The Iby and Aladar Fleischman Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

Autonomous Nano Drones Indoor Navigation System Based on Depth-RGB Image Processing

סער אדוטלר 208414466
מור יגל 207117748
מנחה: יונתן מנדל, אוניברסיטת תל אביב

פרויקט גמר מס' 18-1-1-1622

ספר פרויקט

תוכן עניינים

תקציר

מטרת הפרויקט היא לפתח מערכת **ניידת וקומפקטית** לשליטה ובקרת מיקום ריכוזית, המאפשרת לתכנן ולבצע **ניווט אוטונומי** של **קבוצת** רחפנים זעירים מסוג "CrazyFlie" בצורה **מתואמת**, בסביבת **פנים**, תחת אילוצי זמן אמת.

המערכת, מורכבת ממספר תתי-מערכות:

1. **רחפן זעיר** (Nano-Drone) מסוג "CrazyFlie" – בעל בקר ניווט פנימי.
2. **מצלמת עומק** (Intel Realsense D435) – מספקת תמונת עומק של סביבת הניווט.
3. **מערכת שליטה ריכוזית** – רצה בסביבת ROS (Robotic Operating System).

אופן פעולת מערכת הפרויקט:

1. על כל רחפן מונח כדור בצבע ייחודי המסמל אותו.
2. המערכת שולפת מהמצלמה תמונת צבע ותמונת עומק של סביבת הניווט.
3. המערכת מעבדת את המידע שהתקבל מהמצלמה, מאתרת את מיקומם של הרחפנים (ביחס אליה) בסביבת הניווט, ומפיצה אותם אל הרחפנים.
4. במקביל, המערכת שולחת לכל רחפן מיקום יעד.
5. הרחפן, שמקבל את מיקומו הנוכחי (בכל רגע) ומיקום היעד, ניעזר בבקר (הנמצא על הרחפן) לצורך חישוב ושליחת פקודות מתאימות למנועיו עד להגעתו ליעד.

במסגרת תוצרי הפרויקט, המערכת שפותחה מאפשרת לתכנן ולבצע מסלולי ניווט מדויקים שיבצעו קבוצת רחפנים באופן מתואם, כגון ריחוף יציב, ניווט בין נקודות ציון, עקיבה רציפה אחר מטרה, ועוד.

הקדמה

- <u>מטרות הפרויקט</u>

- לפתח מערכת **ניידת וקומפקטית** לשליטה ובקרת מיקום ריכוזית, המאפשרת לתכנן ולבצע **ניווט אוטונומי** של **קבוצת** רחפנים זעירים מסוג "CrazyFlie" בצורה **מתואמת**, בסביבת **פנים**, תחת אילוצי זמן אמת.

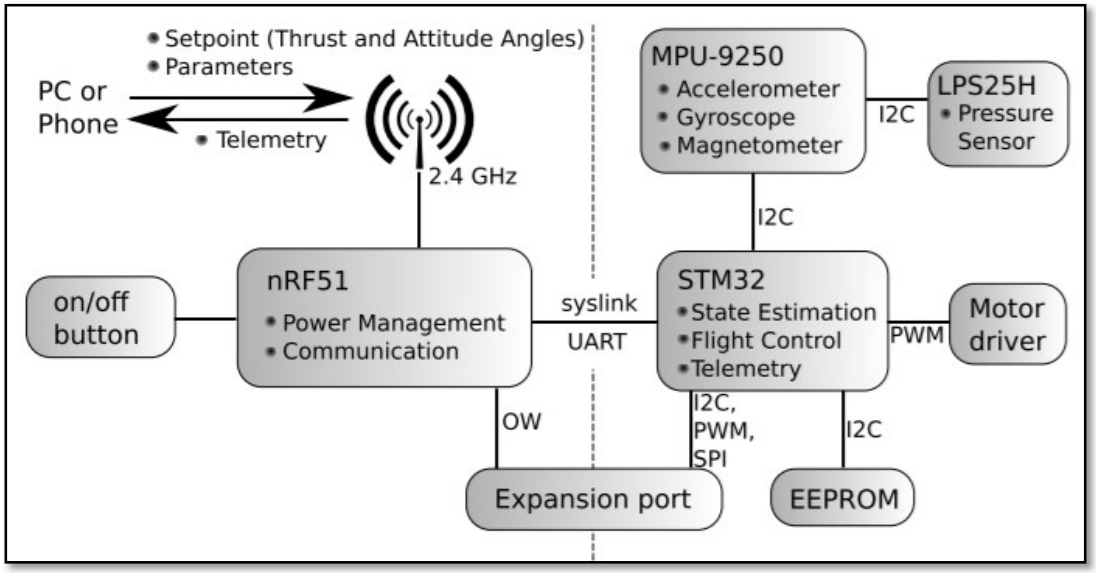
- <u>המוטיבציה</u>

- תחום פיתוח הרחפנים בכלל, הוא מוקד עניין משמעותי בשנים האחרונות הן במחקר והן בתעשייה לצרכים שונים.
- בעוד קיימים ונפוצים פיתוחים רבים של הטסה ידנית של רחפן יחיד (ע"י שלט לדוגמה), קיימים הרבה פחות פתרונות ופיתוחים בתחום טיסה אוטונומית לרחפנים, עוד פחות מזה של טיסה אוטונומית במרחב סגור (שכן טיסה במרחב חיצוני מתבססת "בקלות" על G.P.S), ועוד פחות מזה של טיסה אוטונומית ומתואמת של "להקת" רחפנים, כאשר הפתרונות הקיימים בתחום זה (לדוגמא, שימוש במערך מצלמות לשם מציאת מיקום הרחפן במרחב) דורשים מערכת שליטה המורכבת מציוד רב, קשה לשינוע, הצבה, ושימוש בפועל. מכאן, מגיע הצורך לפתרון המשתמש בציוד נייד ומינימאלי למימוש מערכת שליטה שכזאת.

מרכיבי המערכת

רחפן BitCraze CrazyFile 2.0:

- רחפן Open-Source, Open-Hardware שנבנה למטרות מחקר, השכלה ופיתוח.
- רחפן זעיר, 92 מ"מ מרחק אלכסוני מנוע-מנוע, וקל משקל, 29 גרם בלבד.
- נשלט מ-PC באמצעות "CrazyRadio PA USB dongle" (מצורף איור).



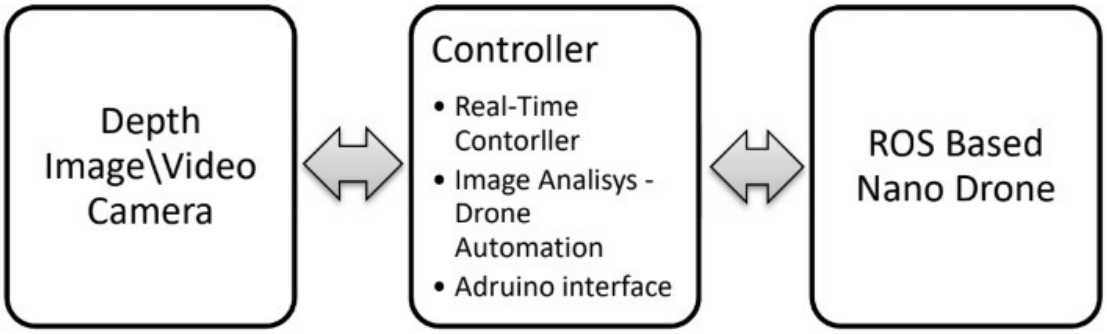
- ארכיטקטורת החומרה של הרחפן (המוצגת באיור המוצג מעלה), מורכבת מ-2 מיקרו בקרים. מיקרו בקר ראשון (Cortex M4) STM32 הוא הבקר הראשי שאחראי על בקרת הטיסה ומיקרו בקר שני (Cortex M0) nRF51 אחראי על התקשורת האלחוטית, המתבצעת בתדרי 2.4GHz (+offset שונה עבור כתובת ייחודית לכל רחפן בהטסה של "להקת" רחפנים).

מצלמת עומק – Intel RealSense D435:

- מצלמת צבע-עומק קומפקטית מבית אינטל.
- מתחברת למחשב על ידי ממשק USB 3.0.
- בעלת חיישן RGB, שני חיישני עומק, ו- IR Projector.
- מספקת תמונת צבע ברזולוציית 1920x1080-30FPS ותמונת עומק 1280x720-90FPS.

תכנון ומימוש

הן בתוכנית העבודה והן בדוח המעקב, הוצג תרשים בלוקים (מוצג מטה) המציג תמונה מופשטת של המערכת.



במהלך תכנון הפרויקט, היה עלינו לתכנן ולממש את מערכת השליטה (הבלוק האמצעי), וכן את התקשורת שלה עם שאר גורמי המערכת, אשר בוצעו באופן הבא:

1. שליפת תמונת הצבע-עומק מהמצלמה:
   - תחילה, נעזרנו ב- Intel RealSense SDK 2.0, בכדי לקשר בין המחשב והמצלמה
   - לאחר מכן, נעזרנו ב- pyrealsense2, אשר מהווה wrapper של ה-SDK הנ"ל עבור Python ומאפשר שימוש בפונקציות המצלמה (ביניהם שליפת תמונות הצבע והעומק, הגדרת פרמטרים שונים של המצלמה, ועוד) ב-Python.
2. שימוש בתמונת הצבע-עומק לשם איתור מיקום הרחפן במרחב:
   - על כל רחפן, הונח כדור בצבע ייחודי, המסמל אותו.
   - באמצעות עיבוד תמונת הצבע, המערכת מוצאת את מיקומם של כל אחד מהכדורים בתמונה.
   - באמצעות מיקום הכדור בתמונת הצבע, המערכת ניעזרת בתמונת העומק ומחשבת את מיקום הכדור **במרחב** (קואורדינטיות מרחביות, במטרים, ביחס למיקום המצלמה) ולבסוף, מתאימה כל אחד מהכדורים לרחפן המתאים לו (על פי צבעם) ובכך יודעת את מיקומם.
3. העברת המיקום הנוכחי של הרחפן אל הבקר (הממוקם על הרחפן עצמו):
   - שימוש במערכת מבוססת ROS (מפורט בהמשך)

מהו ROS?

- ROS[1] היא אוסף של מסגרות עבודה לפיתוח תוכנה המיועדת לפיתוח פרויקטים Open-Source בתחום הרובוטיקה בכלל, והרחפנים בפרט, אשר מייצרת אבסטרקציה בתהליך הפיתוח, ומהווה סביבה נוחה ויעילה לאינטגרציה בין מודולים שונים, שכל אחד נתמך ע"י חבילת ROS מתאימה.

במימוש הפרויקט, נעזרנו בשני חבילות ROS עיקריות:

1. crazyflie_ros, אשר מהווה חבילת ROS לתקשורת מול רחפני crazyflie.
2. realsense2_camera, אשר מהווה חבילת ROS לתקשורת מול מצלמות Intel RealSense, בה השתמשנו בתחילת הפרויקט לשם למידה על המצלמה ודרכי השימוש בה, אך בתצורת הפרויקט הסופית לא בוצע שימוש בחבילה זו.

[1] ROS Logo

פירוט אופן הפעולה של המערכת (בצירוף קטעי קוד):

המערכת מחולקת ל-3 גורמים עיקריים:

1. מציאת מיקום הרחפנים במרחב:

- המערכת תחילה מתחברת אל המצלמה, ומגדירה את הרזולוציה הפורמט והקצב של תמונות הצבע והעומק שיתקבלו ממנה (בחרנו לעבוד עם רזולוציה של 640x480 על 30FPS בכדי שתהליך עיבוד התמונה ואיתור הרחפנים יהיה מהיר ויעיל ככל האפשר):

```
# create, config and start realsense streaming pipeline
pipeline = rs.pipeline()
config = rs.config()
# RGB and depth:
# Resolution 640 x 480 (for faster proccessing, default of depth is 1080p)
# FPS : 30
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.rgb8, 30)
pipeline.start(config)
```

- לאחר אתחול המצלמה, הגדרת קבועים (לדוגמא גבולות הצבע הירוק והאדום בHSV) וכו, מתחילה הלולאה (האינסופית) המרכזית, אשר מבצעת את תהליך שליפת התמונה מהמצלמה, עיבוד התמונה, איתור הכדורים הייחודיים ולבסוף שליחת המיקום של הרחפנים הלאה:

```
while True:
        iterNum += 1

        rgb , depth = get_aligned_rgb_depth()
        if (depth is None):
                continue

        # set BGR frame standard, resize, blur, and convert it to HSV
        frame = cv2.cvtColor(rgb, cv2.COLOR_RGB2BGR)
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)

        # get hsv frames
        hsvG = cv2.cvtColor(blurred, cv2.COLOR_RGB2HSV)
        hsvR = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

        cntsG = get_contour_from_hsv(hsvG , lowerG , upperG , show_mask = False)
        cntsR = get_contour_from_hsv(hsvR, lowerR , upperR, lowerBound2 = lowerR2 , upperBound2 =
upperR2 , show_mask = False)

        ProccessCircle(frame , cntsG , depth , GNum , graph_iter = 0)
        ProccessCircle(frame , cntsR , depth , RNum , graph_iter = 0)

        # show the frame to our screen
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the 'q' key is pressed, stop the loop
        if key == ord("q"):
                break

# close all windows
cv2.destroyAllWindows()
```

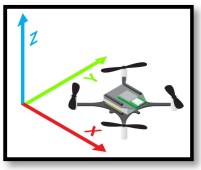
- בתחילה הלולאה מתבצעת קריאה לפונקציה ()get_aligned_rgb_depth (ראה בהמשך), פונקציה זו שולפת מהמצלמה את תמונת הצבע והעומק, מבצעת תהליך alignment (מבצע חפיפה בין תמונת הצבע והעומק שכן חיישן הצבע וחיישני העומק בעלי מיקום **פיזי** שונה), ולבסוף מחזירה את תמונות הצבע והעומק לאחר ביצוע החפיפה ביניהן
- לאחר מכן, תמונת הצבע מומרת לפורמט BGR (לשם עבודה עם OpevCV) ועוברת תהליך Blurring (הפעלת רעש גאוסיאני על תמונת הצבע לטשטוש קל של התמונה – שוב לשם ייעול של תהליך עיבוד תמונת הצבע), ולאחר מכן מומרת לפורמט HSV.

```
def get_aligned_rgb_depth():
        # grab the current frame -> rgb and depth
        curr_frame = pipeline.wait_for_frames()

        # Create alignment primitive with **depth** as its target stream:
        align = rs.align(rs.stream.depth)
        curr_frame = align.process(curr_frame)

        depth = curr_frame.get_depth_frame().as_depth_frame()
        rgb = curr_frame.get_color_frame()
        if not (rgb and depth):
                return (None , None)

        rgb = np.asanyarray(rgb.get_data())
        return (rgb , depth)
```

- לאחר שבידינו תמונת הצבע המעובדת בפורמט HSV, מתבצעת קריאה לפונקציה get_contour_from_hsv (ראה בהמשך) אשר בהינתן תמונת הצבע בפורמט HSV וגבולות של צבע מסוים, מחזירה מערך של כל ה-Contours של צבע זה באותה תמונה.

```
def get_contour_from_hsv(hsv , lowerBound , upperBound , lowerBound2 = None , upperBound2 = None ,
show_mask = False):
        # construct a mask for the color "green", then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        mask = cv2.inRange(hsv, lowerBound, upperBound)

        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        if not (lowerBound2 == None or upperBound2 == None):
                mask2 = cv2.inRange(hsv, lowerBound2, upperBound2)
                mask2 = cv2.erode(mask2, None, iterations=2)
                mask2 = cv2.dilate(mask2, None, iterations=2)
                mask = cv2.bitwise_or(mask, mask2)
        if show_mask is True:
                cv2.imshow("Mask", mask)

        # find contours in the mask
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)

        return cnts
```

- לבסוף, מתבצעת קריאה לפונקציה ProccessCircle (ראה בהמשך) אשר בהינתן מיקומי ה-Contours שנמצאו ותמונת העומק, מבצעת את התהליך המרכזי של הלולאה, אשר כולל בין היתר: מציאת ה-Contour המקסימאלי וסימונו בתמונת הצבע (כמו כן סימון של "היסטורית" ה-Contours שנמצאו), שליפת ערך העומק של אותו Contour והמרת הקואורדינטות שהתקבלו ליחידות מרחק מציאותיות (מפיקסלים למטרים), אתחול המיקום ההתחלתי של הכדורים (ממוצע על 50 הדגימות הראשונות), חישוב מיקום יחסי של הדגימות הבאות ביחס לממוצע ההתחלתי (ואחסון שלהם לצרכי Debugging לדוגמא הצגת גרף של היסטורית הדגימות), ושליחת המיקומים שנמצאו אל הגורם הבא במערכת.
- בנוסף לאתחול המיקום היחסי ממנו ימדדו הדגימות של מיקום הרחפן, נדרשה התאמה בין צירי הרחפן והמצלמה, לשם כך נקבע כי פניו של הרחפן תמיד יהיו מופנים אל המצלמה, וערכי הדגימות שונו בהתאם לצירי הרחפן:



```
def ProccessCircle(frame , cnts , depth , ballNum , graph_iter = 0):

        # only proceed if at least one contour was found
        if len(cnts) > 0:

                ((x, y), radius) = cv2.minEnclosingCircle(max(cnts, key=cv2.contourArea))

                # only proceed if the radius meets a minimum size
                if radius > 5:

                        # draw the circle, centroid and line on the frame
                        markBallInFrame(frame , x ,y , radius , ballNum)

                        # convert from pixels coordinates to real world cordinates
                        depth_val = depth.get_distance(int(x),int(y))
                        depth_intrin = depth.profile.as_video_stream_profile().intrinsics
                        (x , y , z) = rs.rs2_deproject_pixel_to_point(depth_intrin , [x,y] , depth_val)

                        # update x , y , z according to initial state and publish
                        update_publish_axes(x , y , z , ballNum)

                        # build and output graphes if needed (debugging purposes)
                        if graph_iter > 0:
                                if iterNum == graph_iter:
                                        iters = [i for i in range(len(axes[ballNum]))]
                                        plt_axes = zip(*axes[ballNum])
                                        plt.plot(iters , list(plt_axes[0]), 'r--', iters ,
list(plt_axes[1]), 'bs', iters , list(plt_axes[2]), 'g^')
                                        plt.show()
```

2. קונפיגורציה של הרחפנים, התחברות אליהם, והעברת מיקום הרחפנים אל הבקר:
   - גורם זה מנוהל באופן מלא על ידי קובץ Launch – קובץ קונפיגורציה ראשי של כלל הפרמטרים והתהליכים שייווצרו עם הארגומנטים המתאימים.
   - בקובץ ה-Launch, מוגדרים כתובות הרחפנים (הצרובים ב-Firmware של כל רחפן), ה-Topic הייחודי של כל רחפן (אליו כותב הגורם הקודם את מיקום הרחפן הנוכחי), שמות מייצגים עבור הרחפנים, וכן פרמטרים שונים עבור התקשורת מול הרחפנים.

```
<launch>
  <arg name="uri1" default="radio://0/80/2M/E7E7E7E708" /> <!-- Red Ball -->
  <arg name="frame1" default="crazyflieR" />
  <arg name="topic1" default="/ball_poseR" />

  <arg name="uri2" default="radio://0/80/2M/E7E7E7E702" /> <!-- Green Ball -->
  <arg name="frame2" default="crazyflieG" />
  <arg name="topic2" default="/ball_poseG" />


  <include file="$(find crazyflie_driver)/launch/crazyflie_server.launch">
  </include>
```

```
  <group ns="$(arg frame1)">
    <node pkg="crazyflie_driver" type="crazyflie_add" name="crazyflie_add" output="screen">
      <param name="uri" value="$(arg uri1)" />
      <param name="tf_prefix" value="$(arg frame1)" />
      <param name="enable_logging" value="False" /> <!--Logging disabled to reduce bandwith-->
      <param name="enable_logging_imu" value="False" />
      <param name="enable_logging_temperature" value="False" />
      <param name="enable_logging_magnetic_field" value="False" />
      <param name="enable_logging_pressure" value="False" />
      <param name="enable_logging_battery" value="False" />
      <param name="enable_logging_packets" value="False" />
      <rosparam>
        genericLogTopics: ["log1"]
        genericLogTopicFrequencies: [100]
        genericLogTopic_log1_Variables: ["stateEstimate.x", "stateEstimate.y", "stateEstimate.z"]
      </rosparam>
    </node>

    <node name="publish_external_position_RS_OBC_R" pkg="crazyflie_demo"
type="publish_external_position_RS.py" args ="$(arg frame1)" output="screen">
      <param name="topic" value="$(arg topic1)" />
    </node>
  </group>
```

- עבור כל רחפן מוגדר group עם מרחב שם (namespace) ייחודי. כל group יכיל 2 nodes: node שיפעיל שירות של הוספת הרחפן לרשימת הרחפנים שמנהל השרת (וכן הגדרת מאפייני החיבור), ו-node שני שיפעיל את התהליך של ה-publisher (יפורט בהמשך).
- תהליך ה-publisher הינו python script שתפקידו לשלוף את מיקום הרחפן הנוכחי מה-Topic הייחודי אליו כותבת תכנית איתור הרחפנים (הוצגה מעלה), לעדכן את הפרמטרים של המשערך שעל גבי הרחפן באמצעות נתוני מיקומי פתיחה, להמיר את סוג ההודעה לסוג המתאים לשרת, ובסופו של דבר להפיץ נתוני מיקום של הרחפן לשרת, שאחר כך ישדר אותם לרחפן עצמו.
- תהליך ה-publisher תחילה מבצע Subscribe ל-Topic של אותו רחפן (מתקבל כפרמטר), לשם קריאת מיקום הרחפן הנוכחי, ובמקביל מגדיר Publisher, לשם שליחת המיקום לבקר.

```
if __name__ == '__main__':
    rospy.init_node('publish_external_position_RS', anonymous=True)
    topic = rospy.get_param("~topic")
    rospy.wait_for_service('update_params')
    rospy.loginfo("found update_params service " + topic)
    update_params = rospy.ServiceProxy('update_params', UpdateParams)
    firstTransform = True

    msg = PointStamped()
    msg.header.seq = 0
    msg.header.stamp = rospy.Time.now()
    pub = rospy.Publisher("external_position", PointStamped, queue_size=1)
    rospy.Subscriber(topic, PoseStamped, onNewTransform)
    rospy.spin()
```

- בכל קריאה של ה-Subscriber, מתבצעת פעולת callback, שתפקידה להמיר את מיקום הרחפן שהתקבל לצורה המתאימה עבור הבקר (תוך שימוש במשערך Kalman), וכן מבצעת אתחול למשערך על פי הדגימה הראשונה שהתקבלה.

```
def onNewTransform(pose):
    global msg
    global pub
    global firstTransform

    if firstTransform:
        # initialize kalman filter
        rospy.set_param("kalman/initialX", pose.pose.position.x) # Change pose according to your topic
        rospy.set_param("kalman/initialY", pose.pose.position.y)
        rospy.set_param("kalman/initialZ", pose.pose.position.z)
        update_params(["kalman/initialX", "kalman/initialY", "kalman/initialZ"])

        rospy.set_param("kalman/resetEstimation", 1)
        update_params(["kalman/resetEstimation"])
        firstTransform = False

    else:
        msg.header.frame_id = pose.header.frame_id # Change pose according to your topic
        msg.header.stamp = pose.header.stamp
        msg.header.seq += 1
        msg.point.x = pose.pose.position.x
        msg.point.y = pose.pose.position.y
        msg.point.z = pose.pose.position.z
        pub.publish(msg)
```

3. High-Level-Script:

- תכניתו ניווט שתפקידן לשלוח פעולות תנועה בסיסיות לרחפן כגון: המראה, נחיתה ותזוזה ממקום למקום.
- בנינו מספר תכניות ניווט בסיסיות לשם הדגמה של פעולת המערכת (דוגמאות מצורפות בהמשך): תזוזה משולשית של רחפן יחיד, תזוזה במלבן של שני רחפנים, ומעקב של רחפן יחיד אחר עצם מזוהה. בדוגמאות המוצגות מטה, ניתן לראות שימוש בפעולות takeoff, land, goTo המדוברות לשם מימוש של תזוזה משולשית ומעקב אחר עצם:

```
def test_one_rect(cf, param = 0.3):

        cf.takeoff(targetHeight = param ,duration = 2.0)
        rospy.sleep(2.0)

        cf.goTo(goal = [-param/2, 0.0, 0.0], yaw=0.0, duration = 2.0, relative = True)
        rospy.sleep(2.0)

        for i in range(3):
                cf.goTo(goal = [-2*param, param, 0.0], yaw=0.0, duration = 4.0, relative = True)
                rospy.sleep(4.0)

                cf.goTo(goal = [0.0, -2*param, 0.0], yaw=0.0, duration = 4.0, relative = True)
                rospy.sleep(6.0)

                cf.goTo(goal = [2*param, param, 0.0], yaw=0.0, duration = 4.0, relative = True)
                rospy.sleep(4.0)

        cf.goTo(goal = [param/2, 0.0, 0.0], yaw=0.0, duration = 2.0, relative = True)
        rospy.sleep(2.0)

        cf.land(targetHeight = 0.05, duration = 3.0)
```

```
def onNewTransform(pose):
        global cfR
        global counter
        global ref_rate
        global ref_samples
        green_pose = [pose.pose.position.x , pose.pose.position.y , pose.pose.position.z]
        print(green_pose)
        counter+= 1
        if(counter == ref_samples):
                cfR.land(targetHeight = 0.0, duration = 2.0)
                rospy.sleep(3.0)
                cfR.stop()
        elif(counter < ref_samples):
                cfR.goTo(goal = green_pose, yaw=0.0, duration = ref_rate, relative = False)
        rospy.sleep(ref_rate)

if __name__ == '__main__':
        rospy.init_node('test_high_level')

        cfR = crazyflie.Crazyflie("crazyflieR", "crazyflieR")
        cfR.setParam("commander/enHighLevel", 1)

        counter = 0
        ref_rate = 1.0
        ref_samples = 30
        cfR.takeoff(targetHeight = 0.1 ,duration = 2.0)
        rospy.sleep(3.0)
        rospy.Subscriber("/ball_poseG", PoseStamped, onNewTransform , queue_size = 1)
        rospy.spin()
```

תוצרי הפרויקט

במסגרת העבודה על הפרויקט, הושגו היעדים הבאים כחלק ממימוש מטרת הפרויקט:

- "ריחוף" מיוצב – Stabilized Hovering – כלומר, בחירת מיקום במרחב, התייצבות של הרחפן בנקודה ושמירה על המיקום לאורך זמן.
- ניווט בין נקודות – Way-Point-Navigation – כלומר, מעבר והתייצבות בין נקודות במרחב המוגדרות מראש בתכנית.
- טיסה מתואמת של "להקה" (Swarm) של רחפנים.
- עקיבה רציפה אחר מטרה – דגימת מיקומה של מטרה כלשהי (כדור בצבע ייחודי) על ידי מצלמת העומק בקצב קבוע, והגדרתה כיעד עבור הרחפן.

סיכום, מסקנות והמלצות להמשך

בחינת תוצאות הפרויקט מול המטרות שהוגדרו מלכתחילה

- ניתן לומר, כי מטרות הפרויקט הושגו במלואן, ואף מומשו יכולות מעבר למטרות שתוכננו בראשיתו (כגון הרחבת מימוש הפרויקט עבור להקת רחפנים, ועקיבה רציפה אחר מטרה).

הצעות לשיפור המערכת בפעילות (פיתוח/מחקר) עתידית

על אף עבודה רבה על אופטימיזציות רבות, עדיין ניתן לשפר ולהרחיב את המערכת הקיימת על ידי:

- שימוש ב- Machine-Learning לשם איתור הרחפנים בתמונת הצבע, דבר שיסיר את הצורך בשימוש בסממן זיהוי (כדורים צבעוניים) עבור הרחפנים, ובכך יצמצם את מגבלות המערכת.
- הרחבת המערכת לשימוש במספר מצלמות עומק במקביל, בכך ניתן יהיה להשתמש במערכת בחללי ניווט גדולים יותר או בחללים בעלי מחיצות (חדרים, מחסנים וכו')
- אינטגרציה עם סנסורים למדידת מרחק על גבי הרחפנים לזיהוי מכשולים.
- כתיבת ובחינת אלגוריתמים מתקדמים לתנועה מורכבת במרחב ו/או אלגוריתמים לשיתוף פעולה בין קבוצת רחפנים.

רשימת מקורות

- Robot Operating System (ROS), The Complete Reference (Volume 2), Anis Koubaa, Springer, Volume 707
- Flying Multiple UAVs Using ROS, Wolfgang Honig and Nora Ayanian, Department of Computer Science, University of Southern California, Los Angeles, CA, USA, 2017