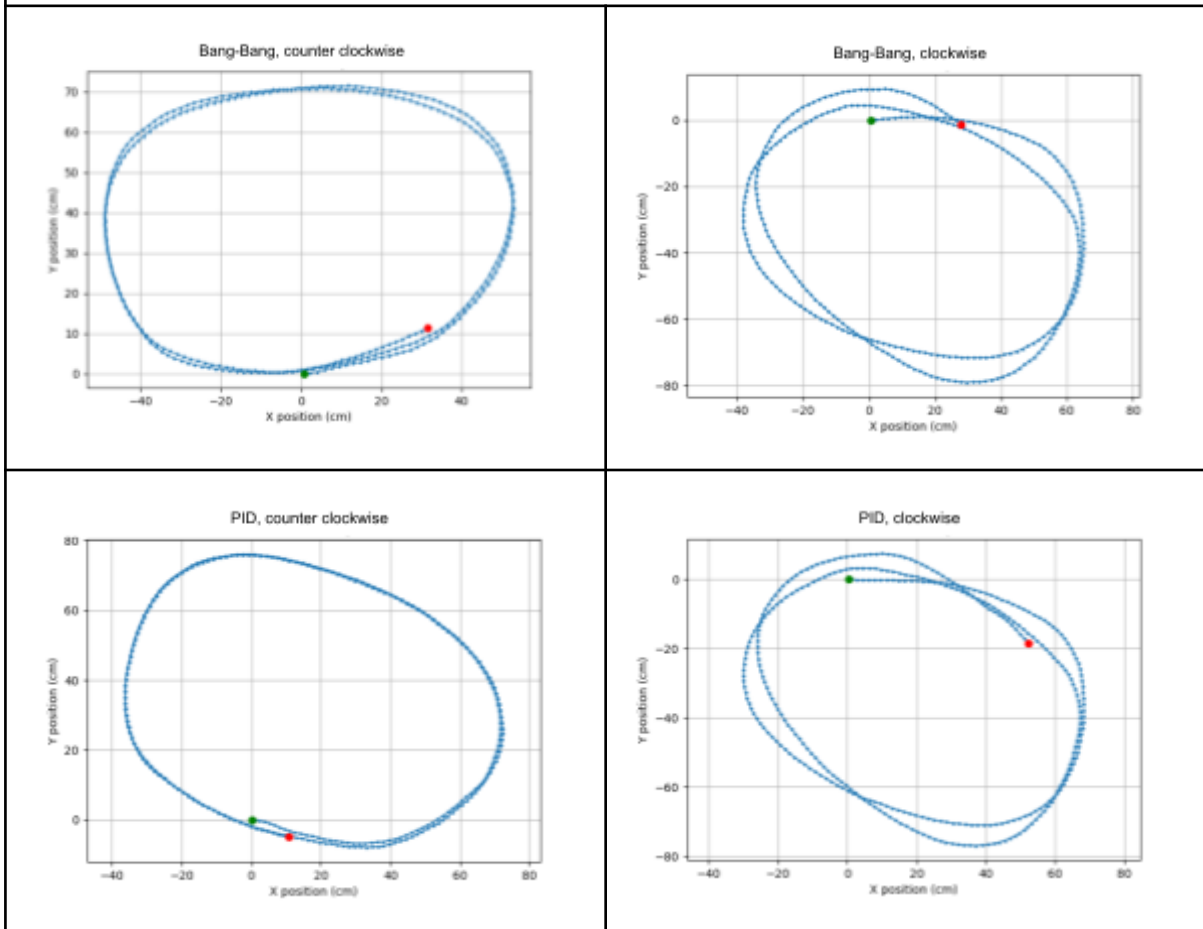
1. תדירות דגימה מול מהירות נסיעה: ככל שהרובוט נוסע מהר יותר, הוא עובר מרחק רב יותר בין כל דגימה של המעבד. בקשתות חדות, דגימה איטית מדי תגרום לחישוב "לחתוך" את הסיבוב ולהפוך קשת חלקה לקו ישר (Aliasing), מה שיוצר שגיאה מצטברת במיקום.
2. החלקה: בתאוצות גבוהות או בלימות פתאומיות (האופייניות לאלגוריתם Bang-Bang), הגלגלים נוטים להחליק על המשטח. המקודדים "חושבים" שהרובוט נע, אך בפועל הוא מחליק במקום. נסיעה איטית יותר ממזערת החלקה ומשפרת את הדיוק הפיזי.

כפי שניתן לראות בגרפים הבאים (בעמוד הבא), בשני האלגוריתמים הרובוט נוסע בצורה חלקה יותר נגד כיוון השעון מאשר עם כיוון השעון.



הסיבה לכך קשורה להבדל בהיקפי הגלגלים: גלגל שמאל קטן יותר (היקף 17.5 ס"מ) וגלגל ימין גדול יותר (היקף 17.7 ס"מ). מאחר שגלגל ימין מתקדם מעט יותר בכל סיבוב, נוצר סטייה טבעית שתורמת לכך שהתנועה נגד כיוון השעון מאוזנת יותר עם הפער הזה, ולכן דורשת פחות תיקוני מסלול. לעומת זאת, כאשר הרובוט נוסע עם כיוון השעון, אותו פער בין הגלגלים פועל נגד כיוון הסיבוב, מה שמגדיל את הצורך בתיקונים תכופים יותר. לכן מתקבל מסלול פחות חלק ויותר "מזוגזג".

### הצגת המסלול בעזרת חישוב אודומטריה



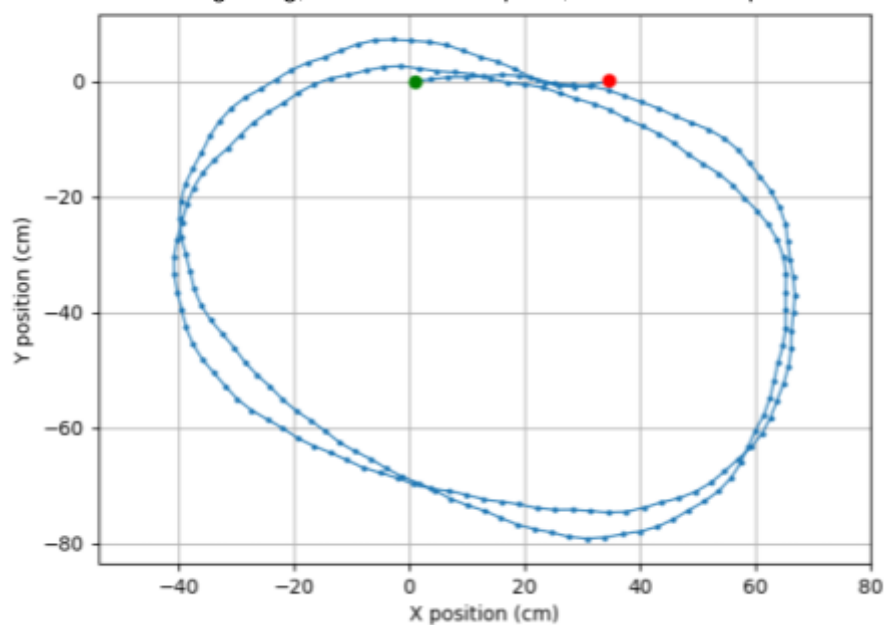
הצגת המסלול על גבי המסך,  
2 סיבובים באלגוריתם PID נגד כיוון השעון



הצגת המסלול על גבי המסך,  
2 סיבובים באלגוריתם PID עם כיוון השעון



Bang-Bang, with increased speed, trade-off example



## ביבליוגרפיה

אודומטריה וחישוב מסלול:

1. Going from Odometry to Position in Two Dimensions .

<http://credentiarity2.blogspot.co.il/2010/06/going-from-odometry-to-position-in-two.html>

2. Odometry .

ויקיפדיה.

<http://en.wikipedia.org/wiki/Odometry>

3. Rossum, R. Differential Steering for Mobile Robots .

<http://rosum.sourceforge.net/papers/DiffSteer/>

4. Calculate position of differential drive robot .

Robotics StackExchange

<http://robotics.stackexchange.com/questions/1653/calculate-position-of-differential-driverobot>

בקר PID:

5. PID Controller .

ויקיפדיה.

[http://en.wikipedia.org/wiki/PID\\_controller](http://en.wikipedia.org/wiki/PID_controller)

6. PID Controller for LEGO Mindstorms Robots .

Inpharmix

[http://www.inpharmix.com/jps/PID\\_Controller\\_For\\_Lego\\_Mindstorms\\_Robots.html](http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html)

7. Using a PID Controller .

Seattle Robotics Society

[http://www.seattlerobotics.org/encoder/200108/using\\_a\\_pid.html](http://www.seattlerobotics.org/encoder/200108/using_a_pid.html)

דוקומנטציית שפת NXC:

8. NXC API Documentation .

<https://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html>

מאמרים מקצועיים נוספים:

9. Nahmed, N. Wheel Odometry Model for Differential Drive Robotics .

Medium

<https://medium.com/@nahmed3536/wheel-odometry-model-for-differential-drive-robotics-91b85a012299>

שימוש בכלים נוספים:

10. שימוש במערכת Gemini לתיקון שגיאות תחביר (syntax) בקוד .

אלגוריתם למשימת העקיבה הראשונה:

11. Purdue SIGBots. (n.d.). *Bang-bang control*. Retrieved November 24, 2025, from .

<https://wiki.purduesigbots.com/software/control-algorithms/bang-bang>