

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 אלקטרוניקה.

תוכן העניינים

- סקירת החומרה
- תחילת עבודה
- תנועה
- חיישנים
- סרוויים וזרוע
- (אודומטריה ויזואלית) (מעקב מיקום)
- תוכניות לדוגמה
- הרצת בדיקות החומרה
- פתרון בעיות

1. סקירת החומרה

רכיב	תיאור
מנועי הנעה	שני מנועים (שמאל וימין) לשליטה בתנועה ובפנייה
סרוויים	חמישה סרוויים: סיבוב מצלמה, הטיית מצלמה, הטיית 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. תחילת עבודה

הפעלת הרובוט

- חברו מקלדת, עכבר וצג לרובוט.
- יאתחל ויופיע שולחן העבודה Raspberry Pi-הפעילו את הרובוט. ה.
- טעון — זו סביבת התכנות שלכם `sandbox.py` נפתח אוטומטית עם **Thonny IDE**.

כתיבה והרצה של התוכנית

- הוא התוכנית שלכם `run()` כל מה שבתוך `sandbox.py` בתוך `run()` ערכו את פונקציית.
- להרצת התוכנית (Thonny-ב-Run או לחצו על כפתור ה) **F5** לחצו.
 - Run. שרת הבקרה המקוון מופסק אוטומטית כאשר לוחצים.
 - החומרה של הרובוט זמינה עכשיו באופן בלעדי לתוכנית שלכם.
- בכל עת לעצירה בטוחה של הרובוט (**F2** או) **Stop** לחצו על כפתור.
- כדי לאתחל לשרת הבקרה המקוון (מצב הדגמה), הפעילו מחדש את הרובוט.

עצירת חירום

Thonny-אם הרובוט בורח או מתנהג בצורה בלתי צפויה ואינכם יכולים להגיע ל

- בשולחן העבודה **EMERGENCY STOP** לחצו פעמיים על האייקון.
- פעולה זו מפסיקה מיידית את התוכנית הרצה ועוצרת את המנועים.

3. תנועה

לא מצוין, **duration** אופציונלי בשניות. אם **duration**-מהירות, (0–100) ו **speed** כל פונקציות התנועה מקבלות **robot.stop()**-הרובוט ימשיך לנוע עד שתקראו ל.

```
robot.forward(speed=50, duration=2.0) # נוע קדימה 2 שניות במהירות
robot.backward(speed=50, duration=1.0) # נוע אחורה שנייה אחת
robot.turn_left(speed=50, duration=0.5) # (פנה שמאלה (גלגל ימין נוסע, שמאל עוצר)
robot.turn_right(speed=50, duration=0.5) # (פנה ימינה (גלגל שמאל נוסע, ימין עוצר)
robot.spin_left(speed=40, duration=1.0) # סתובב במקום נגד כיוון השעון
robot.spin_right(speed=40, duration=1.0) # סתובב במקום עם כיוון השעון
robot.stop() # עצור מיד
robot.wait(1.5) # (המתן 1.5 שניות (הרובוט נשאר במקום
```

תיאור הפונקציות

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

נוע קדימה.

- **speed** — 50 (ברירת מחדל: 0–100) (עוצמת מנוע, (ברירת מחדל: 50
- **duration** — **stop()** נוסע עד לקריאה ל, **None** שניות לנסיעה; אם

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

forward. נוע אחורה. אותם פרמטרים כמו

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

פנה שמאלה. גלגל ימין נוסע קדימה; גלגל שמאל עוצר

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

פנה ימינה. גלגל שמאל נוסע קדימה; גלגל ימין עוצר

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)

סתובב עם כיוון השעון במקום.

robot.stop()

עצור את כל המנועים מיידית.

robot.wait(seconds)

המתן למשך מספר השניות הנתון. הרובוט נשאר במקום.

4. חיישנים

חיישן מרחק

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

robot.get_distance() → float

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

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

מחזיר מהירות זוויתית במעלות לשנייה כמילון עם מפתחות 'x', 'y', 'z'.

ציר	משמעות
x	(גלגול) הטייה לצדדים
y	(הצעדה) הטייה קדימה/אחורה
z	(פנייה) סיבוב שמאל/ימין

אם החיישן אינו מחובר `{'x': 0, 'y': 0, 'z': 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

$z \approx$ כאשר הרובוט שטוח ובמנוחה. 'x', 'y', 'z' כמילון עם מפתחות $g \approx 9.81$ (1 g) מחזיר תאוצה ליניארית ב $x \approx y \approx 0$ (כבידה) ו $z \approx 1.0$.

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

5. סרוויים וזרוע

מיקום) בין -1.0 (קצה אחד) ל-1.0+ (הקצה השני), כאשר 0.0 הוא המרכז) **position** כל פונקציות הסרווי מקבלות.

<code>robot.set_camera_pan(-0.5)</code>	# סובב מצלמה חצי דרך לשמאל
<code>robot.set_camera_tilt(1.0)</code>	# הטה מצלמה לחלוטין למעלה
<code>robot.set_head_tilt(0.0)</code>	# מרכז הטיית ראש
<code>robot.set_arm(0.8)</code>	# הרם זרוע 80% מהדרך למעלה
<code>robot.set_gripper(-1.0)</code>	# סגור אחיזה לחלוטין
<code>robot.set_gripper(1.0)</code>	# פתח אחיזה לחלוטין
<code>robot.reset_servos()</code>	# החזר את כל הסרוויים למרכז

תיאור הפונקציות

robot.set_camera_pan(position)

סובב את המצלמה שמאלה או ימינה.

- שמאל מלא, $0.0 =$ מרכז, $1.0+ =$ ימין מלא $= -1.0$

robot.set_camera_tilt(position)

הטה את המצלמה למעלה או למטה.

- למטה מלא, $0.0 =$ מרכז, $1.0+ =$ למעלה מלא $= -1.0$

robot.set_head_tilt(position)

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

- למטה מלא, $0.0 =$ מרכז, $1.0+ =$ למעלה מלא $= -1.0$

robot.set_arm(position)

הזז את מפרק הזרוע למעלה או למטה.

- לחלוטין למטה, $0.0 =$ מרכז, $+1.0 =$ לחלוטין למעלה $= -1.0$

`robot.set_gripper(position)`

פתח או סגור את האחיזה

- סגור לחלוטין, $0.0 =$ מרכז, $+1.0 =$ פתוח לחלוטין $= -1.0$

`robot.reset_servos()`

(החזר את כל הסרוויים למיקום המרכזי שלהם) מיקום 0.0

6. (מעקב מיקום) ויזואלית (אודומטריה ויזואלית)

אודומטריה ויזואלית מעריכה את מיקום הרובוט על-ידי ניתוח התנועה בין פריימים עוקבים של המצלמה. היא **זרימה אופטית לוקאס-קאנדה** כדי לעקוב כיצד הסצנה משתנה מפריים לפריים, ואז **FAST** משתמשת ב**זיהוי נקודות** מחשבת את ההזזה התלת-מימדית של המצלמה.

מגבלות חשובות

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

סחיפה: האודומטריה הויזואלית צוברת שגיאות קטנות עם הזמן. מסלולים ארוכים יראו סחיפה מהמיקום האמיתי.

דורש `start_odometry()`. **התנגשות בשימוש במצלמה:** שרת הבקרה המקוון משתמש גם הוא במצלמה F5 ב-Thonny. שהשרת יהיה מופסק — הדבר קורה אוטומטית כאשר לוחצים

שימוש

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

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

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

# אפס נקודת מוצא למיקום הנוכחי
robot.reset_position()

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

תיאור הפונקציות

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

התחל מעקב מיקום ברקע.

- `focal_length` — 537.0 למצלמת (ברירת מחדל: 537.0 למצלמת Pi 480×640)
- `pp` — (בפיקסלים (ברירת מחדל: (240.0, 320.0) נקודת העיקרון (cx, cy))
- `scale` — 1.0 (ברירת מחדל: 1.0) (מקדם סקאלה המוחל על כל שלב תרגום)

אם לא ניתן לפתוח את המצלמה `RuntimeError` מחזיר.

`robot.get_position()` → (x, y, z)

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

`robot.reset_position()`

`start_odometry()` אם לא נקרא `RuntimeError` אפס את המיקום הנוכחי ל-(0, 0, 0). מחזיר.

`robot.stop_odometry()`

עצור מעקב מיקום ושחרר את המצלמה.

7. תוכניות לדוגמה

דוגמה 1 — נוע קדימה ועצור לפני מכשול

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

    robot.forward(speed=40) # (ללא duration - התחל לנסוע)

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

        if distance < SAFE_DISTANCE:
            robot.stop()
            print("זוהה מכשול! עוצר.")
            break

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

דוגמה 2 — סריקת מצלמה עם קריאות מרחק

```
def run():
    import time

    print("סורק...")
    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 שהסרור יתייצב
        distance = robot.get_distance()
        angle_label = f"{int(pos * 90):+d}°"
        print(f" {angle_label}: {distance:.2f} מ'")

    robot.reset_servos()
    print("הסריקה הושלמה.")
```

דוגמה 3 — רישום נתוני חיישנים לקובץ

```
def run():
    import csv
    import time

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

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

    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 מרחק={dist:.2f}מ' "
              f"gyro_z={gyro['z']:.2f}")
        time.sleep(0.1)
```

```
print("בגיליון אלקטרוני לניתוח הנתונים CSV-סיום. פתחו את קובץ ה")
```

8. הרצת בדיקות החומרה

כדי לוודא שכל החומרה עובדת כראוי:

```
# בקוד הבא run() החליפו את תוכן, ב-sandbox.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 ברצף ויציא ומזיזה את הרובוט פיזית, לכן וודאו שיש מקום סביבו.

9. פתרון בעיות

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

Thonny המצלמה בשימוש על-ידי תהליך אחר (בדרך כלל שרת הבקרה המקוון). וודאו שהרצתם את התוכנית דרך (F5), שמפסיק את השרת אוטומטית לפני התחלת הקוד.

חיישן בהפעלה / I2C שגיאות

מחזירות ערכי `get_gyro()` ו-`get_accel()`, אם האתחול נכשל. I2C מתקשר דרך MPU6050 המד-תאוצה/ג'ירוסקופ (F5), בדקו שהחיישן מחובר ומופעל I2C-אפס במקום לקרוס. `sudo raspi-config` → Interface Options → I2C).

סרוויים לא זזים

מופעל. אם רק חלק מהסרוויים I2C-אותו אפיק כמו הג'ירוסקופ). בדקו את החיבורים (וש I2C-בקר הסרוו משתמש ב PWM-נכשלים, ייתכן שיש בעיה בחיבור ערוץ ה

F2 הרובוט לא עוצר כאשר לוחצים

שעוצר את `robot.cleanup()`-קולט אות זה וקורא ל `sandbox.py`. לסקריפט הרץ SIGTERM שולח Thonny בשולחן העבודה **EMERGENCY STOP** המנועים. אם הרובוט עדיין נע, השתמשו בקיצור הדרך

בהפעלה GPIO אזהרות

זה לא מזיק — `RuntimeWarning: This channel is already in use.` עשוי להציג Raspberry Pi של GPIO-ה. לא שוחרר כראוי בריצה קודמת. הרובוט עדיין יעבוד כהלכה GPIO-פירושו שה

המנועים מסתובבים אך הרובוט לא נוסע ישר

דרך `radius` גבוה יותר, או שנו את הפרמטר `speed` לשני המנועים עשויה להיות יעילות שונה מעט. השתמשו בערך `move.move()` (שימוש מתקדם).

