

מדריך תכנות לסטודנטים — RaspTankPro

מדריך זה מסביר כיצד לכתוב תוכניות לרובוט באמצעות **סביבת הסנדבוקס**. תשתמשו בפייתון וסט פשוט של פקודות — אין צורך בידע קודם בחומרה או אלקטרונייה.

תוכן העניינים

1. [סקירת החומרה](#)
2. [תחלית עבודה](#)
3. [תנוועה](#)
4. [חישנים](#)
5. [סרוויים וזרוע](#)
6. [\(אודומטריה ויזואלית \(מעקב מיקום](#)
7. [תוכניות לדוגמה](#)
8. [הרצת בדיקות החומרה](#)
9. [פתרון בעיות](#)

1. סקירת החומרה.

רכיב	תיאור
מנועי הנעה	שני מנועים (שמאל ימין) לשילטה בתנוועה ובפנינה
סרוויים	חמשה סרוויים: סיבוב מצלמה, הטית מצלמה, הטית ראש, זרוע, ואחיזה
חיישן אולטרסוני	(מודד מרחק למכשול הקרוב ביותר (במטרים
MPU6050	(ג'ירוסקופ ומד-תאוצה משולבים (מהירות זוויתית + תאוצה
מצלמה	משמשת לשידור הווידאו ולאודומטריה ויזואלית Raspberry Pi מצלמת

2. תחלית עבודה.

הפעלת הרובוט

1. חיבור מקלדת, עכבר וציג לרובוט.
2. יתחל וויפיע שולחן העבודה Pi-Raspberry הפעלו את הרובוט. ה.
3. **Thonny IDE** נפתח אוטומטית עם [sandbox.py](#).

כתיבה והרצה של התוכנית

- הוא התוכנית שלכם ([run](#)) כל מה שבתוכה [sandbox](#) בתוך ([run](#)) ערכו את פונקציית.
1. להרצה התוכנית (thonny-כ-Run-או לחצו על כפתור ה **F5** לחצו).
 2. Run. שרת הבקרה המקורי מפסיק אוטומטית כאשר לווחצים ◦.
 3. החומרה של הרובוט זמינה עכשו באופן בלבד לתוכנית שלכם ◦.
 4. בכל עת לעצירה בטוחה של הרובוט (**F2** או **Stop** לחצו על כפתור).
 5. כדי לאותל לשרת הבקרה המקורי (מצב הדגמה), הפעילו מחדש את הרובוט.

עיצוב חירום

Am-harobot-barah-o-matnag-betzora-belti-zpohia-oinecm-icoleim-lahgeul-thonny:

- בשולון העבודה **EMERGENCY STOP** לחצו פעמיים על האיקון.
- פעולה זו מפסיקת מיידית את התוכנית הריצה ועצרת את המנוועים.

תנוועה 3.

לא מצוין, אופציוני **duration** בשניות. אם **speed**- מהירות, 0–100) או **duration** כל פונקציות התנוועה מקבלות

```
נעו קדימה 2 שניות במחזית מהירות #  
נעו אחורה שנייה אחת #  
(פנה שמאל (גלגל ימין נוסע, שמאל עוצר #  
פנה ימינה (גלגל שמאל נוסע, ימין עוצר #  
סתובב במקום נגד כיוון השעון #  
סתובב במקום עם כיוון השעון #  
עצור מיד #  
(המתן 1.5 שניות (הרובוט נשאר במקום #  
robot.stop()  
robot.wait(1.5)
```

תיאור הפונקציות

robot.forward(speed=50, duration=None)

נעו קדימה.

- **speed** — 50 (ברירת מחדל: 50)
- **duration** — **None** עד לקריאה ל **duration** לשניות לנסעה; אם — **stop()**

robot.backward(speed=50, duration=None)

נעו אחורה. אותו פרמטרים כמו **forward**.

robot.turn_left(speed=50, duration=None)

פנה שמאלה. גלגל ימין נסע קדימה; גלגל שמאל עצור.

robot.turn_right(speed=50, duration=None)

פנה ימינה. גלגל שמאל נסע קדימה; גלגל ימין עצור.

robot.spin_left(speed=50, duration=None)

סתובב נגד כיוון השעון במקום. שני הגלגלים נעים באותה מהירות בכיוונים מנוגדים.

robot.spin_right(speed=50, duration=None)

סתובכ עם כיוון השעון במקומ.

robot.stop()

עצור את כל המנועים מיידית.

robot.wait(seconds)

המתן למשך מספר השניות הנתון. הרובוט נשאר במקום.

4. חישונים

חישון מרחק

```
distance = robot.get_distance()  
print(f"מטר מהרובוט נמצא {distance:.2f}")
```

robot.get_distance() → float

מחזיר את המרחק (במטרים) למכשול הקרוב ביותר ישירות מול החישון. ערכים מעל ~2 מטר עשויים להיות לא מדויקים. מחזיר 0 אם לא התקבל זה.

ג'ירוסקופ

```
gyro = robot.get_gyro()  
print(f"סיבוב x: {gyro['x']:.2f} y: {gyro['y']:.2f} z: {gyro['z']:.2f} °/s")
```

robot.get_gyro() → dict

'x', 'y', 'z' מחזיר מהירות זוויתית במעלות לשניה כミילון עם מפתחות

משמעות ציר

x אgelol (הטייה לצדדים)

y הצעדה (הטייה קדימה/אחורית)

z פנינה (סיבוב שמאל/ימין)

אם החישון אינו מחובר { 'z': 0, 'y': 0, 'x': 0 } מחזיר.

מד-תאוצה

```
accel = robot.get_accel()
print(f"תאוצה - x: {accel['x']:.2f} y: {accel['y']:.2f} z: {accel['z']:.2f}
g")
```

robot.get_accel() → dict

כasher הרובוט שטוח ומנוחה. x, y, z מילון עם מפתחות $(1 \text{ g} \approx 9.81 \text{ מ/ש}^2)$ ממוצע תאוצה ליניארית ב- 1.0 - x -כ维奇ה ($y \approx 0$).

אם החישון אינו מחובר $\{x: 0, y: 0, z: 1\}$ מחזיר.

5. סרווויים וזרוע.

מייקום בין -1.0 (קצה אחד) ל- 1.0 (קצה השני), כאשר 0.0 הוא המרכז) כל פונקציות הסרווי מתקבלות

<code>robot.set_camera_pan(-0.5)</code>	סובב מצלמה צדי דרך לשמאל #
<code>robot.set_camera_tilt(1.0)</code>	הטה מצלמה לחליותין למעלה #
<code>robot.set_head_tilt(0.0)</code>	מרכז הטיות ראש #
<code>robot.set_arm(0.8)</code>	הרם זרוע 80% מהדרך למעלה #
<code>robot.set_gripper(-1.0)</code>	סגור אחיזה לחליותין #
<code>robot.set_gripper(1.0)</code>	פתח אחיזה לחליותין #
<code>robot.reset_servos()</code>	הזרע את כל הסרווים למרכז #

תיאור הפונקציות

robot.set_camera_pan(position)

סובב את המצלמה שמאליה או ימינה.

- -1.0 = מלא, 0.0 = מרכז, 1.0 = ימין מלא

robot.set_camera_tilt(position)

הטה את המצלמה למעלה או למטה.

- -1.0 = מלא, 0.0 = מרכז, 1.0 = למעלה מלא

robot.set_head_tilt(position)

הטה את ראש הרובוט (פלטפורמת החישונים) למעלה או למטה.

- -1.0 = מלא, 0.0 = מרכז, 1.0 = למעלה מלא

robot.set_arm(position)

הזע את מפרק הזרע למעלה או למטה.

- $-1.0 = \text{לחלוטין למטה}$, $1.0 = \text{מרכז}$, $0 = \text{לחלוטין למעלה}$

`robot.set_gripper(position)`

פתח או סגור את האחיזה.

- $0 = \text{מרכז}$, $1.0+ = \text{פתוח}$, $-1.0 = \text{סגור}$

`robot.reset_servos()`

(החזיר את כל הסרוויסים למיקום המרכז' שליהם (מיקום 0.0).

6. אודומטריה ויזואלית (מעקב מיקום)

אודומטריה ויזואלית מאריכה את מיקום הרובוט על-ידי ניתוח התנועה בין פריים עוקבים של המצלמה. היא זרימה אופטית לקאס-קאנדה כדי לעקוב כיצד הסצנה משתנה מפריים לפריים, ואז FAST משתמש בזיהוי נקודות מחשבת את החזזה התלת-מימדית של המצלמה.

מגבלות חשובות

אי-בahirות סקירה חד-עינית: מצלמה בודדת אינה יכולה לקבוע מרחקים אמיתיים ללא מקור ייחוס חיצוני. הם ביחידות יחסיות, לא מטרים. הסקירה תלויה ב(`get_position`) המוחזרים על-ידי z, y, x, ערכי `absolute_scale` בירית מחדל: 1.0). אם דרושים לכם מרחקים אמיתיים, יש לכיל ערך זה ביחס למרחק) ידוע.

סחיפה: האודומטריה היזואלית צוברת שגיאות קטנות עם הזמן. מסלולים ארוכים יראו סחיפה מהמיוקם האמתי.

דורש (`start_odometry()`. **התנgesות בשימוש במכשיר:** שרת הבקרה המקוון משתמש גם הוא במכשירה דורש F5 ב-Thonny — הדבר קורה אוטומטית כאשר לוחצים.

שימוש

```
# התחילה מעקב מיקום #
robot.start_odometry()

# ... הדז את הרובוט ...
robot.forward(speed=40, duration=3.0)

# ביחידות יחסיות z, y, x קרא את המיקום הנוכחי #
x, y, z = robot.get_position()
print(f"x={x:.2f} y={y:.2f} z={z:.2f}")

# אפס נקודת מוצא למיקום הנוכחי #
robot.reset_position()

# עצור מעקב כשסיימת #
robot.stop_odometry()
```

תיאור הfonוקציות

`robot.start_odometry(focal_length=537.0, pp=(320.0, 240.0), scale=1.0)`

התחל מעקב מיקום ברקע.

- `focal_length` אורך מוקד המצלמה בפיקסלים (ברירת מחדל: 537.0 למצלמת —
ב- 480×640 Pi)
- `pp` (cx, cy) נקודת העיקרון —
(בפיקסלים (ברירת מחדל: 320.0, 240.0))
- `scale` מקדם סקאללה המוחלט על כל שלב תרגום (ברירת מחדל: 1.0)

אם לא ניתן לפתח את המצלמה `RuntimeError` מוחזיר.

`robot.get_position() → (x, y, z)`

של שלושה מספרים עשרוניים. נקודת המוצא היא מיקום הרובוט כאשר `start_odometry()` מוחזיר את הערךת המיקום העדכנית כ-`tuple` (`reset_position()` או `start_odometry()` מוחזיר `RuntimeError` אם לא נקרא `start_odometry()`).

`robot.reset_position()`

אם לא נקרא `RuntimeError` אף אם המיקום הנווכי ל-`(0, 0, 0)`. מוחזיר.

`robot.stop_odometry()`

עצור מעקב מיקום ושחרר את המצלמה.

תוכניות לדוגמה 7.

דוגמה 1 — نوع קדימה ועצור לפני מכשול

```
def run():
    SAFE_DISTANCE = 0.30 # מטרים

    robot.forward(speed=40)    # נסוע מהמשך duration (לאילו)

    while True:
        distance = robot.get_distance()
        print(f"גלאי: {distance:.2f} מ'")

        if distance < SAFE_DISTANCE:
            robot.stop()
            print("דוודה מכשול! עוזר")
            break

    robot.wait(0.05) # הושהייה קצרה בין קריאות
```

דוגמה 2 — סריקת מצלמה עם קריאות מרחק

```

def run():
    import time

    print("קָרְסָוּ...")
    positions = [-1.0, -0.5, 0.0, 0.5, 1.0]

    for pos in positions:
        robot.set_camera_pan(pos)
        robot.wait(0.5) # המתן עד שהсерוו יתני צב
        distance = robot.get_distance()
        angle_label = f"{int(pos * 90)}:{+d}°"
        print(f" {angle_label}: {distance:.2f} מ'")

    robot.reset_servos()
    print("הסרים הושלמה.")

```

דוגמה 3 — רישום נתונים חיישנים לקובץ

```

def run():
    import csv
    import time

    LOG_FILE = '/home/pi/sensor_log.csv'
    DURATION = 10.0 # שניות

    print(f"רואם חיישנים למשך {DURATION} → {LOG_FILE}")

    with open(LOG_FILE, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['time_s', 'distance_m',
                        'gyro_x', 'gyro_y', 'gyro_z',
                        'accel_x', 'accel_y', 'accel_z'])

        start = time.time()
        while time.time() - start < DURATION:
            t = time.time() - start
            dist = robot.get_distance()
            gyro = robot.get_gyro()
            accel = robot.get_accel()

            writer.writerow([
                round(t, 3), dist,
                gyro['x'], gyro['y'], gyro['z'],
                accel['x'], accel['y'], accel['z'],
            ])
            print(f"t={t:.1f}s  מַרְאֵת={dist:.2f}מ'  "
                  f"gyro_z={gyro['z']:.2f}")
            time.sleep(0.1)

```

("בגילוון אלקטרוני לניטוח הנתונים CSV-סיוום. פתחו את קובץ ה-")

הרצת בדיקות החומרה 8.

כדי לוודא שכל החומרה עובדת כראוי:

```
# בקוד הבא (run) החליפו את תוכן run.py  
from robot_test import run_all_tests  
  
def run():  
    run_all_tests(robot)
```

לכל אחד. הבדיקה המלאה נמשכת כ-30 שניות [FAIL] או [PASS] סקריפט הבדיקה יבדוק כל רכיב ברכף ויצא ומציה את הרובוט פיזית, לכן לוודאו שיש מקום סבירו

פתרון בעיות 9.

"Cannot open camera ל start_odometry() - בעת קריאה ל"

הצלמה בשימוש על-ידי תחילה אחר (בדרך כלל שרת הבקרה המקורי). לוודאו שהרצתם את התוכנית דרך (F5). שופסיק את השרת אוטומטית לפני התחלה הקוד.

חישון בהפעלה / I2C שגיאות

מחזירות ערכי (get_gyro() | get_accel() |), אם האתחול נכשל I2C מתקשר דרך MPU6050 המד-תאוצה/ג'ירוסקופ מופעל I2C-אפס במקום לקרים. בדקו שהחישון מחובר ו sudo raspi-config → Interface Options → I2C.

סרויים לא זים

מופעל. אם רק חלק מהסרוויים I2C-אוטו אפיק כמו הג'ירוסקופ. בדקו את החיבורים ו(I2C-בקר הסרווי משתמש בPWM-נכשלים, יתכן שיש בעיה בחיבור ערוץ ה

F2 הרובוט לא עוצר כאשר לוחצים

שעוצר את (cleanup() Thonny-robot.sandbox.py). לסקריפט הרץ SIGTERM שלוחם בשולחן העבודה **EMERGENCY STOP** המנועים. אם הרובוט עדין נע, השתמשו בקייזר הדרך.

בhfупלה GPIO אזהרות

זה לא מזיך — GPIO-Raspberry Pi עשוי להציג RuntimeWarning: This channel is already in use. לא שוחרר כראוי ביריצה קודמת. הרובוט עדין יעבד כהלה GPIO-פירושו שהוא

המנועים מסתובבים אך הרובוט לא נסע ישיר

דרך radius גובה יותר, או שנו את הפרמטר speed לשני המנועים עשוי להיות עיליות שונה מעט. השתמשו בערך move.move() (ישירות לשליטה מדינית יותר (שימוש מתקדם)).

