



ex1.pdf

PDF

תבצע בשבילי 3 משימות:

1) – תסכם לי את החומר מהקישורים הבאים

<http://credentiaity2.blogspot.co.il/2010/06/going-from-odometry-to-position-in-two.html>

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

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

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

2) – תסכם לי את החומר מהקישורים הבאים

PID בקר

http://en.wikipedia.org/wiki/PID_controller

http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html

http://www.seattlerobotics.org/encoder/200108/using_a_pid.html

מצורף לכאן (כ) תקרא את התרגיל שלי בקורס רובוטים ניידים (3) מסודר ומפורט של כל מה שאני roadmap ותבנה לי, (ex1.pdf), צריך לעשות לרגיל, שלב אחרי שלב, כולל הכל

– התחלתי כבר את התרגיל וזה הקוד שלי בינתיים

```
#define THRESHOLD 50
```

```
#define LEFT OUT_C
```

```
#define RIGHT OUT_A
```

```
#define BOTH OUT_AC
```

```
int SPEED = 60;
```

```
task main()
```

```
{
```

```
    // Set light sensor in active mode
```

```
    SetSensorLight(IN_3, true);
```

```
    // Assume initial rotation direction: true = clockwise
```

```
    bool dirCW = true;
```

```
    // Start moving forward
```

```
    OnFwd(BOTH, SPEED);
```

```

int dir = -1;

while (true)
{
    int lightValue = Sensor(IN_3);

    // Display light value on LCD
    ClearScreen();
    TextOut(0, LCD_LINE1, "Light:");
    NumOut(50, LCD_LINE1, lightValue);
    Wait(10);

    // If light exceeds threshold, search for dark area
    while (lightValue > THRESHOLD)
    {
        int searchTime = 100; // initial sweep duration

        // Sweep until dark area found
        while (lightValue > THRESHOLD)
        {
            OnFwd(RIGHT, (SPEED / 2) * dir * -1);
            OnFwd(LEFT, (SPEED / 2) * dir);

            Wait(searchTime);
            Off(BOTH); // stop to read sensor
            lightValue = Sensor(IN_3);

            if (lightValue <= THRESHOLD) {
                OnFwd(BOTH, SPEED); // dark area found,
move forward
                break;
            }

            // Switch sweep direction
            dir = -dir;

            // Gradually increase sweep range
            searchTime += 50;
        }
    }
}

```

```
// Optional: update rotation direction based on
motion pattern
// For example, if LEFT motor is always faster than
RIGHT, it's CW
// Here we just leave it static unless you implement
an odometry check
}
}
```

מצוין 👍

קראתי את קובץ התרגיל שלך **ex1.pdf** וגם ראיתי את הקוד שכתבת.
הבקשה שלך כוללת שלוש משימות, אז הנה התשובות מסודרות לפי שלבים:

משימה 1 - סיכום החומר על אודומטריה 🧩

1 מה זה אודומטריה

אודומטריה (Odometry) היא שיטה להערכת המיקום והכיוון של הרובוט על סמך מדידת סיבובי הגלגלים.

השיטה מבוססת על חישוב ההתקדמות של כל גלגל, והמרה שלה לשינוי במיקום ובזווית של הרובוט.

2 עקרון הפעולה

- שני הגלגלים (שמאל וימין) מסתובבים במהירות שונה → נוצר רדיוס סיבוב.
- לפי המרחק שכל גלגל עובר, אפשר לחשב את Δx , Δy ו- $\Delta \theta$ (שינוי במיקום ובכיוון).

נוסחאות עיקריות (לדיפרנציאל דרייב):

אם:

- R = רדיוס גלגל
- L = מרחק בין הגלגלים
- $\Delta \phi_L, \Delta \phi_R$ = שינוי בזוויות הסיבוב של הגלגלים

אז:

$$L \Delta \phi \cdot R = L \Delta s$$

$$R \Delta \phi \cdot R = R \Delta s$$

ההתקדמות הממוצעת:

$$\frac{L \Delta s_R + \Delta s}{2} = \Delta s$$

$$\frac{L\Delta s_R - \Delta s}{L} = \Delta\theta$$

והמיקום החדש:

$$\left(\frac{\Delta\theta}{2} + \cos(\theta) \cdot \Delta s + \text{old}X\right) = \text{new}X$$

$$\left(\frac{\Delta\theta}{2} + \sin(\theta) \cdot \Delta s + \text{old}Y\right) = \text{new}Y$$

$$\Delta\theta + \text{old}\theta = \text{new}\theta$$

3 שגיאות באודומטריה

- הצטברות שגיאות לאורך זמן (drift).
- החלקת גלגלים, סטיות קטנות בקריאה, רדיוס שונה בגלגלים.
- תיקון אפשרי ע"י שילוב עם חיישנים נוספים (IMU, מצלמה וכו').

משימה 2 - סיכום על בקר PID

1 עקרון הפעולה

בקר (Proportional-Integral-Derivative) **PID** הוא מנגנון שמנסה לשמור משתנה של מערכת (למשל - מיקום על הקו) קרוב לערך יעד (setpoint).

$$\frac{de(t)}{dt}K_d + K_i \int e(t)dt + K_p e(t) = u(t)$$

- **P (Proportional)**: תיקון לפי גודל השגיאה.
- **I (Integral)**: תיקון לפי הצטברות השגיאה (מונע bias).
- **D (Derivative)**: תיקון לפי קצב שינוי השגיאה (מרסן תנודות).

2 כוונן (Tuning)

יש לקבוע את שלושת המקדמים K_p, K_i, K_d :

- K_p גבוה מדי → תנודות.
- K_i גבוה מדי → חוסר יציבות.
- K_d גבוה מדי → תגובה איטית.

שיטה מקובלת: Ziegler-Nichols או ניסוי וטעייה.

3 שימוש בעקיבה אחרי קו

- $e(t)$ = ההבדל בין ערך החיישן הנוכחי לבין ערך המטרה (גוויי אפור על הקו).
- $u(t) \rightarrow$ תיקון מהירות בין שני הגלגלים.

דוגמה:

C

```
error = target - light;
integral += error;
derivative = error - lastError;
correction = Kp*error + Ki*integral + Kd*derivative;
```

```
OnFwd(LEFT, baseSpeed + correction);
OnFwd(RIGHT, baseSpeed - correction);
```

משימה 3 – Roadmap מפורט לתרגיל שלך

שלב 1 – עקיבה בסיסית אחר קו (מה שכבר התחלת)

מטרה: הרובוט ינוע על הקו עם חיישן אור יחיד.

✓ מה לעשות:

1. לכייל את חיישן הבהירות \rightarrow לקבוע THRESHOLD בין כהה לבהיר.
2. להניע את שני הגלגלים קדימה (OnFwd(BOTH, SPEED)).
3. אם חיישן רואה בהיר \rightarrow לפנות לכיוון הקו (כמו בקוד שלך).
4. לשמור על עקיבה יציבה במשך שני סיבובים מלאים.

בדיקה: במעבדה – שני סיבובים מלאים (CW ו-CCW).

שלב 2 – שיפור עקיבה עם בקר PID

מטרה: להחליף את העקיבה הפשוטה בבקר PID.

✓ צעדים:

1. למדוד ערך יעד – צבע האפור של הקו.
2. לחשב בכל לולאה:
 - $error = target - sensorValue$
 - $correction = Kperror + Kiintegral + Kd*derivative$
3. לעדכן מהירות גלגלים בהתאם.

4. לנסות ערכי K_p , K_i , K_d שונים ולתעד זמני ביצוע.

בדיקה: אותו מסלול בדיוק → לבדוק אם PID מהיר ויציב יותר.

שלב 3 - חישוב מסלול (אודומטריה)

מטרה: לחשב את מיקום הרובוט לאורך המשימה.

✓ צעדים:

1. למדוד את מספר הסיבובים (encoder ticks) של כל גלגל.
 2. בכל לולאה לחשב:
 - $\Delta s_L, \Delta s_R$
 - $\Delta \theta = (\Delta s_R - \Delta s_L) / L$
 - $\Delta x, \Delta y$ לפי הנוסחאות.
 3. לשמור כל $\Delta x, \Delta y, \theta$ לקובץ (או להדפיס למסך).
 4. לבצע ניסוי מבוקר במסלול פשוט לבדוק אם אורך המסלול תואם למדידה.
- בדיקה: פלט של 4 מסלולים (שני כיוונים × שתי גרסאות).
-

שלב 4 - ניתוח והשוואה

✓ ניתוחים להגיש:




1. טבלת זמנים: כמה זמן לקח כל גרסה להשלים 2 סיבובים.
 2. גרף מסלול מהנתונים של האודומטריה של (MATLAB / Python).
 3. השוואה בין גרסה רגילה ו-PID.
 4. הסברים על בחירת משתנים, כיול חיישן, trade-offs.
-

שלב 5 - הגשה

יש להגיש 📦

- קוד סופי לכל שלב (NXC/Lejos/Java וכו').
 - קבצי נתוני מיקום (CSV או דומה).
 - תיעוד כולל הסבר, גרפים, תמונות של הרובוט.
 - סרטוני בדיקות במעבדה (CW, CCW).
 - לציין שימוש בעזרת AI (כמוני 😊).
-

אם תרצה, אני יכול להוסיף גם:

- גרף לדוגמה של מסלול אודומטריה. 
- אלגוריתם PID מלא עם קוד NXC מוכן. 
- תבנית דוקומנטציה מוכנה להגשה (עם פרקים קבועים לפי סעיפי התרגיל). 

רוצה שאכין לך את שלושת אלה?

תוסיף  אלגוריתם PID מלא עם קוד NXC מוכן.

מצוין — הנה אלגוריתם **PID מלא ב-NXC** מותאם לקוד שלך, עם הערות והסברים איך לכתיל ולשפר. הקוד משתמש במבני ההגדרות שכבר העלית (`LEFT` , `RIGHT` , `BOTH` , `IN_3`) ועמיד לתנאי זמן לולאה קבוע (sample time). העיקר — תשנה את `TARGET` , `Kp` , `Ki` , `Kd` לפי הכיול שתבצע במעבדה.

העיקר: הקוד מפעיל PID על בסיס ערך חיישן הבהירות (יחיד), מחשב תיקון ומשנה את מהירויות הגלגלים כדי לשמור על הקו.

```
// PID line follower (NXC)
// מתאים להגדרות שלך:
// #define LEFT OUT_C
// #define RIGHT OUT_A
// #define BOTH OUT_AC
#define THRESHOLD 50
#define LEFT OUT_C
#define RIGHT OUT_A
#define BOTH OUT_AC

// --- Config ---
int BASE_SPEED = 50;    // מהירות בסיס קדימה (100..0) - ניתן להגביר
int SAMPLE_TIME_MS = 50; // זמן לולאת בקר (בחייב לשמור קבוע)
int SENSOR_PORT = IN_3; // חיישן אור

// PID gains - התחילו בערכים אלו ואז כוילו
float Kp = 1.2;
float Ki = 0.01;
```

```
float Kd = 0.25;
```

```
// ערך חיישן שמייצג את מרכז הקו - (target) מטרה
```

```
// לכייל במקום: למדוד ערך בתוך הקו (או ממוצע בין ערכים כהה ובהיר) *
```

```
int TARGET = 35; // צריך לכייל בהתאם למסלול/חיישן
```

```
// שיא/מקס למשתנים
```

```
float INTEGRAL_MAX = 2000.0;
```

```
float INTEGRAL_MIN = -2000.0;
```

```
int MOTOR_MAX = 100;
```

```
int MOTOR_MIN = -100;
```

```
// ---- משתנים פנימיים ----
```

```
float integral = 0.0;
```

```
float lastError = 0.0;
```

```
bool firstLoop = true;
```

```
task main()
```

```
{
```

```
    // ידוא מצב חיישן פעיל
```

```
    SetSensorLight(SENSOR_PORT, true);
```

```
    // אופציונלי: תצוגה התחלתית
```

```
    ClearScreen();
```

```
    TextOut(0, LCD_LINE1, "PID line follower");
```

```
    TextOut(0, LCD_LINE2, "Calibrate TARGET");
```

```
    Wait(1000);
```

```
    // איפוס מצב
```

```
    integral = 0.0;
```

```
    lastError = 0.0;
```

```
    firstLoop = true;
```

```
    // לולאת בקר עיקרית
```

```
    while(true)
```

```
    {
```

```
        int light = Sensor(SENSOR_PORT);    // קריאת חיישן
```

```
        float error = (float)TARGET - (float)light; // שגיאה: חיובי = רובוט שמאלה מהיעד (תלוי בכיול) //
```

```
        // anti-windup אינטגרל עם
```

```
        integral += error * (SAMPLE_TIME_MS / 1000.0);
```



```

if (integral > INTEGRAL_MAX) integral = INTEGRAL_MAX;
if (integral < INTEGRAL_MIN) integral = INTEGRAL_MIN;

// נגזרת
float derivative;
if (firstLoop) {
    derivative = 0;
    firstLoop = false;
} else {
    derivative = (error - lastError) / (SAMPLE_TIME_MS / 1000.0);
}

// PID output (correction)
float correction = Kp * error + Ki * integral + Kd * derivative;

// חישוב מהירויות גלגלים (base +/- correction)
// נחלק כך שמתקן מתאים לפניות - correction בתלות בסימן
int leftSpeed = (int) (BASE_SPEED + correction);
int rightSpeed = (int) (BASE_SPEED - correction);

// להגבלה בטווח חוקי
if (leftSpeed > MOTOR_MAX) leftSpeed = MOTOR_MAX;
if (leftSpeed < MOTOR_MIN) leftSpeed = MOTOR_MIN;
if (rightSpeed > MOTOR_MAX) rightSpeed = MOTOR_MAX;
if (rightSpeed < MOTOR_MIN) rightSpeed = MOTOR_MIN;

// שליחת פקודות למנועים
OnFwd(LEFT, leftSpeed);
OnFwd(RIGHT, rightSpeed);

// שמירת ערכים לזלזלה הבאה
lastError = error;

// לשם דיבאג/כיול - LCD תצוגה על
ClearScreen();
TextOut(0, LCD_LINE1, "Light:");
NumOut(50, LCD_LINE1, light);
TextOut(0, LCD_LINE2, "Err:");
NumOut(36, LCD_LINE2, (int)error);
TextOut(0, LCD_LINE3, "Corr:");
NumOut(45, LCD_LINE3, (int)correction);
TextOut(0, LCD_LINE4, "L R:");

```

```

NumOut(30, LCD_LINE4, leftSpeed);
NumOut(60, LCD_LINE4, rightSpeed);

Wait(SAMPLE_TIME_MS);
}
}

```

הערות מעשיות לכיול ושיפור

1. כיול TARGET

- הנח את החיישן מעל הקו וכתוב על LCD את ערך `Sensor(IN_3)` (או השתמש בקוד למעלה בצורתו) — קבע את ה-TARGET לערך המייצג את מרכז הקו (יכול להיות ממוצע בין ערכים על הקו ובחוץ).

2. סרגל התחלה של מקדמים (Tune)

- התחל ב- $K_i = 0$ ו- $K_d = 0$. הגדל K_p עד שהרובוט מתחיל להתנוודד מעט.
- הכנס K_d כדי לשכך תנודות.
- הכנס K_i במינונים כדי להסיר שגיאת סטטית (אם הרובוט נוטה לצד אחד באופן קבוע).

3. SAMPLE_TIME_MS

- מומלץ 30–100 ms. אם תקטין את ה-sample time, כייל שוב את המקדמים (כי הם תלויים ב-dt).

4. Anti-windup

- קלענו אינטגרל עם גבולות (`INTEGRAL_MAX/MIN`) כדי למנוע הצטברות גדולה. אפשר להרחיב/להקטין לפי הצורך.

5. כיוון השגיאה

- אם התיקון פועל בכיוון הפוך (רובוט פונה בצד הלא נכון), החלף את סמן החישוב של `error` (למשל `error = light - TARGET`).

6. מהירות בסיס (BASE_SPEED)

- העלה לאט כדי לראות תופעות החלקה/איבוד אחיזה. מהירות גבוהה מגדילה את שגיאות האודומטריה אך מקטינה את זמן המשימה.

7. בדיקות

- בצע בדיקות CW ו-CCW, תעד זמני השלמה וערכי יציבות.

אם תרצה שאעשה הלאה

- להמיר את הקוד כך שירשום ל-CSV (לא בטוח אם ה-NXC שלך תומך בכתיבה לקובץ — אפשר להוסיף אם תרצה ואני אבדוק גרסה עם `FileWrite`).

- לשלב אודומטריה (להוסיף קריאות encoder ולשמור $X/Y/\theta$ בזמן אמת).
- להכין סקריפט Python שיקרא לוג מרובוט/קובץ וייצר גרף מסלול וניתוח ביצועים.

רוצה שאוסיף ממש גם:

1. גרסת קוד עם רישום ל-CSV (אם ה-NXC שלך תומך ב-FileWrite) — או
 2. הרחבה שמישלת אודומטריה (אם יש לך אופציונלי encoders) — או
 3. עזרה בכיול ערכי $k_p/k_i/k_d$ לפי תוצאות ניסוי שתריץ?
- איזה מהם מעדיף — להכין עכשיו?

קישור לשיחה זו -

<https://chatgpt.com/share/6915ed7e-9924-800e-8822-81349ae3d037>