

סיכום אודומטריה ורובוטיקה



ex1.pdf

PDF

תבצע בשבייל 3 שימושות:

- תסכם לי את החומר מהקישורים הבאים (1)

<http://credentiality2.blogspot.co.il/2010/06/going-from-odometry-to-position-in-two.html>

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

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

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

- תסכם לי את החומר מהקישורים הבאים (2)

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

מצורך לכך (כ) תקרא את התרגיל שלו בקורס רובוטים נידים (3)
מסודר ומפורט של כל מה שאני roadmap ותבנה לי, ex1.pdf,

צריך לעשות לרגל, שלב אחרי שלב, כולל הכל

- התחלתי כבר את התרגיל זהה הקוד שלו בינתים

```
#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 

קרأتם את קובץ התרגיל של **ex1.pdf** וגם ראתם את הקוד שכתבתם.
הבקשה שלכם כוללת שלוש משלימות, אז הנה התשיבות 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 התקדמות של כל גלגל, והמרת אלה לשינוי במיקום ובזווית של הרובוט.

2 עקרון הפעולה

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

נוסחאות עיקריות (לדיפרנציאל דרייב):

אם:

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

אז:

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

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

ההתקדמות המומוצעת:

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

השני בדואית:

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

וالمיקום החדש:

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

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

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

3 שגיאות באודומטריה

- הצבירות שגיאות לאורך זמן (drift).
- חילקת גלגלים, סטיות קטנות בקריאה, רדיוס שונה בגלגליים.
- תיקון אפשרי ע"י שילוב עם חיישנים נוספים (UME, מצלמה וכו').

משימה 2 – סיכום על בקר PID

1 עקרון הפעולה

בקר (PID) הוא מגנון שמנסה לשמר משתנה של מערכת (למשל – מיקום על הקו) קרוב לערך יעד (setpoint).

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

- P: תיקון לפי גודל השגיאה.
- I: תיקון לפי הצבירות השגיאה (מנוע bias).
- D: תיקון לפיקצב שינוי השגיאה (מרסן תנודות).

2 כוונן (Tuning)

יש לקבוע את שלושת המקדים K_p, K_i, K_d :

- K_p גבוהה מדי \rightarrow תנודות.
- K_i גבוהה מדי \rightarrow חוסר יציבות.
- K_d גבוהה מדי \rightarrow תגובה איטית.

שיטה מקובלת: Ziegler–Nichols או ניסוי וטעייה.

3 שימוש בעקביבה אחרי קנו

- $e(t) =$ ההבדל בין ערך החישון הנוכחי לבין ערך המטרה (גונו אפור על הcano).
- $\dot{e}(t) \rightarrow$ תיקון מהירות בין שני הגלגלים.

דוגמאות:

```
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 אחרoko (מה שכבר התחילו)

מטרה: הרובוט ינוע על הcano עם חישון אוור ייחיד.

 מה לעשות:

1. לכיד את חישון הבוחרות → לקבוע THRESHOLD בין כהה לבhair.
2. להניע את שני הגלגלים קדימה (SPEED). OnFwd(BOTH,
3. אם חישון רואה בהair → לפנות לכיוון הcano (כמו בקוד שלך).
4. לשמר 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 הcano.
2. לחשב בכל לולאה:
 - $error = target - sensorValue$
 - $correction = Kp*error + Ki*integral + Kd*derivative$
3. לעדכן מהירות גלגלים בהתאם.

4. לנשות ערכי K_d , K_p שונים ולתעד זמני ביצוע.

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

שלב 3 – חישוב מסלול (אודומטריה)

מטרה: לחשב את מיקום הרובוט לאורך המשימה.

צעדים:

1. למדוד את מספר הטיובים (encoder ticks) של כל גלגל.

2. בכלי לוויה לחשב:

- $\Delta s_L, \Delta s_R$
- $\Delta\theta = (\Delta s_R - \Delta s_L)/L$
- y, x לפי הנוסחאות.

3. לשמר כל $\theta, \Delta y, \Delta x$ 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 לוקח כל גרסה להשלים 2 סיבובים.

2. גרף מסלול מהנתונים של האודומטריה (MATLAB / Python).

3. השוואה בין גרסה רגילה ו-PID.

4. הסברים על בחירת משתנים, כiol חיישן, trade-offs.

שלב 5 – הגשה

יש להגיש:

• קוד סופי לכל שלב (Java/Lejos/NXC וכו').

• קבצי נתונים מיקום CSV או דומה.

• תיעוד כולל הסבר, גרפים, תמונות של הרובוט.

• סרטוני בדיקות במעבדה (CW, CCW).

• ליצין שימוש בעזרת AI (כמוני 😊).

אם תרצה, אני יכול להוסיף גם:

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

רוצה שacky ליר את שלושת אלה?

תוסיפ אלגוריתם PID מלא עם קוד NX מוכן.

מצוין — הנה אלגוריתם **PID מלא ב-NXC** מותאם לקוד שלך, עם הערות והסבירים איך לכיל ולשפר. הקוד משתמש במבנה ההגדרות שכבר העלית (`3_IN_3`, `LEFT`, `RIGHT`, `BOTH`, `TARGET`, `Kp`, `Ki`, `Kd`) ועמיד לתנאי זמן לולאה קבוע (sample time). העיקר — תנסה את `TARGET` לפי הcoil שתבצע במעבדה.

העיקר: הקוד מפעיל PID על בסיס ערך חישון הבבירות (יחיד), מחשב תיקון ומפנה את מהירות הגלגלים כדי לשמר 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 (0..100) - ניתן להגבר //
int SAMPLE_TIME_MS = 50;   // זמן לולאת בקר (בחיוב לשמור קבוע) //
int SENSOR_PORT = IN_3;    // חישון אור //

// PID gains - התחילה בערכים אלו ואז כילו
float Kp = 1.2;
float Ki = 0.01;
```

```

float Kd = 0.25;

// ערך חישון שמייצג את מרכז היקו - (target) מטרה //
// לכ"ל במקום: למדוד ערך בתחום היקו (או מוצע בו ערכים כהה ובהיר)* //
int TARGET = 35; // צרי לכ"ל בהתאם למסלול/חישון //

// ---- משתנים פנימיים ----
float integral = 0.0;
float lastError = 0.0;
bool firstLoop = true;

task main()
{
    // ידיא מצב חישון פעיל //
    SetSensorLight(SENSOR_PORT, true);

    // אופציונלי: תצוגה התחלה //
    ClearScreen();
    TextOut(0, LCD_LINE1, "PID line follower");
    TextOut(0, LCD_LINE2, "Calibrate TARGET");

    Wait(1000);

    // איפוא מצב //
    integral = 0.0;
    lastError = 0.0;
    firstLoop = true;

    // לולאת בקר עיקרית //
    while(true)
    {
        int light = Sensor(SENSOR_PORT); // קראית חישון //
        float error = (float)TARGET - (float)light; // שגיאה: חיבוי = רובוט שמאליה מהיעד (תלו' בכ"ל) //

        // אינטגרל עם anti-windup //
        integral += error * (SAMPLE_TIME_MS / 1000.0);
    }
}

```

```

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

// גזרת //
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 מהירות גלגלים // שפטן כרך שמתאים לפניות - correction ביחס ל涑מון //
int leftSpeed = (int) (BASE_SPEED + correction);
int rightSpeed = (int) (BASE_SPEED - correction);

// הגבלת טווח תוקן //
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);

// שירת נתונים ללולאה הבאה //
lastError = error;

// לשם דיבאג/כיו"ל - LCD תציג עלי //
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 לכיוון ושייפור

1. כוֹל TARGET

- הנץ את החישון מעל הקו וכותב על LCD את ערך (3_NI) Sensor (או השתמש בקוד לمعלה ביצורתו) — קבע את ה- TARGET לערך המציג את מרכז הקו (יכול להיות ממוצע בין ערכים על הקו ובוחוץ).

2. סרגל התחלתה של מקדים (Tune)

- התחל ב- $K_i = 0$ ו- $K_d = 0$. הגדל K_p עד שהרובוט מתחילה להתנדד 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 100-30 ms. אם תקטיין את sample time, יכול שוב את המקדים (כי הם תלויים ב- dt).

4. Anti-windup

- קלענו אינטגרל עם גבולות (MIN/MAX) כדי למנוע הצברות גדולות. אפשר להרחיב/להקטין לפי הצורך.

5. כיוון השגיאה

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

6. מהירות בסיס (BASE_SPEED)

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

7. בדיקות

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

אם תרצה שאעשה הלאה

- להמיר את הקוד כך שיירשם ל-CSV (לא בטוח אם ה-CNC שלו תומך בכתיבה לקובץ — אפשר להוציא אם תרצה ואני אבדוק גרסתם עם `FileWrite`).

- לשלב אודומטריה (להוסיף קרייאות encoder ולשמור θ/Y/X בזמן אמיתי).
- להכין סקריפט Python שיקרא לוג מרובוט/קובץ ויצר גרף מסלול וניתוח ביצועים.

רוצה שאוסיף ממש גם:

1. גרסת קוד עם רישום ל-CSV (אם ה-CXC שלך תומך ב-`FileWrite`) — או
2. הרחבהISM שלבב אודומטריה (אם יש לך אופציונלי `encoders`) — או
3. עזרה בכיוול ערכי `Kd/Ki/Kp` לפי תוצאות ניסוי שתريיצ'

איזה מהם מועדיפי — להכין עכשווי?

קישור לשיחה זו -

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