Department of Electrical Engineering
Electronics
Computers
Communications

Laboratory of ComputerGraphics & Multimedia
Department of Electrical Engineering Technion
all that meets the eye...

פרויקט א׳ – ספר פרויקט

Robot Vision Project

סטודנטים: אהוד יוסף גולדיק ושחר דרור

מנחים: חובב גזית ואביעד לויס

סמסטר: חורף תשע״ו

תוכן עניינים

1 הקדמה

1.1 מטרת הפרויקט

פרויקט זה הוא חלק מפרויקט פיתוח רובוט שמפותח בכמה צוותים ממעבדות שונות בפקולטה להנדסת חשמל בטכניון. מטרת הרובוט היא לשוטט ברחבה הראשית של בניין, לענות על שאלות ולתרום מידע לעוברי אורח. כך הרובוט מנגיש מידע באופן אוטומטי. הפרויקט שלנו הוא מימוש הראייה של הרובוט כך שיוכל להגיב לסביבה שלו ולתת תחושה של תגובתיות ומודעות.

הפרויקט מממש מערכת ראייה אוטומטית ותוצאותיו יוטמעו ברובוט. דרך הפעולה של הרובוט מתחילה בזיהוי אנשים אליהם הרובוט יגיב. שלב זה דורש מערכת ראייה פעילה באופן רציף ועל כן צריכה להיות מאוד יעילה. על מערכת זו לסרוק את רחבת הבניין ולעבד את תמונת המצב למציאת פנים המעוניינות לפנות לרובוט. לאורך השיחה, הרובוט צריך להתמקד בהתמדה בבן אדם אותו הוא משרת.

מטרתנו בפרויקט זה היא לכתוב תוכנה הממשת את מערכת הראייה שלו, כפי שמתוארת לעיל. מערכת זו תורכב מתוכנה שתחבר בין מספר רכיבי חומרה שונים: מצלמת אינטרנט ובקר זעיר השולט על מנועי תנועה. עיבוד התמונה אשר יתבצע בתוכנה, כולל בקרה מתמדת של מיקום הרובוט ביחס לפנים ותקשורת עם ה-microcontroller.

השלב הראשוני של הפרויקט הוא הגדרת מכונת המצבים (FSM) של מערכת הראייה ומימושו בתוכנה. בשלב זה תיכננו את שלבי הפעולה של הרובוט מתחילת הפעלתו ועד שהרובוט פונה אל בן אדם אחד. בתום שלב זה, התוכנה הצליחה לשלוט על המצלמה לבצע עיבוד תמונה ולפעול על פי התוצאות. העיבוד כולל זיהוי פנים בתמונה, חישוב הסטייה של הפנים ממוקד המצלמה בשני הצירים וחישוב המרחק בין הפנים למצלמה.

בשלב השני מימשנו את הקושחה (firmware) של מערכת הראייה על גבי Arduino. הגדרנו את דרכי התקשורת בין התוכנה לבקר ובין הבקר למנועים אשר שולטים על תנועת המצלמה. בשלב זה לא יכולנו להתבסס על חישובים בלבד אלא נדרשנו למשוב ולפעול בהתאם לכך.

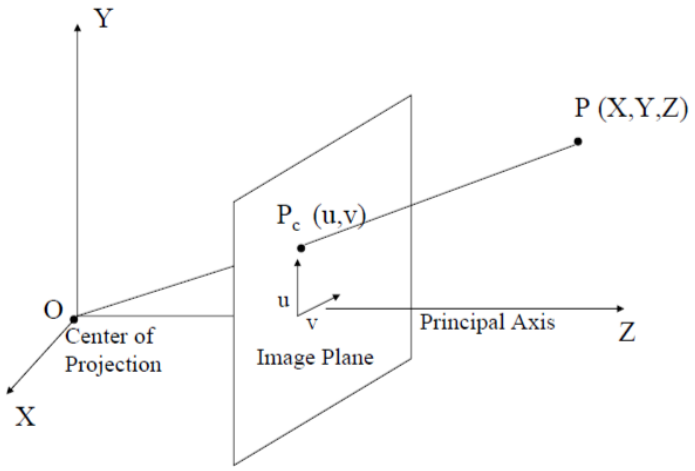
לאחר שהבנו את המערכת לפעולה בסיסית, ביצענו אופטימיזציה אמפירית לפי התרשמותנו מתגובת המערכת (באופן סובייקטיבי). במצב הסופי הרובוט עוקב בהצלחה אחר פנים של אדם העומד בשדה הראייה שלו.

2 רקע תיאורטי

מערכת הראייה של הרובוט הינה מערכת המבצעת זיהוי פנים ועיבוד תמונה לפקודות השולטות על המשך הפעולה של הרובוט. מערכת זו מורכבת ממצלמה קטנה וקלה ולכן בעלת מפתח מאוד מוגבל. לצורך עיבוד התמונה, אנו ממדלים את המצלמה כמצלמת חריר. מודל זה מתאים לכל המצלמות הפשוטות אך מכיוון שלמצלמה יש עדשה, נדרש תיקון אמפירי בכדי להתאים למודל.

2.1 מודל מצלמת חריר

נתאר את מודל המצלמה בעזרת הסימונים הבאים:



איור 2.1.1 - מודל מצלמת חריר

- $O-$ מרכז ההקרנה של המצלמה.
- $P_c(u,v)$ – פיקסל בתמונה.
- $P(X,Y,Z)-$ נקודה במרחב התלת-מימדי המוטלת על מישור התמונה בנקודה P_c.

התמונה נמצאת במישור XY כאשר ציר האופטי הינו ציר Z מטעמי סימטריה. התמונה במוקד המצלמה ונסמן את המרחק בין הנקודה O לתמונה ב- f. מרחק זה נקרא focal length.

המודל מתאר טרנספורמציה מנקודה במרחב התלת-מימדי לפיקסל בתמונה הדו-מימדית. כלומר, טרנספורמציה הממפה את הנקודה $P(X,Y,Z)$ לנקודה $P_c(u,v)$. המודל מניח שלמצלמה אין עדשה המרכזת קרני אור והחישובים הגיאומטריים מניחים שקואורדינטות התמונה רציפות. לכן, המודל אינו לוקח בחשבון את התופעות הנגזרות משימוש עדשה, כגון עיוותים בקצוות התמונה ושמצלמות פיסיקליות בעלות קואורדינטות בדידות. משמעות הדבר היא שהמודל משמש כקירוב מסדר ראשון לטרנספורמציה הרצויה.

כאשר ראשית הצירים של קואורדינטות התמונה נמצאת בנקודה $(X,Y,Z)=(0,0,Z)$, עבור כל $Z>0$, ניתן לחשב את קואורדינטות הנקודה P_c דרך דמיון משולשים:

$$\frac{f}{Z}=\frac{u}{X}=\frac{v}{Y}$$

בהינתן גודל קבוע $Z>0$ ניתן להסיק את הקשר:

$$\begin{pmatrix} u \\ v \end{pmatrix}=\frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}$$

לפיתוח מטריצת המצלמה (Camera Matrix) נרחיב את הנקודה P_c לקואורדינטה הומוגנית במרחב התלת-מימדי. הטרנספורמציה מציגה קשר לינארי בין P ו- P_c, כלומר $\underline{P_c}=\underline{\underline{\mathbf{C}}}\underline{P}$.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}=\mathbf{C}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, w\equiv 1$$

מתוך הדרישה $w\equiv 1$ נקבל:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}=\underbrace{\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{C}}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

כאשר ראשית הצירים של קואורדינטות התמונה אינה מיושרת עם ראשית הצירים של המרחב התלת-מימדי, יש צורך להוסיף היסטים שונים בציר u ובציר v של התמונה, אותם נסמן ב t_u ו- t_v בהתאמה. כעת מטריצת המצלמה תהיה:

$$\mathbf{C}=\begin{pmatrix} f & 0 & t_u \\ 0 & f & t_v \\ 0 & 0 & 1 \end{pmatrix}$$

כעת, הנקודות P_c,P ביחידות שונות ולכן קיים בכל ציר קבוע (אותם נסמן ב α,β) התלוי בפרמטרים פנימיים של המצלמה, כגון מפתח ורזולוציית המצלמה. במידה וקיימים עיוותים נוספים בתמונה, ייתכן כי ציר u אינו מאונך לחלוטין לציר v ולכן פיקסל בציר u בעל רכיב בציר v הנקרה skew. את הקשר המלא ניתן לתאר על ידי:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}=\begin{pmatrix} \alpha f & s & \alpha t_u \\ 0 & \beta f & \beta t_v \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

2.2 כיול

מטרת תהליך הכיול היא לחשב את מטריצת המצלמה $\mathbf{C}$, מתוכה להסיק את מוקד המצלמה (focal length) ואת ההיסט של התמונה (t_u, t_v). למטרה זו נעזרנו בספרייה נפרדת של python, calibrateCamera, המחשבת את המטריצה $\mathbf{C}$ מאוסף תמונות עם אובייקט מוכר ומוגדר מראש. לפרויקט השתמשנו באובייקט של לוח שח בגודל 7×10 ריבועים.

תהליך הכיול מתבצע עם תחילת התוכנה. סקריפט שכתבנו מצלם תמונות של לוח השח וקורא לפונקציה calibrateCamera. הסקריפט מחלץ את מוקד המצלמה ואת מרכז התמונה ושומר את הפרמטרים להמשך הריצה של מערכת הראייה של הרובוט. מוקד המצלמה משמש לחישוב מרחק של פיקסל מהרובוט וההיסט קובע את מרכז התמונה האמתית, כלומר נקודת הרפרנס של מערכת הראייה.

2.3 אלגוריתם זיהוי פנים

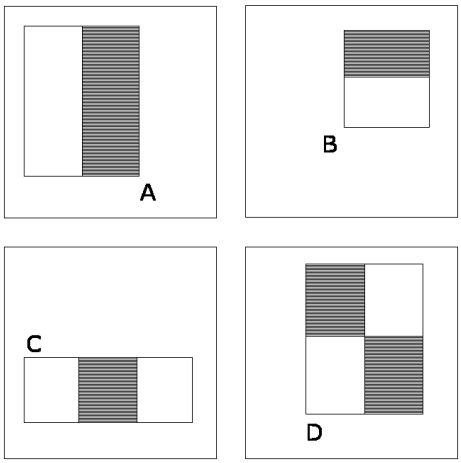
השלב הראשון ב-FSM של מערכת הראייה הינו לזהות אנשים פוטנציאלים לאינטראקציה. תהליך זיהוי הפנים הינו החלק היסודי של שלב זה. הזיהוי נעשה באמצעות אלגוריתם שהציעו Paul Viola ו-Michael Jones בשנת 2001 [1].

תהליך הזיהוי פועל על תמונות בגוון אפור. עיקרי התהליך כוללים את השלבים הבאים:

1. זיהוי פרמטרים דומיננטיים של הפנים. תכונות אלו מכונות Haar Features. דוגמה: גשר האף יותר בהיר מאזור העיניים.
2. קביעת תבניות למיון מאפייני הפנים ובניית מערכת לומדת.
3. דירוג התבניות על פי ביצועי המערכת הלומדת.

כאמור, נתוני המערכת הן תמונות פנים דו-ממדיות כאשר כל פיקסל עם ערך המייצג גוון אפור. את התבניות, המוגדרות על ידי מאפיינים חזותיים של הפנים (Haar Features), מחלצים מתוך קובץ עזר מסוג xml, haarcascade_frontalface_alt_tree.

התבניות מוגדרות על בסיס המחלקה הבאה של תבניות, כאשר כל תבנית יכולה להיות ווריאציה של תבנית במחלקה. למשל רוטציה, הגדלה, הקטנה, תשליל של התבנית וכו'.



איור 2.3.1 - תבניות לגילוי מאפיינים חזותיים

לכל תבנית אנו מבצעים את החישוב הבא עם התמונה; מוצאים את ההפרש בין סכום הפיקסלים בתמונה שמיקומם באזורים הכהים של התבנית, לבין סכום הפיקסלים בתמונה הממוקמים באזורים הבהירים של התבנית. את ההפרש הנ"ל נסמן באמצעות x. לקבלת אוסף כללי החלטה H מסווגים כל איטרציה של התהליך על ידי ערך סף כלשהו, a:

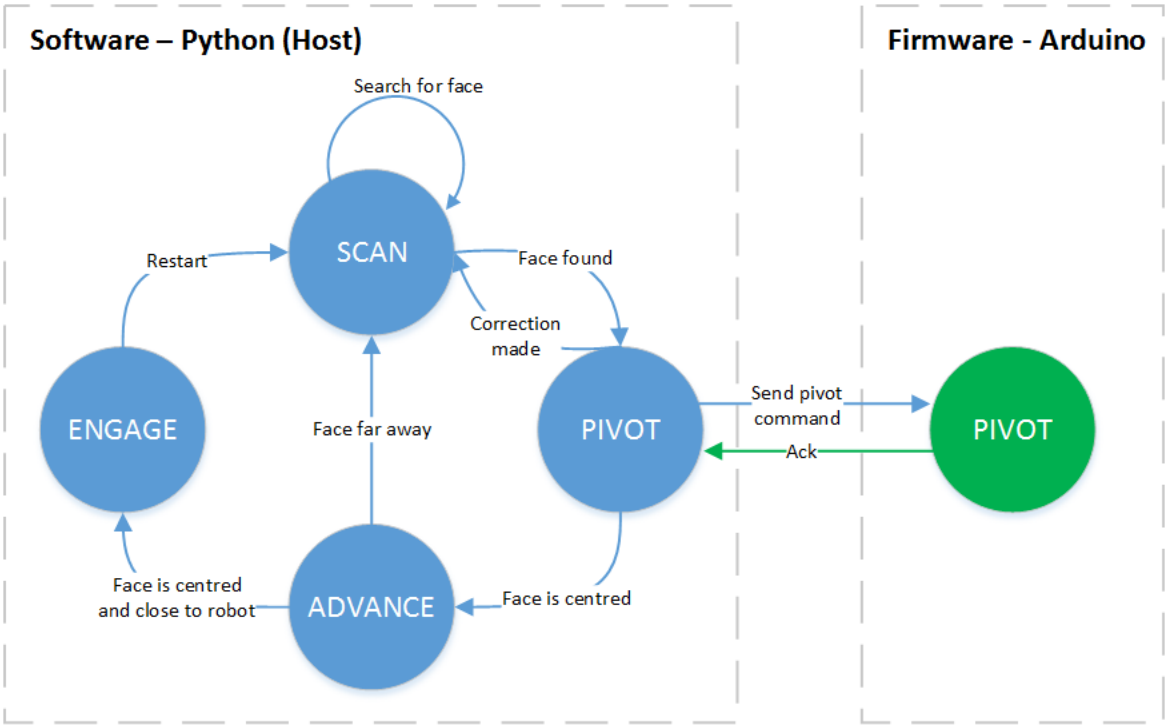
$$x = \sum pixels\ in\ black\ area - \sum pixels\ in\ white\ area$$

$$H = \left\{ x \mapsto \begin{cases} 1, & x > a \\ 0, & x \leq a \end{cases} \quad \mathbb{R} \right.$$

הערך של H הוא אינדיקטור לקיום האובייקט (הפנים) בתמונה ומיקומו. לאחר גיבוש אוסף גדול וסופי של כללי החלטה, מובטח לנו ש- H הוא $PAC\ learnable$ ולכן נוכל להפעיל את תהליך הלמידה על האוסף H. מטרת תהליך הלמידה, הפועל בשיטת Adaboost, היא לבחור קבוצה קטנה של מאפיינים חזותיים, קריטיים לזיהוי מוצלח של פנים, מתוך האוסף המקורי הגדול. האוסף המצומצם של תבניות מהווה כלל החלטה טוב המצביע על מיקום פנים בתמונה באופן יעיל. השלב האחרון באלגוריתם של Viola & Jones הוא מיון נוסף לבניות שהתקבלו על מנת להסיר השפעות של רקע התמונה. שלב זה נועד להתמודד עם תופעת false-positive ולהבטיח שרוב הכוח החישובי יתמקד במציאת מאפיינים חזותיים באזור הפנים. שלב זה גם משתמש בעקרונות הלמידה למציאת כללי החלטה אופטימליים.

3 מימוש

3.1 FINITE STATE MACHINE



איור 3.1.1 - דיאגרמת מצבים

לצורך מימוש הרובוט, הגדרנו מכונת מצבים סופית בכדי לנהל את זרימת המעקב אחר פנים.
המצבים הינם:

1. SCAN – סריקה
2. PIVOT – סיבוב
3. ADVANCE – התקדמות
4. ENGAGE – הפעלה

3.1.1 SCAN:

במצב זה מתבצעת הגישה למצלמת האינטרנט לצורך לכידת תמונה.

לאחר לכידת התמונה אנו מפעילים על המסגרת שצולמה את אלגוריתם זיהוי הפנים כמתואר בפרק הרקע התיאורטי.

לצורך הפעלת האלגוריתם אנו מעבירים את התמונה לייצוג שחור לבן כנדרש על ידי אלגוריתם viola Jones כפי שממומש בספריית opencv.

את אלגוריתם זה אנחנו מפעילים באמצעות קריאה לפונקציית Cascade.detectMultiScale מספריית opencv אשר מחזיר לנו מערך של מלבנים החוסמים בהם אזורים בתמונה אשר זוהו כפנים.

הקריאה לזיהוי הפנים מתבצעת בלולאה אשר תנאי העצירה בה הוא מציאת התאמה כלשהי, כלומר, כל עוד לא נמצאה התאמת פנים על ידי האלגוריתם בגודל מינימלי אשר מפורט במהלך הקריאה, מצב זה יישאר בלולאה סגורה ולא נתקדם בזרימת מכונת המצבים.

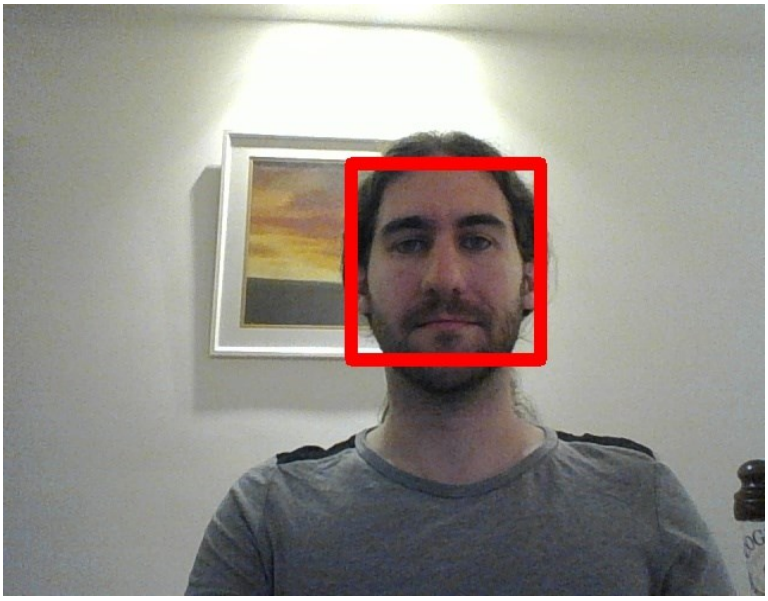
```
while articles is ():
        ##capture frame
        s, frame = cam.read()

        ##convert to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        #detecting faces and pouring to candidate list
        articles = Cascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=2,
            minSize=(60, 60),
            flags = cv2.CASCADE_SCALE_IMAGE
        )
```

לאחר זיהוי התאמה אחת או יותר לפנים, אנו בוחרים מרשימת הפנים שנמצאו את הדומיננטי על פי גודל (שטח המלבן הגדול ביותר החוסם את הפנים).

את מלבן זה אנחנו מאפיינים על פי ארבעה מספרים:

1. X- אינדקס אופקי של פיקסל הפינה השמאלית התחתונה של המלבן.
2. Y- אינדקס אנכי של פיקסל הפינה השמאלית התחתונה של המלבן.
3. W- רוחב המלבן (בפיקסלים)
4. H- גובה המלבן (בפיקסלים)

את ארבעת המספרים האלה אנחנו מעבירים כערך החזרה מפונקציית המימוש של המצב.



איור 3.1.1.1 - מלבן זיהוי פנים כפלט מאלגוריתם הזיהוי (מסומן על פני התמונה המקורית, לפני המעבר לשחור-לבן)

המצב הבא:

בכל מקרה בו יוצאים ממצב SCAN, המצב הבא הוא PIVOT, בכדי לוודא מרכוז הפנים אשר נמצאו (כאמור נצא ממצב SCAN רק במידה ואכן נמצאו פנים).

3.1.2 PIVOT:

במצב זה אנחנו בודקים היסט של מרכז המלבן שהתקבל משלב הסריקה ממרכז התמונה בשני הצירים.

מרכז התמונה מסומן ב-`Cx, Cy` והוא מתקבל ממטריצת הכיול כמתואר בפרק הרקע התיאורטי.

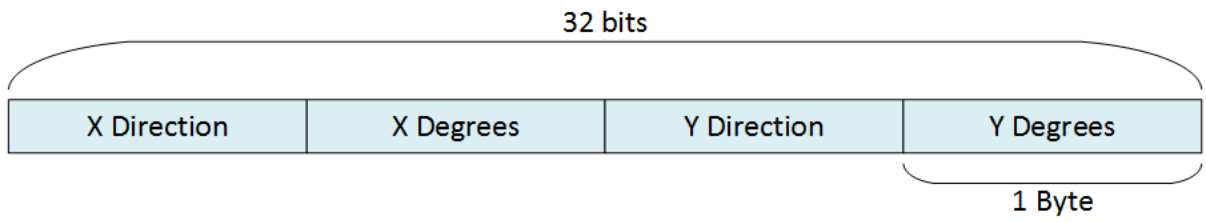
החישוב מתבצע באופן הבא:

```
correction_x = atan((mid_x - centre_x)/focal_length)

correction_y = atan((mid_y - centre_y)/focal_length)
```

עם זאת, באופן אמפירי, חילקנו את התיקון בכל ציר בפקטור של 1.45, זאת בכדי למנוע over-shoot, כך, התיקון מתבצע במספר צעדים קטנים, ואין צורך בתיקון "חזור".

במידה והתיקון באחד הצירים גדול מערך הסף אשר מתקבל כפרמטר לפונקציית המצב, נשלחת פקודת תיקון למיקרו-בקר, הממומש על גבי ה-Arduino.

החלטנו לממש את פקודת התיקון כמחרוזת יחידה באורך 32 סיביות, בכדי ששליחת פקודת התיקון תהיה אטומית.



איור 3.1.3.1 - מבנה פקודת תזוזה בשני צירים

ובאופן מפורש, אריזת הפקודה ושליחתה:

```
# command_str format = <x_dir><x_deg><y_dir><y_deg>
command_str = str(unichr(corr_x_dir)) + str(unichr(corr_x_deg)) + str(unichr(corr_y_dir)) +
str(unichr(corr_y_deg)) + ''

# send command_str to robot head
robot_head_serial.write(command_str)
```

לאחר שליחת הפקודה, הקוד ממתין באופן Busy-wait לתשובת המיקרו-בקר השולט בראש הרובוט.

המצב הבא:

1) אם היה נדרש תיקון- SCAN, זאת בכדי לאמת את מרכוז הפנים לאחר התיקון.
2) אם לא נדרש היה תיקון - ADVANCE.

3.1.3 ADVANCE:

מצב זה הוא למעשה הכנה לעתיד. המטרה היא שהרובוט יוכל להתקדם לכוון האובייקט לאחר שמרכז אותו בשדה הראייה.

לצורך כך, אנו משערכים את מרחקו של האובייקט מהרובוט באמצעות הנחת גודל ראש סטנדרטי.

המרחק במצב זה מוערך על ידי הנוסחה:

$$\text{distance} \approx \frac{\text{focal\_length} \cdot \text{AVG\_HEAD\_W}}{\text{detected\_face\_width}}$$

כאשר focal_length הוא תכונת המצלמה המתקבלת ממטריצת הכיול (ראה פרק 2.2 – כיול), AVG_HEAD_W הוא רוחב הראש הממוצע (1.52 סנטימטרים [2]) ו- detected_face_width הוא רוחב המלבן חוסם הפנים הדומיננטי שזוהה בתמונה בשלב ה-SCAN.

המצב הבא:

1) אם המרחק המשוערך גדול מערך הסף SCAN – INTERACTION_DIST.
2) אם קטן מערך הסף – ENGAGE.

3.1.4 ENGAGE:

מצב זה גם הוא הכנה לעתיד, ולמעשה אז הרובוט יגיב לאדם אתו החליט ליצור קשר, בצורה שתבחר.

במימוש הנוכחי מצב זה ריק.

המצב הבא:

המצב הבא הינו SCAN, למעשה בשלב זה "סיימנו" להגיב לאדם וחזרנו לסרוק את הסביבה אחר אדם חדש.

3.2 HOST CODE

זהו הסקריפט שרץ על פיתון ומממש את מכונת המצבים, למען פשטות תחזוקת הקוד והקריאות, הוא חולק למספר קבצים המממשים פונקציות.

```
STATE = SCAN
mode is  0
STATE = PIVOT
correction_x = 0.165535
correction_y = 0.015091
STATE = SCAN
mode is  0
STATE = PIVOT
correction_x = 0.162534
correction_y = 0.014474
STATE = SCAN
mode is  0
STATE = PIVOT
STAY THERE (XY - axis)!
STATE = ADVANCE
Distance is astimated at 0.948182
STATE = ENGAGE
```

איור 33.2.1 – פלט ריצה

3.2.1 robot_vision.py:

קובץ זה מהווה את ההיררכיה הגבוה ביותר של הקוד (נקודת הכניסה), ומטרתו לממש את הלולאה הרצה על מצבי המכונה, אך לא לפני שמבצעים כמה שיותר מההכנות הנדרשות לפני הריצה.

מכיוון שאנו מעוניינים בתגובה מהירה ככל הניתן על ידי הרובוט, רצינו להשאיר את הלולאה של הריצה כמה שיותר חסכונית, ולכן בתחילת הריצה אנו טוענים את קובץ מטריצת הכיול ומאתחלים את הגישה לבקר הארדואינו.

הלולאה עצמה מכילה רק switch case (ממומש בפיתון עם if/elif) כאשר כל מקרה מתאים לאחד המצבים ומתבצעת בו קריאה לפונקציה בודדת הממשת את אותו המצב, לאחר החזרה מפונקציות מימוש המצב, נקבע ערכו המייצג של המצב הבא (ראה 3.2.5).

3.2.2 rv_scan.py:

קובץ זה מאתחל את מכשיר הצילום וניגש לקובץ ה-xml שהוא ה-cascade בו נשתמש באלגוריתם זיהוי הפנים (ראה פרק 2.3 ברקע התיאורטי).

טעינות אלה ממומשות בתחילת הסקריפט ולא בתוך פונקציית מימוש המצב, בכדי שיקראו פעם אחת בתחילת הריצה (כאשר נקראת שורת הייבוא: from rv_scan import scan), ולא בכל איטרציה של הלולאה הקוראת למצב זה.

מלבד האתחולים, מוגדרת בקובץ זה פונקציית מימוש המצב SCAN.

```
scan(scan_mode, centre_x, centre_y)
```

פונקציה זו נועדה לבצע את זיהוי הפנים, על ידי צילום תמונה, ובאמצעות אלגוריתם ויולה ג'ונס וקובץ זיהוי הפנים, יוצר רשימה של כל האזורים התמונה שזוהו כפנים הפונות אל המצלמה.

כל אזור המזוהה כפנים מיוצג על ידי רביעיית ערכים (x, y, w, h), כאשר x, y הם אינדקס הפינה התחתונה השמאלית של הפנים בתמונה, ו-w ו- h הם רוחב וגובה הפנים בהתאמה.

צילום המסגרת וזיהוי הפנים מתבצע בלולאה עד שנמצאת התאמה אחת לפחות, זאת בכדי להבטיח שלא נצא משגרת הסריקה עד שנמצא מושא להמשך השלבים.

לאחר יצירת רשימת הפנים המוזכרת, אנחנו מסמנים את הפנים אשר מוגדרות להיות ה"דומיננטיות" (ראה 3.2.5).

פרמטרים:

scan_mode – שיטת העדפת הפנים הדומיננטיות במסגרת (מועבר כפרמטר ל-dom_square).

centre_x – אינדקס הפיקסל המרכזי בתמונה בציר האופקי

centre_y – אינדקס הפיקסל המרכזי בתמונה בציר האנכי

ערכי החזרה:

frame_w – רוחב התמונה בפיקסלים

face – רביעיית הערכים המייצגת את הפנים הדומיננטיות שנמצאו בתמונה (מובטח שנמצאו כאמור)

3.2.3 rv_pivot.py:

קובץ זה מממש את פונקציית המצב PIVOT.

```
pivot(robot_head_serial, frame_w, angle_horz, angle_vert, face, focal_length,
                                                        centre_x, centre_y)
```

מטרת פונקציה זו היא מציאת המיקום היחסי במסגרת של מרכז הפנים, ובמידה ומיקום זה לא עונה על קריטריוני המרכוז שנקבעו, מחשבת את שערוך התיקון בשני הצירים שנדרש ושולחת ערכים אלה לבקר הארדואינו.

פרמטרים:

robot_head_serial – החיבור הסדרתי דרכו תתבצע התקשורת עם בקר הארדואינו (במידה ויהיה צורך בתיקון)

frame_w – רוחב התמונה בפיקסלים

angle_horz – הזווית האופקית [רדיאנים] אשר מגדירה את ההיסט המרבי ממרכז התמונה אשר לא יצרוך תיקון על ידי סיבוב ראש הרובוט

angle_vert – הזווית האנכית [רדיאנים] המרבית שלא מצריכה תיקון

face – רביעיית בערכים המגדירה את הפרצוף שנמצא בשלב הסריקה

focal_length – תכונה גיאומטרית של המצלמה כפי שהתקבלה ממטריצת הכיול

centre_x – אינדקס הפיקסל המרכזי בתמונה בציר האופקי

centre_y – אינדקס הפיקסל המרכזי בתמונה בציר האנכי

ערך החזרה:

next_state – ערך המצב הבא, ADVANCE באם הפנים ממורכזות מלכתחילה; SCAN באם נדרש היה תיקון

3.2.4 rv_advance.py:

קובץ זה מכיל את פונקציית מימוש המצב ADVANCE.

```
advance(face, focal_length)
```

פונקציית חישוב המרחק מהאובייקט שזוהה.

פרמטרים:

face – רביעיית הערכים המגדירה את הפרצוף שנמצא בשלב הסריקה

focal_length – תכונה גיאומטרית של המצלמה כפי שהתקבלה ממטריצת הכיול

ערך החזרה:

next_state – ערך המצב הבא, ENGAGE באם האובייקט קרוב למצלמה; SCAN באם הפנים רחוקות מערך הסף שנקבע

3.2.5 rv_utilities.py:

קובץ זה מגדיר פונקציות ומשתני עזר לשימוש פונקציות המצבים.

```
def enum(**enums):
    return type('Enum', (), enums)

STATE = enum(SCAN=0, PIVOT=1, ADVANCE=2, ENGAGE=3)
```

זאת אינומרציית המצב לפיה עוברים בין המצבים השונים של פעילות קוד ה-HOST.

```
dom_square(articles, mode, centre_x, centre_y)
```

פונקציה זאת עוברת על רשימת הפנים שזוהו בתמונה ובוחרת את הדומיננטי ביניהם.

קריטריון הדומיננטיות הוא אחד משניים, הגדול ביותר או הממורכז ביותר.

פרמטרים:

`articles` – מערך רביעיות פנים כפי שהתקבלו מריצת אלגוריתם זיהוי הפנים על מסגרת אחת.

`mode` – קריטריון תעדוף הפנים הדומיננטיות. כאשר '0' – הרביעייה שתבחר היא זו המייצגת את ריבוע הפנים הגדול ביותר בשטחו; כאשר '1' – הרבעייה שתבחר היא זו שמרכזה קרוב ביותר למרכז התמונה.

`centre_x` – אינדקס הפיקסל המרכזי בתמונה בציר האופקי

`centre_y` – אינדקס הפיקסל המרכזי בתמונה בציר האנכי

ערך החזרה:

`face` – (x, y, w, h) רביעיית הערכים המייצגים את הפרצוף הדומיננטי בקרב רשימת הפנים שניתנו על פי קריטריון הדומיננטיות שנקבע.

```
mid_face(face)
```

פונקציה זאת מוצאת את אינדקס הפיקסל המרכזי בשני הצירים של הפנים שניתנו.

פרמטרים:

`face` – (x, y, w, h) רביעיית הערכים המייצגים את הפרצוף

ערך החזרה:

`x, y` – אנדקסי הפיקסל המרכזי של המלבן החוסם את הפנים בציר האופקי (x) ובציר האנכי (y)

3.3 FIRMWARE

הראש הרובוטי לא מקבל את הפקודות ישירות מקוד ה-Host אלא נשלט על ידי בקר נפרד הממומש על גבי בקר זעיר מסוג Arduino Nano אשר מתקשר הוא עם קוד ה-Host. בין בקר הArduino לhost code הגדרנו פרוטוקול תקשורת השולח הוראות תנועה לראש הרובוטי. ההתקשרות בין שני הרכיבים מתבצע בקצב של *9600bps* המבטיח זמן תגובה מינימאלי של החומרה ותגובה נכונה. כפי שמומחש באיור 3.1.2.1 הפרוטוקול, ה-host code שולח ארבעה תווים מסוג ASCII character (אורך כל תו הינו *1Byte*) המגדירים את הציר, כיוון ומספר המעלות שיש לנוע בכל ציר, כאשר הכיוון מוגדר

על ידי סיבית סימן. כל צעד של הרובוט מתקדם ב- $0.102\,{}^{deg}\!/_{step}$. את המספר חישבנו על פי מספר הצעדים, בכל ציר, שלוקח לראש הרובוטי להסתובב ב- 90°.

3.3.1 Definitions

```
#define VERT_DIR_PIN 7
#define VERT_STEP_PIN 9
#define HORZ_DIR_PIN 8
#define HORZ_STEP_PIN 10
#define UP true
#define DOWN false
#define LEFT true
#define RIGHT false
#define DEG_PER_STEP 0.102
```

ההגדרות בבקר ה-Arduino נותנות שם עם משמעות לפינים החשמליים ולכיווני ההתקדמות.

3.3.2 Setup

```
void setup()
```

שגרה סטנדרטית של בקר ה-Arduino הנקראת פעם אחת עם הפעלת הבקר. במימוש שלנו, בפונקציה זו אנו מקצים את הפינים בהם נשתמש ופותחים את הפורט הסדרתי בתדר $9600bps$.

3.3.3 Step

```
void step_vert(boolean dir,int deg)
```

```
void step_horz(boolean dir,int deg)
```

בבקר ה-Arduino ממומשות שתי פונקציות השולטות על ההתקדמות של הראש הרובוטי בציר האנכי והאופקי, בהתאמה. הקלט של כל פונקציה זה כיוון ההתקדמות ומספר המעלות שיש להסתובב באותו ציר. פלט הפונקציה זה אות חשמלי מתאים אל הפין האחראי לכיוון הסיבוב בנוסף לרצף פולסים, כמספר הצעדים, שיש לקדם את הראש הרובוטי, אל הפין האחראי לקדם את הראש בצעד אחד. את מספר הפולסים מחשבים על ידי הקשר האינטואיטיבי:

$$\#\text{steps} = \frac{\text{degree}}{\text{degree per step}}$$

בין שליחת כל פולס יש להשהות את הבקר על מנת להתחשב בזמן התקדמות האות במעגל וזמן תגובת הבקר. זמן זה נמדד אמפירית וערכו $850\mu\sec$.

3.3.4 Loop

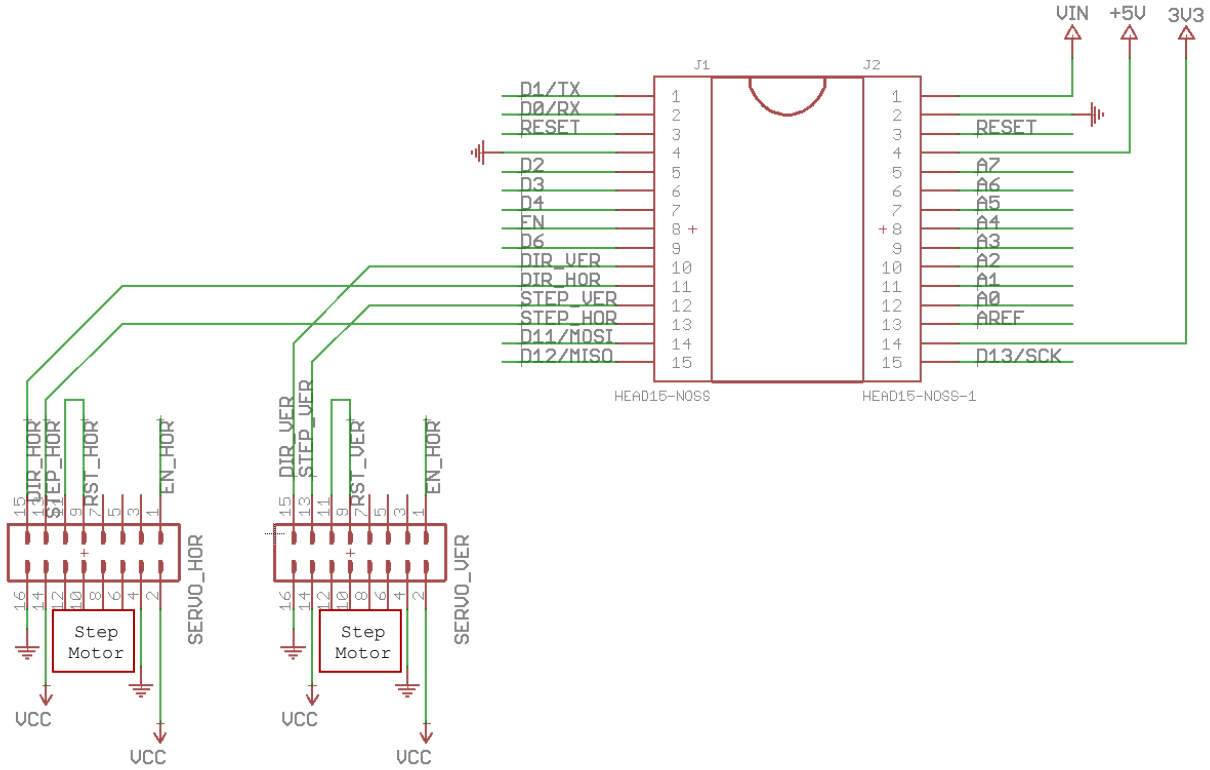
```
void loop()
```

זו השגרה הסטנדרטית העיקרית של בקר ה-Arduino. פונקציה זו נקראת בלולאה עד כיבוי הבקר. במימוש שלנו בפונקציה זו אנו ממתינים לשורת הפקודה מקוד ה-host (ראה פרק 3.1.2), מפענחים

אותה וקוראים לפונקציות step בהתאם. לאחר הצעד בכל ציר, ה-Arduino שולח מחרוזת חזרה אל ה-host המתארת את הצעדים שננקטו ושמשמת כחיווי ל-host שהבקר סיים את הצעד.

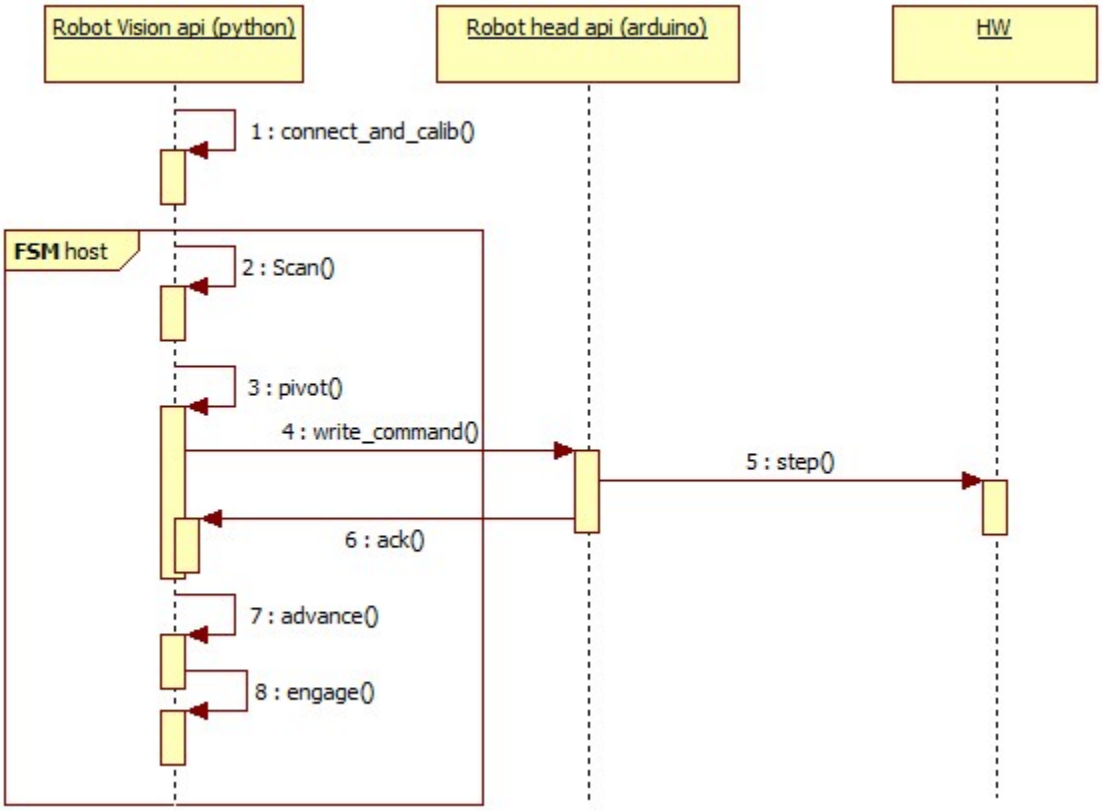
3.4 HARDWARE

בין ה-Arduino לראש הרובוטי מחוברים שני A4988 DMOS Microstepping Driver, אחד לכל ציר תנועה של הראש הרובוטי. את השרטוט הסכמתי של המעגל החשמלי ניתן לראות בתרשים 3.4.1. הדרייברים משמשים לשליטה על הצעדים של הראש הרובוטי. פלט בקר ה-Arduino, פולס בודד, מתורגם על ידי הדרייבר כך שהראש הרובוטי מתקדם בצעד אחד. פלט הדרייברים מועבר דרך חיבור סריאלי אל הראש הרובוטי מסוג Flir PTU. הראש מורכב משני מנועים, מנוע ראשון אחראי על סבסוב המצלמה ומנוע שני אחראי על עלרוד המצלמה. המצלמה מותקנת על גבי מנוע העלרוד.



איור 3.4.1 - תרשים סכמתי של המעגל החשמלי המחבר בין ה-Arduino, דרייברים והמנועים

תמונה 3.4.2 – ראש רובוטי עם מצלמה וספק מתח



איור 3.1 - תרשים רצף (Sequence Diagram)

4 תוצאות והסקת מסקנות (סיכום)

במצב הסופי, הרובוט עוקב בהצלחה אחר פנים של שני כותבי מסמך זה (ובכך נראה שמצליח לזהות פנים של בן ובת, עם ובלי זקן ועם ובלי משקפיים).

כמו כן הרובוט הציג זיהוי ומעקב מוצלחים ממרחק של כחמישה מטרים.

עם זאת, בשלב תיקון הזווית בהזזת ראש, הרובוט הציג בזווית קטנות (שמתאימות לאובייקט מרוחק) overshoot, שגרם לקושי בהתבייתות על האובייקט. הדבר התבטא במספר תנועות הלוך ושוב עד הגעה לשיווי משקל.

לצורך תיקון התופעה, הקטנו את הצעדים שהרובוט מבצע בכל תיקון זווית על ידי קבוע ריסון, ובכך מחד מזערנו תנועות גדולות והרובוט מבצע לעיתים מספר צעדים כאשר יכול היה לבצע צעד אחד, אך מאידך התנועה תמיד מתכנסת מיד והרובוט לא עובר את האובייקט.

אתגר מרכזי במהלך העבודה עלה מהדרישה שלנו לתגובה מידית ככל האפשר. כלומר שראש הרובוט יגיב בתזוזה בזמן קצר אחרי תחילת תנועת האובייקט. דבר זה מקבל משנה חשיבות כאשר האובייקט הוא מהיר, אז יש חשש שלולא הרובוט יתחיל לזוז בהקדם, האובייקט יצא מטווח הראייה של הרובוט ובכך "ייעלם" מעולמו של הרובוט לפני שהשני מהשניים יספיק להגיב. לצורך כך נקטנו במספר צעדים.

הצעד המרכזי היה להוציא כמה שיותר פעולות אל מחוץ ללולאת הפעילות הראשית של הרובוט, ובכך להבטיח שכל איטרציה בצד ה-Host מתרחשת במהירות האפשרית. בפרט, מזערנו גישות לדיסק, ובהן קריאת קובץ ה-haar_cascade ומטריצת הכיול אל תוך משתנים, ואתחול הגישה למצלמה ואל בקר הארדואינו בתחילת הריצה.

כמו כן, בצענו מספר בדיקות תגובתיות עם רזולוציות צילום שונות, כאשר ככל שהגדלנו את הרזולוציה, כצפוי הצילום מאט, אך יש צורך בפחות איטרציות מכיוון שהזיהוי מוצלח יותר.

לאחר ביצוע שלבים אלה, צוואר הבקבוק בהיבט הביצועים הוא מהירות תגובת מנועי הרובוט, אותו לא הצלחנו למזער.

נושא נוסף אשר עלה במהלך בניית אלגוריתם הריצה הוא קריטריון העדפת פרצוף דומיננטי.

הגדרנו בראשית כי הפנים הדומיננטיות יבחרו על-פי האדם הקרוב ביותר למצלמה (המלבן חוסם בעל השטח הגדול ביותר). עם זאת לאחר מכן הוחלט לבחון את האפשרות של קריטריון נוסף והוא מרכוז הפנים.

החלטנו שבסריקה הראשונית הרובוט יעדיף את הפרצוף הקרוב ביותר, אך לאחר זיהוי ראשוני וסיבוב לכוונו יעדיף את הפנים הנמצאות במרכז, זאת כדי למנוע שאחרי סיבוב יעבור הרובוט אל אובייקט חדש שעכשיו נכנס לשדה הראייה שלו גם אם קרוב יותר.

עם זאת, לאחר מימוש המצב ניכר שהדבר דווקא מוסיף רעש מכיוון שכאשר אובייקט זז לאחר תיקון ראשון, הרובוט העדיף אדם שעמד מאחוריו על פניו גם אם היה רחוק יותר משמעותית.

התוצאה הייתה רובוט בעל התנהגות "אפאתית", ובאופן סובייקטיבי העדפנו את התנהגות הרובוט התגובתית יותר.

לפיכך מצב זה לא נמצא בשימוש, אך השארנו אותו בקוד, כך אם וכאשר המערכת תותקן ברובוט שאכן מסוגל להתקדם, יוכל לנצל מצב סריקה זה, ואף נוספים (התשתית למצבי סריקה שונים עודנה קיימת).

בנוסף המצלמה שנמצאת בשימוש כרגע היא מצלמת אינטרנט ביעודה שנועדה לאובייקט ממורכז אחד, ולכן בעלת שדה ראיה מצומצם של 60° (על פי אתר היצרן). לא מן הנמנע שלו ישתנה השימוש למצלמה בעלת שדה ראייה גדול יותר תעלה דרישה לקריטריון תעדוף נוסף, למשל לתת העדפה לאזור מרכזי בתמונה, אך שפנים באזור אחר של התמונה מעל גודל סף מסוים יגברו על פרצוף קטן מהאזור המועדף.

5 הצעות להמשך

הפרויקט כפי שמומש עד כה מתעסק בתהליך זיהוי הפנים, מיקום האובייקט במרחב הראייה של הרובוט והזזת ראש הרובוט בשני צירים בכדי להביט ישר אל פני האובייקט, עם זאת, עד כה הרובוט עצמו הוא נייח.

המשך טבעי לפרויקט יהיה למקם את הרובוט על עגלה שתאפשר לו להתקדם לכוון האובייקט על פי המרחק שמחושב בשלב ה-ADVANCE.

ייתכן ולצורך כך נרצה להעביר את קוד ה-Host אל פלטפורמה שונה, כדוגמת לינוקס, בכדי להריצה על מחשב זעיר (e.g. Raspberry Pi), אך ייתכן שהעברה זו למחשב חלש יותר בהשוואה תעלה שוב את קשיי הביצועים בהם נתקלנו.

6 הוראות התקנה

1. את הפרויקט ניתן להריץ מכל מחשב (נבדק על מערכת Windows) עם python החל מגרסה 2.7. נדרש להתקין ספריית OpenCV שניתן להתקין מהקישור:

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html#install-opencv-python-in-windows

2. להתקין את ספריית pyserial (הקובץ `setup.py`) מהקישור:

https://github.com/pyserial/pyserial

3. להפעלת pyserial יש להקליד את הפקודה הבאה בטרמינל:

```
python -m pip install pyserial
```

4. להרצת הכיול ניתן לעקוב אחר ההוראות שכתובות בקובץ `Readme.txt` בתקיית calibration.
5. בשלב זה יש לחבר את ה-Arduino למחשב ולעדכן את שורה 24 בקובץ `Robot_vision.py` לפורט הנכון אליו חובר ה-Arduino (כרגע מוגדר COM4).
6. בשלב זה ניתן להריץ את הקוד המרכזי באמצעות קריאה לקוד בטרמינל תחת תיקיית הפרויקט:

```
Python robot_vision.py
```

- במידה והקוד רץ על מחשב נייד ייתכן והוא ילכוד תמונות מהמצלמה הפנימית של המחשב ולא ממצלמת הרובוט. במקרה זה יש לשנות את אינדקס המצלמה בשורה 14 בקובץ `rv_scan.py` כאשר האינדקס הוא מספר שלם החל מאפס.

7 מקורות ספרותיים

[1] P. Viola, M. Jones, “Rapid object detection using a boosted cascade of simple features.”

[2] C. Avendano, R. O. Duda, “Estimation of a Spherical-Head Model from Anthropometry. ”