The Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

ZumoPi Obstacle Avoidance

פרויקט מס' 21-1-1-2322

דו"ח סיכום

מבצעים:

שם: עידו ארד ת.ז. 322641127

שם: מיתר אמיר ת.ז. 211871892

מנחים:

ארקדי רפלוביץ' אוניברסיטת ת"א

מקום ביצוע הפרויקט: אוניברסיטת ת"א

תוכן עניינים

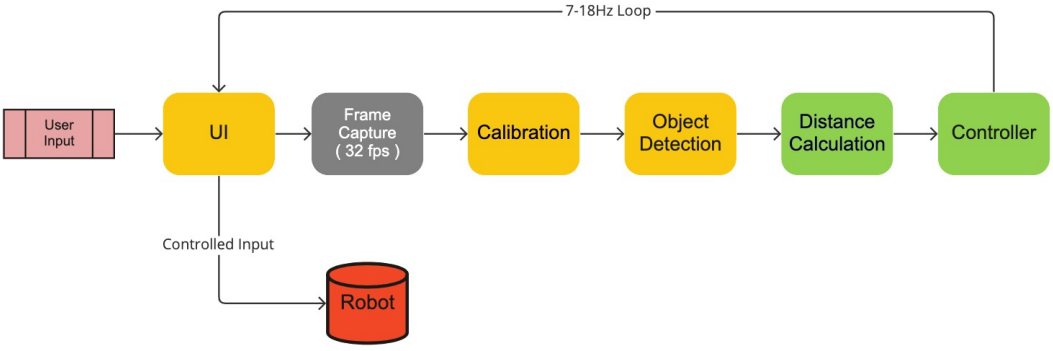
רשימת איורים

רשימת טבלאות

תקציר

במרכז הפרויקט עומד רובוט ממונע המבוסס על raspberry pi, הפועל בסביבת Linux . הרובוט מצוי בסביבה המכילה כדורי פלסטיק שמהווים מכשולים וניתן לשליטה מרחוק בעזרת התחברות לשרתי האוניברסיטה. בסביבת הפרויקט קיימת מצלמה הנמצאת על גבי הרובוט עצמו ומצלמת בזווית הראיה של הרובוט.

מטרת הפרויקט שלנו הינה יצירת ממשק ואלגוריתם המאפשר זיהוי עצמים והתחמקות אוטונומית מהם תוך שימוש במצלמה המחוברת לרובוט. התוצר הראשי של הפרויקט שלנו הוא כתיבה ויישום של אלגוריתם אשר נכתב אל תוך לוח הרובוט אשר יאפשר לו לבצע התחמקות אוטונומית ממכשולים הנמצאים בדרכו. בנוסף, מכיוון שכיום לא קיימת דרך פרקטית ויעילה להפעלת הרובוט, התוצר המשני בפרויקט הוא בניית User Interface – UI אשר מטרתו לשרת את המשתמש ברובוט ולאפשר לו ממשק עבודה נוח ואינטואיטיבי. כפי שניתן לראות מדיאגרמת הבלוקים הבאה, הפרויקט שלנו בנוי משני רבדים התואמים לשני התוצרים הסופיים, הראשי הוא מימוש ההתחמקות האוטונומית והמשני הוא יצירת ממשק מתאים.



איור 1 –דיאגרמת בלוקים עבור מערכת הפרויקט

1 הקדמה

הפרויקט שלנו עוסק במימוש התחמקות אוטונומית של רובוט המהווה חלק מסביבת למידה ZumoPi. מטרת סביבת עבודה זו בעתיד הינה לשמש סטודנטים במעבדות מתקדמות ומחקר. לכן, תפקידנו במטרה זו היא המשך פיתוח המערכת וממשק המשתמשים העתידיים, תוך מתן משוב למנחה הפרויקט על נקודות לשיפור המערכת. על כן, ניתן להגיד שהחשיבות של הפרויקט היא רבה משום שאנו, הסטודנטים של היום, עוזרים לשפר מערכת אשר תשמש סטודנטים עתידיים בשלל נושאים מגוונים.

כיום קיימות סביבות לימוד שונות המשמשות אוכלוסיות מגוונות של אנשים בלמידת תוכנה, התנסות בבקרה ורובוטיקה ועיבוד אותות. אלטרנטיבה מוכרת היא Lego NXT , אשר מאפשרת בניית רובוטים ותכנות שלהם בעזרת אבני בניין ותוכנה של לגו. מערכת זו פשוטה לתפעול ולא מאפשרת למידה מספיק עמודה כמצופה מסטודנטים בתארים אקדמאיים. אלטרנטיבה נוספת, אשר נעשה בה שימוש רחב

באוניברסיטה היא תוכנת MATLAB המאפשרת ביצוע סימולציות המלמדות את הנושאים שצוינו בתחילת הפסקה לעומק. החיסרון של תוכנה זו נובע מכך שלא חווים את המערכות שהיא מדמה באופן ממשי. לכן, לעיתים קרובות נוצר קושי בהבנה מצד הסטודנטים ונפגעת חווית הלמידה. לעומת זאת, מערכת ZumoPi מאפשרת למידה לעומק תוך התנסות מעשית בפועל.

דרך העבודה שלנו כללה את השלבים הבאים:

- כיול מצלמת הFish-eye הנמצאת על פני הרובוט.
- זיהוי המכשול (האובייקט).
- אומדן המרחק של הרובוט מהמכשול.
- כתיבת אלגוריתם שיחליט כיצד הרובוט מתנהג בהתאם לתנאי השטח, תוך הימנעות מהתנגשות במכשולים ודפנות המגרש.
- יצירת ממשק המקבץ את כל הפונקציות השונות ומאפשר שימוש נוח למשתמש.

לצורך זיהוי המכשולים, נדרשנו ללמוד כיצד נוכל לזהות אותם בעזרת המצלמה שעל הרובוט. היה עלינו לכתוב קוד אשר מקבל מטריצה של ערכים מספריים המייצגים את קלט המצלמה ויודע לזהות את האובייקט, לאמוד את המרחק ממנו ובהתאם לשנות את התנהגות הרובוט. ניתן לראות כי נדרש מאתנו ידע רחב בעיבוד תמונה.

אחרי שפתרנו את בעיית עיבוד התמונה, היה עלינו לנתח את אפשרויות התנועה של הרובוט בהתאם לפידבק המצלמה, המשתנה בכל רגע נתון בהתאם לתנאי השטח המשתנים. את הידע הנדרש לפרויקט השגנו מהאינטרנט ובנוסף מקורסים שנלמדו בנושאי בקרה ועיבוד אותות.

את הרקע התיאורטי שרכשנו, מימשנו בקוד שנכתב בשפת PYTHON אשר יישום על גבי מערכת ההפעלה מסוג Linux של רובוט ה – ZumoPi . המצלמה אשר ממוקמת על גבי הרובוט היא בזווית למישור התנועה ומסוג Fish-eye . לכן תחילה על הקוד לכייל את צירי התמונה בהתאם לצירי העולם האמיתי. לאחר מכן, כפי שצוין קודם, רצינו שהקוד ינתח את הקלט שהתקבל מהמצלמה ויזהה היכן נמצאים המכשולים ביחס לרובוט. לבסוף, בעזרת מימוש חוג בקרה מתאים, הרובוט יתנהג בהתאם למכשולים המוצבים בפניו תוך הימנעות מהתנגשות בהם. כל הפונקציות השונות שצוינו לעיל יקובצו יחד תחת UI שייכתב על ידינו.
כל תוצרי הפרויקט מרוכזים ב -GitHub מיועד לפרויקט [12].

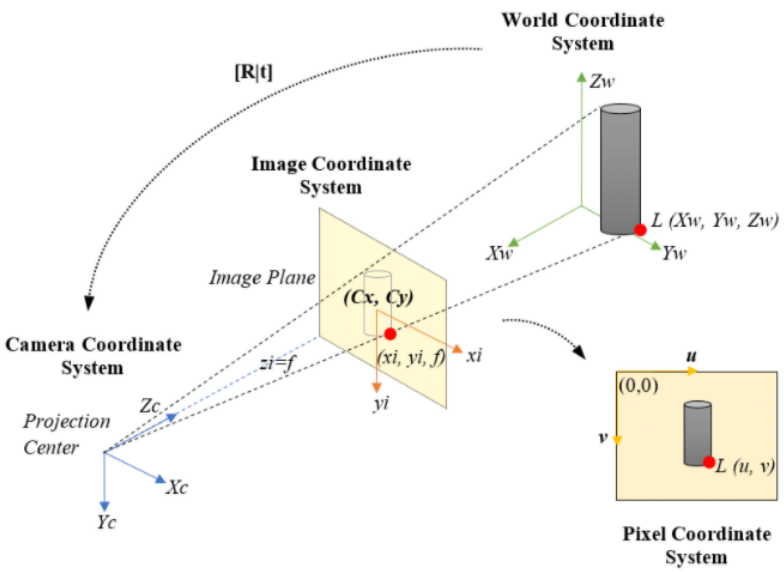
2 רקע תיאורטי

בהתבסס על איור 1 - "דיאגרמת הבלוקים עבור מערכת הפרויקט", נתאר את הרקע התיאורטי עבור הבלוקים המרכיבים את המערכת.

כיול מצלמה

על מנת לבצע התחמקות ממכשולים, נדרשנו לנתח את סביבת הרובוט על ידי שימוש במצלמה המחוברת עליו. המצלמה הינה מסוג fish-eye, משמע העדשה בעלת אורך מוקד קצר במיוחד היוצר עיוות משמעותי של התמונה. כדי לאפשר את זיהוי המכשול העומד בפני הרובוט, ובעקבותיו לכתוב קוד אשר יבצע בקרה על תנועת הרובוט, כיילנו ראשית את המצלמה כך שהתקבלה תמונה ללא עיוות עדשת המצלמה והקוד קלט תמונה כמה שיותר קרובה למציאות. את קוד כיול המצלמה כתבנו בפייתון על ידי שימוש בOpenCV – ספריה המכילה פונקציות שונות ביניהם פונקציות הקשורות לעיבוד תמונה.

הכיול הינו תהליך של מציאת הפרמטרים החיצוניים והפנימיים של המצלמה במטרה לקבל את הקשר שבין העולם האמיתי 3D לבין התמונה 2D . מודל "Pinhole" מתאר מצלמה על ידי חור במיקום מסוים ואורך מוקד (המרחק בין החור למישור התמונה). בעזרת שרטוט סכמתי של המודל, נסביר את התיאוריה מאחורי מציאת הפרמטרים הרצויים לכיול המצלמה.



איור 2 – שרטוט סכמתי של מודל Pinhole [1]

מערכת קואורדינטות של העולם האמיתי (X_w, Y_w, Z_w) מתורגמת למערכת קואורדינטות המצלמה (X_c, Y_c, Z_c) על ידי שימוש בפרמטרים החיצוניים של המצלמה. בעזרת הפרמטרים הפנימיים נוכל לקבל את המעבר בין (X_c, Y_c, Z_c) לבין מערכת הצירים ביחידות פיקסלים הומוגניות (u, v) .

הפרמטרים החיצוניים של המצלמה [2] באים לידי ביטוי בשיקוף נקודה מהעולם האמיתי לנקודה על מישור התמונה המתואר על ידי תהליך של סיבוב המתבטא בתור $R - Rotation\ matrix$ בגודל 3x3 ותהליך של תרגום, כלומר שינוי המיקום המתבטא בתור $t - Translation\ vector$ בגודל 3x1 .

בנוסף, קיימים גם מקדמי עיוות :

עיוות רדיאלי k_n – נובע מכך שיותר קרני אור העוברות דרך העדשה פוגעות בקצוות שלה מאשר במרכזה, מה שגורם לעיוות. במקרה שלנו, מדובר בעיוות מסוג "barrel" משום שזו מצלמת fish-eye, כלומר מדובר עיוות רדיאלי חיובי.

עיוות משיקי (tangential distortion) p_n – נובע מכך שמישור התמונה (חיישן המצלמה) לא מקביל לעדשת המצלמה, ולכן ישנם אזורים בתמונה הנראים צרים יותר ממה שהם באמת.

נשים לב כי ככל שהעיוות גדול יותר, כך נצטרך ליותר מקדמים על מנת לייצג אותו. OpenCV עובד עם עד 6 מקדמי עיוות רדיאליים ועד 2 מקדמי עיוות משיקיים.

הפרמטרים הפנימיים של המצלמה באים לידי ביטוי על ידי מטריצה התמונה K הכוללת את הרכיבים הבאים:

$$(1)\ K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

באשר:

- $f_x, f_y [pixels]$ – אורכי המוקד, המרחק בין המרכז האופטי (של עדשת המצלמה) לבין מישור התמונה. נשים לב כי ככל שאורך המוקד גדול יותר, כך זווית הראיה של המצלמה קטנה יותר. בקירוב זהים משום שהפיקסל הוא ריבוע ולכן אין עיוות בין הצירים.
- $c_x, c_y [pixels]$ – מרכז מישור התמונה.
- γ – עיוות בין הצירים x ו- y של חיישן המצלמה. כפי שהסברנו אין עיוות ולכן יהיה שווה ל-0.

בהתבסס על המודל, נייצג כעת בצורה מתמטית את הקשר שבין העולם האמיתי 3D לבין התמונה 2D.הנקודה במרחב (X_w, Y_w, Z_w) באה לידי ביטוי בקואורדינטות המצלמה (X_c, Y_c, Z_c) באופן הבא:

$$(2)\ \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + t = [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

עבור הקרנה למישור התמונה נשתמש במטריצת הפרמטרים הפנימיים ונקבל:

$$(3)\ \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

סך הכל הטרנספורמציה הכוללת תהיה:

$$(4)\quad s\begin{bmatrix}u\\v\\1\end{bmatrix}=K[R|t]\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$
$$=K[r_1 \quad r_2 \quad r_3 \quad t]\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

כאשר s הוא scaling factor ו- r_i, t הם וקטורי עמודות מטריצה $[R|t]$ בגודל 3x1 .

עבור המעבר למערכת צירים ביחידות פיקסלים הומוגניות השתמשנו בתכונות גיאומטריות וביחס צלעות במשולשים דומים, חילקנו בגודל בציר z שהינו ידוע ושווה לאורך המוקד $z_i = f$ וכך מצאנו את הקשר הבא:

$$(5)\quad u=\frac{x_i}{f}, v=\frac{y_i}{f}$$

המעבר למערכת צירים הומוגנית (u,v) נעשה על מנת שנוכל לייצג מעבר בין העולם האמיתי לבין מישור הפיקסלים.

נקבל באופן כללי את המעבר הבא מ 3D ל – 2D :

$$(6)\quad \begin{gathered}\begin{bmatrix}x\\y\\z\end{bmatrix}=R\begin{bmatrix}X_w\\Y_w\\Z_w\end{bmatrix}+t\\ x'=\frac{x}{z}, y'=\frac{y}{z}\\ u=f_x\cdot x'+c_x\\ v=f_y\cdot y'+c_y\end{gathered}$$

עלינו כעת להוסיף למודל מקדמי עיוות מכיוון שאנו לא במצב אידאלי בניגוד למודל "pinhole" :

$$(7)\quad \begin{gathered}x''=x'\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+2p_1x'y'+p_2\\ y''=y'\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+p_1(r^2+2y'^2)+2p_2x'y'\\ where\ r^2=x'^2+y'^2\\ u=f_x\cdot x''+c_x\\ v=f_y\cdot y''+c_y\end{gathered}$$

המטרה שלנו כעת היא לבטל את העיוות הנוצר, כלומר לכייל את המצלמה. נעשה זאת על ידי פתירת מערכת משוואות[1] שבסופה נמצא את הפרמטרים החיצוניים והפנימיים של המצלמה.

[1] ראה המשך בפרק מימוש – תיאור תוכנה : כיול מצלמה, עמוד 15

זיהוי אובייקט

בכדי שנוכל להתחמק מהמכשולים, עלינו קודם כל לזהות אותם ולנסות למצוא היכן הם ממוקמים במרחב. צורת כל המכשולים זהה וידועה מראש, כדור, כאשר התכונה המבדילה ביניהם היא צבע הכדורים ולכן זיהוי הצבע שלהם מהווה מרכיב עיקרי בהבדלה בין האובייקטים. חשוב להזכיר כי לרובוט יש רק מצלמה אחת, משמע הוא רואה את העולם בצורה דו ממדית בלבד.

על מנת להעריך את מיקומם של האובייקטים במרחב התלת ממדי, הצלבנו בין פרמטרים של האובייקטים והרובוט אשר הינם ידועים מראש לבין פרמטרים שהתקבלו מהתמונה הדו ממדית עבור אותם אובייקטים.

בתמונות דיגיטליות, בניגוד לעולם האמיתי, הצבע והבהירות של אובייקט הם משתנים בדידים וקיימות מספר שיטות כדי לייצג אותם. שתי השיטות המוכרות ביותר הן RGB ו-HSV:

RGB – מייצגת כל פיקסל כסופרפוזיציה של שלושה צבעים שונים: אדום, ירוק וכחול. עוצמת הבהירות והצבע של כל פיקסל יקבעו לפי עוצמת הבהירות של כל אחד משלושת הצבעים שצוינו לעיל כאשר טווח עוצמות הבהירות שלהם הוא [0,255].

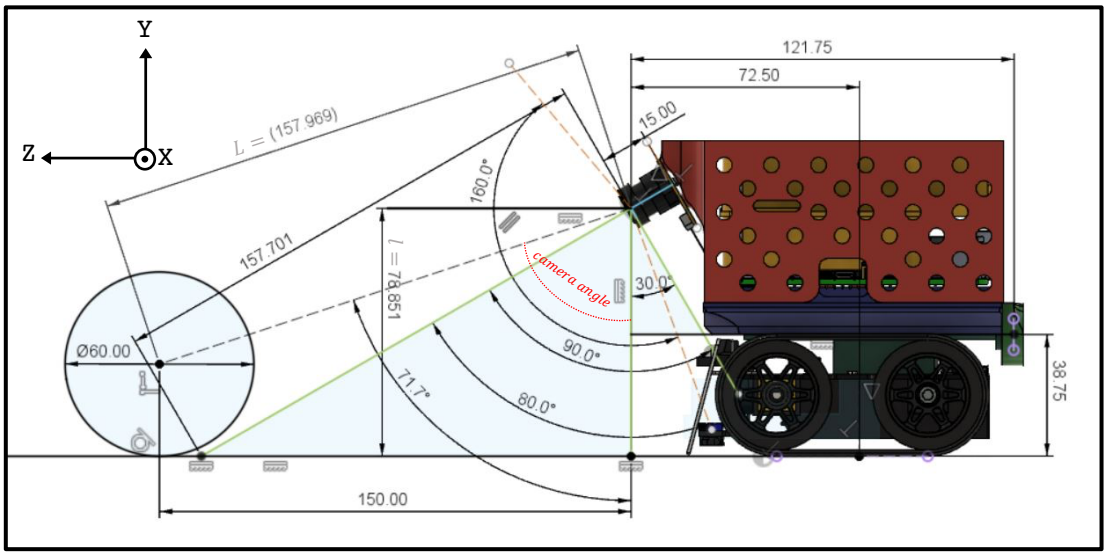
HSV – מייצגת כל פיקסל לפי שלוש תכונות: גוון, רוויה ובהירות. הגוון קובע את צבע הפיקסל ומקבל ערכים בטווח [0,179]. תכונת הרוויה קובעת את הצבעוניות של הפיקסל. ככל שערכה יותר נמוך כך צבע הפיקסל יותר דהוי והיא נקבעת לפי טווח ערכים של [0,255]. תכונת הבהירות קובעת את רמת הבהירות של הפיקסל. ככל שערכה נמוך יותר כך הפיקסל כהה יותר כאשר ב-0 הפיקסל יהיה שחור. טווח הערכים של תכונת הבהירות הוא [0,255].

כפי שציינו קודם, בפרויקט שלנו ההבדל היחיד בין האובייקטים הוא הצבע. לכן, בחרנו להשתמש בשיטת HSV משום שבאמצעותה יכולנו לנתח את צבע האובייקט בעזרת ערוץ אחד בלבד וקיבלנו דרך אינטואיטיבית ופרקטית יותר לתחום את ערכי הפיקסלים המתקבלים עבור כל אובייקט.

המטרה הסופית של חלק זה היא זיהוי האובייקט משמע מציאת המרכז שלו, במקרה שלנו מציאת מרכז הכדור, וחישוב הרדיוס שלו בפיקסלים על ידי האלגוריתם. לאחר מציאת הנתונים הללו, נוכל לעבור לחלק הבא שהוא אומדן המרחק של הרובוט מהאובייקט.

אומדן מרחק מאובייקט

לאחר מציאת מרכז האובייקט, רדיוס הכדור בפיקסלים (כפי שנמצא לפי האלגוריתם) ובידיעת רדיוס הכדור האמיתי 3[cm] קבוע , נוכל למצוא את המרחק של האובייקט מהרובוט. בעזרת הסכימה הבאה המתארת עבור מערכת הרובוט שלנו זוויות וגדלים רלוונטיים ביניהם גובה המצלמה מפני השטח, נוכל לבצע חישוב מקורב ככל הניתן של המרחק.



איור 3 – זוויות ומרחקים רלוונטיים של מערכת הרובוט

הצירים הוגדרו בהתאם לאיור למעלה והמיקומים חושבו בהתאם לנוסחאות הבאות על פי גיאומטריה של המערכת:

$$(8)\ Z\ [cm] = L \cdot sin\left(camera\ angle_{[deg]}\right) \quad where \quad L_{[cm]} = \frac{3_{[cm]} \cdot focal\ length_{[pixels]}}{Radius_{[pixels]}}$$

$$(9)\ \Delta X[cm] = 3_{[cm]} \cdot \left(\frac{object\ center_{x,[pixels]} - 0.5 \cdot\ frame\ width_{[pixels]}}{Radius_{[pixels]}}\right)$$

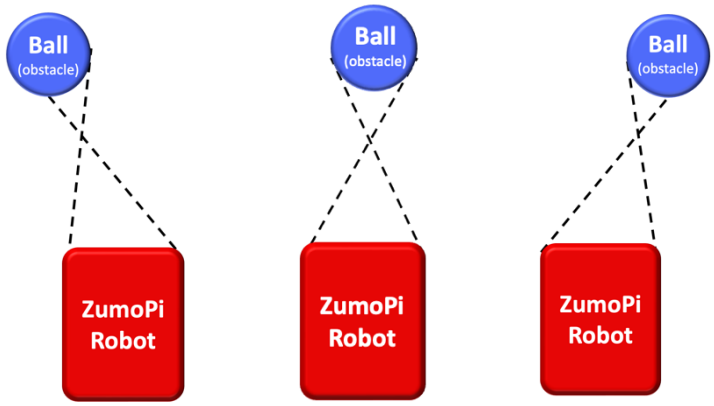
מתקבל מהנוסחאות מרכז הכדור אשר יתואר על ידי הנקודה (x_{Ball}, z_{Ball})

נדגיש כי הזווית $camera\ angle_{[deg]}$ משתנה עבור כל אובייקט ולכן היא תחושב על פי היחס בין הגובה של המצלמה מפני השטח $l_{[cm]} = 7.8851$ לבין הגודל L אשר משתנה עבור כל אובייקט

$$(10)\ camera\ angle_{[deg]} = \cos^{-1}\left(\frac{l}{L}\right)$$

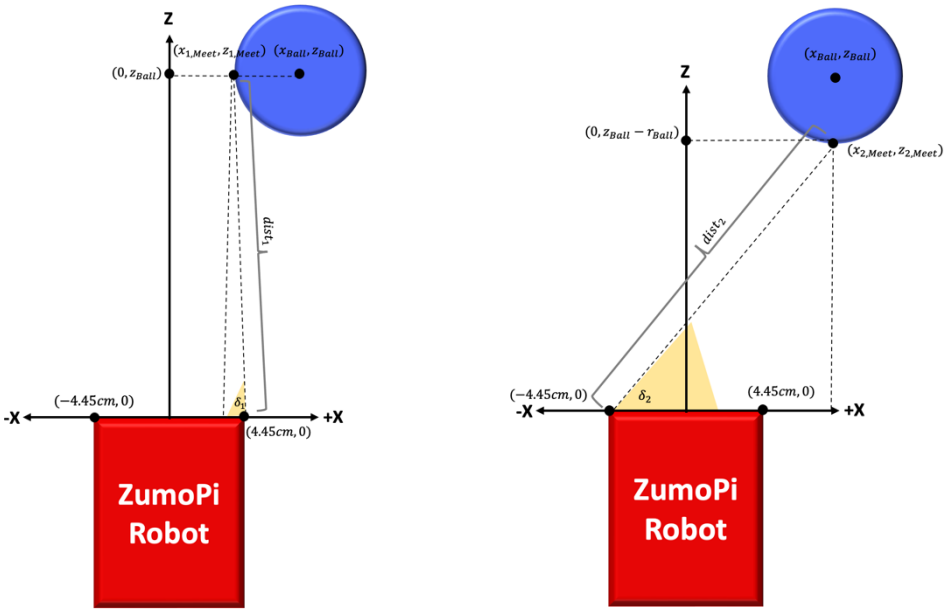
בקרת תנועת הרובוט

כפי שציינו, המטרה הראשית של הפרויקט היא התחמקות ממכשולים. כעת נסביר את הבקרה בה בחרנו להשתמש. הרובוט מופעל על ידי משתמש באמצעות ג'ויסטיק כחלק מ - UI שבנינו למערכת. המטרה שלנו היא לבנות אלגוריתם אשר יבצע התערבות בפעולת משתמש וימנע פגיעה וודאית של הרובוט במכשול שמולו. האלגוריתם יופעל על ידי המשתמש בלחיצת כפתור מובנה ב UI, ויתערב בכיוון התנועה של המשתמש בעת הצורך. כחלק מהנסיעה של הרובוט בסביבה מוגדרת, יתכנו 3 אפשרויות בהן האלגוריתם יאלץ להתערב:



איור 4 – מצבים אפשריים בהם המכשול מפריע לנסיעת הרובוט: ימין, מרכז ושמאל ביחס לרובוט בהתאמה.

הבקרה שלנו מבוססת על מציאת טווח זוויות אסור לנסיעה כדי שהרובוט ידע להתחמק מהמכשול. הטווח מוגדר בהתאם למיקום התיאורטי בו הרובוט היה מתנגש במכשול אילולא היינו מפעילים את האלגוריתם למניעת ההתנגשות. כפי שניתן לראות באיור למעלה, עבור כל מצב ישנם 2 מיקומים, משני צדי הכדור , בהם הרובוט יכול לפגוע. אנו חישבנו את 2 זוויות הפגיעה, משני הצדדים כדי לקבל את הטווח האסור. נסביר כעת את דרך החישוב בהתבסס על המצב בו המכשול מימין לרובוט, כאשר דרך הפעולה תקפה לכל המצבים הקיימים:



איור 5 – מציאת טווח זוויות אסור

- מערכת הצירים מוגדרת ביחס לרובוט, כאשר ראשית הצירים (0,0) מוגדרת להיות במרכז החלק הקדמי של שלו. כתוצאה מכך, נשתמש בכך שרוחב הרובוט הוא $8.9[cm]$ ולכן הנקודות המתארות את הקצוות של חלקו הקדמי הן $(\pm 4.45cm, 0)$.
- מרכז הכדור מתואר על ידי הנקודה (x_{Ball}, z_{Ball})[2] באשר הרדיוס שלו ידוע והוא $r_{Ball} = 3[cm]$.
- נקודת המפגש התיאורטית של הרובוט עם הכדור מתוארת על ידי (x_{Meet}, z_{Meet}).

נשים לב כי נקודות אלה נמצאות בקו ישר עם מרכז הכדור כך שהן מקיימות :

$$\begin{aligned}(x_{1,Meet}, z_{1,Meet}) &= (x_{Ball} - r_{Ball}, z_{Ball}) \\ (x_{2,Meet}, z_{2,Meet}) &= (x_{Ball}, z_{Ball} - r_{Ball})\end{aligned} \qquad (11)$$

- לאחר שנקבל את נקודות המפגש, נוכל למצוא את המרחק בין כל נקודה לנקודת הקצה של הרובוט הרלוונטית לה.
- מתוך שיקולים גיאומטריים במשולש ישר זווית, נמצא את הזוויות $\delta_{1,2}$:

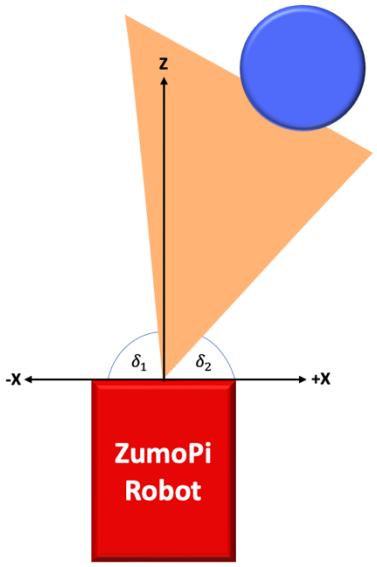
$$(12)\ \delta_i = sin^{-1}\left(\frac{z_{i,Meet}}{dist_i}\right)$$

סך הכל נקבל כי הטווח האסור מוגדר כך ש:

- תיאסר נסיעה שמאלה (ביחס לציר Z) עבור $angle_{user} > \delta_1$.
- תיאסר נסיעה ימינה (ביחס לציר Z) עבור $angle_{user} > \delta_2$.

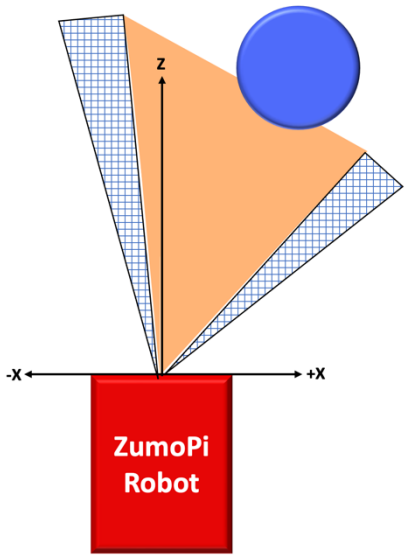
בלומר, הרובוט לא יוכל לנוע בטווח הבא:

[2] מרכז הכדור התקבל בחלק אומדן מרחק כדור מאובייקט, עמוד 10



איור 6 – הטווח האסור

בפועל, קיימת שגיאה מסוימת בחישוב המיקום של הכדור אשר עלולה להביא לשגיאה בטווח האסור שחושב.. לכן, לצורך מניעת ההתנגשות בכדור, הגדלנו את הטווח בהתאם לפונקציית משקל שנקבעת ביחס הפוך למרחק מהכדור. לאחר שהוגדר סופית הטווח האסור, נוכל לבצע את הבקרה על הרובוט ולמנוע התנגשות במכשול[3].

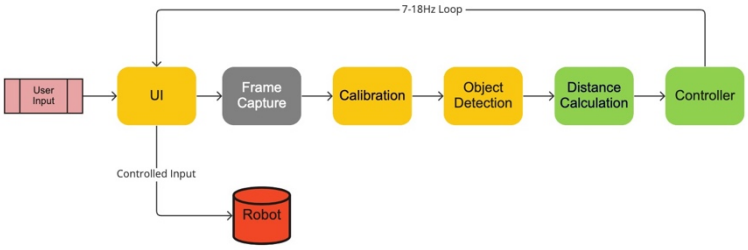


איור 7 – הטווח האסור בפועל

[3] ראה המשך בפרק מימוש – תיאור תוכנה : בקרת תנועת הרובוט, עמוד 23

3 מימוש

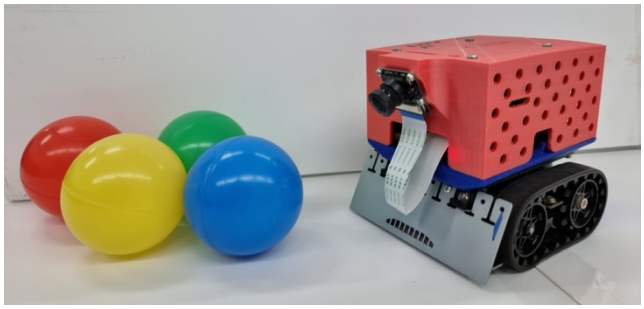
נציג בשנית את איור 1 – "דיאגרמת בלוקים עבור מערכת הפרויקט":



נתאר בהתאם את מימוש החמרה הכולל את הרובוט, מרכיביו והמכשולים במערכת ואת מימוש התוכנה התואם לכיוון התקדמות הדיאגרמה וכולל בהתאמה את כיול המצלמה, זיהוי האובייקט, אומדן מרחק מהרובוט, בקרת תנועת הרובוט ולבסוף ממשק המשתמש.

3.1 תיאור חמרה

הרובוט העומד במרכז הפרויקט נבנה על ידי מנחה הפרויקט. הוא בנוי מ Raspberry Pi 4 Model B [4] הנמצא בתוך מעטפת שהודפסה בעזרת מדפסת תלת ממד. על גבי הרובוט ממוקמת מצלמה מסוג Fish-Eye בעלת 8 מגה פיקסלים וזווית ראייה של 160 מעלות (160 FOV) [5]. המכשולים מהם על הרובוט להתחמק הינם כדורי פלסטיק בעלי רדיוס של $3[cm]$ בצבעים : צהוב, אדום, כחול וירוק. הצבעים קבועים וידועים מראש.



איור 8 – סביבת הפרויקט

3.2 תיאור תוכנה

כיול מצלמה

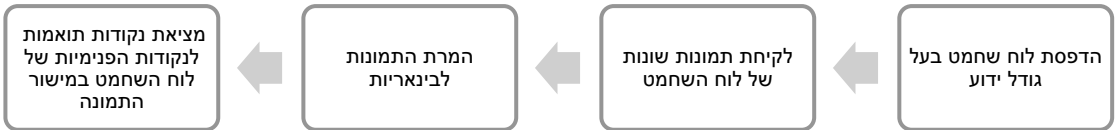
עלינו לבצע כיול של המצלמה היושבת על גבי הרובוט שלנו. את הקוד המבצע את הכיול נכתוב בסביבת פייתון. הכיול מתבצע על ידי שימוש בתמונות שצולמו בעזרת המצלמה הנתונה ברזולוציה של (768x1024). תהליך העבודה יורכב משלושת השלבים הבאים:

שלב 1
מציאת נקודות בעולם האמיתי והנקודות התואמות להן במישור התמונה.

שלב 2
מציאת הפרמטרים החיצוניים והפנימיים של המצלמה.

שלב 3
תיקון התמונה המעוותת המקורית וקבלת התמונה המתוקנת החדשה.

שלב 1



נתחיל בשימוש בפונקציית **findChessboardCorners** שמטרתה לבדוק האם קיים דפוס של לוח שחמט בתמונה נתונה ולהוציא כפלט את מערך מיקום הפינות הפנימיות של לוח זה במידה וקיים.

- הדפסנו לוח שחמט [3] אותו הורדנו מאתר OpenCV והוא בעל גודל ידוע. לוח שחמט הוא בעל צורה גיאומטרית מוגדרת וידועה ולכן קל יותר להשיג באמצעותו נקודות רצויות.
- הנקודות בהן נשתמש עבור הכיול מחולקות לשני סוגים: 3D ו- 2D .

נקודות 3D של העולם האמיתי יקראו object points והנקודות המתאימות להם בתמונה 2D יקראו image points.

- בעזרת מצלמת הרובוט, צילמנו את לוח השחמט עבור זוויות שונות ומיקומים שונים ביחס למצלמה. זאת על מנת שנוכל לתעד בצורה טובה ביותר את העיוות שיוצרת המצלמה.
- התמונות שצולמו יומרו לתמונות בינאריות ויוכנסו לפונקציה בתור קלט.
- הנקודות object points ו - image points הן פלט הפונקציה.

הפונקציה מקבלת תמונה אפורה עם 256 רמות אפור. היא עוברת על התמונה ומחשבת את ההיסטוגרמה שלה. לאחר מכן, הפונקציה מחשבת את גרדיאנט ההיסטוגרמה ונקודות המינימום שלו. בעזרת נקודות אלו היא מוצאת את הנקודה עבורה בקירוב מחצית מהפיקסלים יהיו בהירים יותר, אשר תקבע כנקודת ערך הסף. בעזרת נקודה זו ניתן לשמור בצורה אופטימלית על דפוס לוח השחמט בהמרת התמונה לבינארית. כעת ניתן לחלץ מתוך התמונה הבינארית את גבולות הצורות המינימליות השונות בתמונה.[4] נשים לב כי אנו מעוניינים בריבועי לוח השחמט בלבד ולכן הפונקציה בוחרת רק בצורות העונות על הדרישות הגיאומטריות עבור ריבוע (בוחנת תנאים על אורכי אלכסונים, צלעות ושטח הצורה) ומצמצמת אותן ל- 4 קודקודים, הם מהווים את הנקודות הפנימיות של לוח השחמט.

שלב 2

הפונקציה הבאה בה נעשה שימוש תהיה **calibrateCamera** . הפונקציה מקבלת כפלט

את object points ו - image points ומוצאת בעזרת הקשר ביניהם את הפרמטרים החיצוניים והפנימיים של המצלמה. הפונקציה מבוססת על מודל "pinhole" אותו תיארנו קודם.

[4] ראה הסבר על הפונקציה findcontours בפרק מימוש – תיאור תוכנה : זיהוי אובייקט ואומדן מרחק מהרובוט, עמוד 19

בעזרת המאמר [2] נוכל להסביר כיצד עובדת הפונקציה. נסתכל על נוסחה (4) ונקבע את מישור לוח השחמט ב - $Z = 0$ ולכן ניתן לרשום באופן הבא:

$$(13)\ s\begin{bmatrix}u\\v\\1\end{bmatrix} = K[r_1 \quad r_2 \quad r_3 \quad t]\begin{bmatrix}X\\Y\\0\\1\end{bmatrix} = K[r_1 \quad r_2 \quad t]\begin{bmatrix}X\\Y\\1\end{bmatrix}$$

נוכל לסמן באופן הבא

$$(14)\ H = [h_1, h_2, h_3] = K[r_1 \quad r_2 \quad t] \Longrightarrow s\begin{bmatrix}u\\v\\1\end{bmatrix} = H\begin{bmatrix}X\\Y\\1\end{bmatrix}$$

מהגדרת מטריצת הסיבוב R , אנו יודעים כי r_1, r_2, r_3 אורתונורמליים זה לזה, כלומר:

$$(15)\quad < r_i, r_j > = < K^{-1}h_i, K^{-1}h_j > = (K^{-1}h_i)^T\left(K^{-1}h_j\right) = h_i^T K^{-T} K^{-1} h_j = 0$$

$$(16)\quad < r_i, r_i > = < K^{-1}h_i, K^{-1}h_i > = (K^{-1}h_i)^T(K^{-1}h_i) = h_i^T K^{-T} K^{-1} h_i = 1$$

נסמן $B = K^{-T}K^{-1}$.

עבור כל תמונה שצולמה, יכנס כקלט לפונקציה מערך של הנקודות הפנימיות של לוח השחמט. עקב העובדה שמימדי הלוח ידועים, קביעת אחת מהנקודות כראשית הצירים בעולם האמיתי תקבע את מיקום שאר הנקודות בלוח. הצבה של הנקודות הללו יחד עם מיקומי הנקודות במישור הפיקסלים במשוואה (8), מייצרת מערכת משוואות אשר מאפשרת על ידי שימוש באלגוריתם Levenberg–Marquardt [5] לקרב את ערכה של מטריצה H עבור כל תמונה.

עבור n תמונות שצולמו, נקבל n מטריצות H שונות ונוכל להפיק מתוך משוואות (10), (9) סט של $2n$ משוואות. כך נוכל לקרב את ערכו של B, שוב בעזרת אלגוריתם Levenberg–Marquardt .

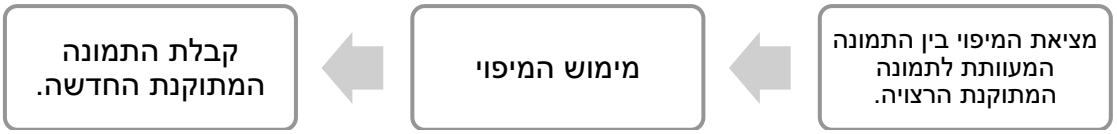
מתוך מטריצה B ניתן לחלץ את ערכי המטריצה הפנימית K , שהם הפרמטרים הפנימיים של המצלמה.

משם, נוכל לחלץ את ערכי הפרמטרים החיצוניים, באופן הבא:

$$(17)\qquad \begin{aligned} r_1 &= K^{-1}h_1 \\ r_2 &= K^{-1}h_2 \\ r_3 &= r_1 \times r_2 \\ t &= K^{-1}h_3 \end{aligned}$$

נשים לב שעד כה התעלמנו ממקדמי העיוות ולכן כעת נמצא אותם. נעשה זאת בדרך דומה, תוך שימוש בערכי K, R, t ובמשוואות (6) , (7) תוך שימוש חוזר באלגוריתם [3] Levenberg–Marquardt אשר יוודא מרחק אלגברי מינימלי בין ההקרנה לבין המיקום בפועל של הנקודה בעולם האמיתי.

שלב 3



נרצה לקבל תמונה מלאה מכוילת וללא אזורים שחורים. לכן עלינו לבצע מיפוי מהתמונה המעוותת לפני הכיול לתמונה הלא מעוותת. כלומר, השאיפה היא קבלת תמונה באותו גודל כמו התמונה המעוותת המקורית.

initUndistortRectifyMap היא פונקציה שמטרתה למצוא את טרנספורמציית המיפוי על מנת לבטל את עיוות התמונה ולתקן אותה. עבור כל פיקסל בתמונה המתוקנת החדשה (u, v), הפונקציה תמצא את הקואורדינטה המתאימה בתמונה המקורית המעוותת שצולמה על ידי המצלמה.

נשים לב שקשר זה הוסבר בתיאוריה שהצגנו כך שהמיפוי המתקבל הוא :

$$(18)\ map_x(u, v) = x''f_x + c_x$$

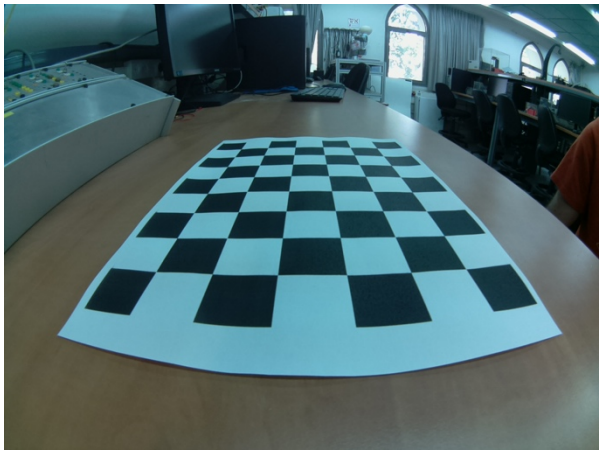
$$(19)\ map_y(u, v) = y''f_y + c_y$$

נשתמש בפונקציית **remap** שמטרתה לממש את המיפוי שנמצא. המיפוי מתרחש בשיטה של אינטרפולציה בליניארית. עלינו למצוא עבור כל פיקסל בתמונה החדשה את הערך שלו לפי התמונה המעוותת המקורית. המיפוי לאו דווקא יפנה אותנו לפיקסל מדויק ופה תבוא לידי ביטוי האינטרפולציה. ערך הקואורדינטה אליה מפנה המיפוי בתמונה המקורית, יחושב עם אינטרפולציה בליניארית ולפי 2x2 ערכי הפיקסל הקרובים אליה. בשיטה זו יילקח הממוצע המשוקלל של ארבעת הפיקסלים כדי להגיע לערך הסופי של האינטרפולציה. משקל כל פיקסל ידוע מבוסס על מרחק הקואורדינטה ממנו.

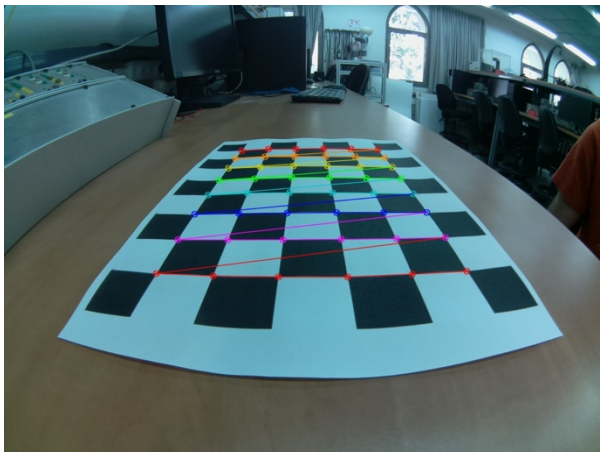
לבסוף, עקב האופי הקמור של התמונה המקורית, כיולה גרם לעיוות בקצותיה משום שהיא נפרסה והפכה לתמונה ישרה. לכן, היה עלינו לחתוך את החלק המעוות שנותר על מנת לקבל את התמונה המכוילת הסופית.

מעתה ואילך, מכיוון שמטריצת הכיול ומקדמי העיוות כבר חושבו והם קבועים עבור אותה מצלמה, עבור כל תמונה שנרצה לכייל, נחזור אך ורק על שלב 3.

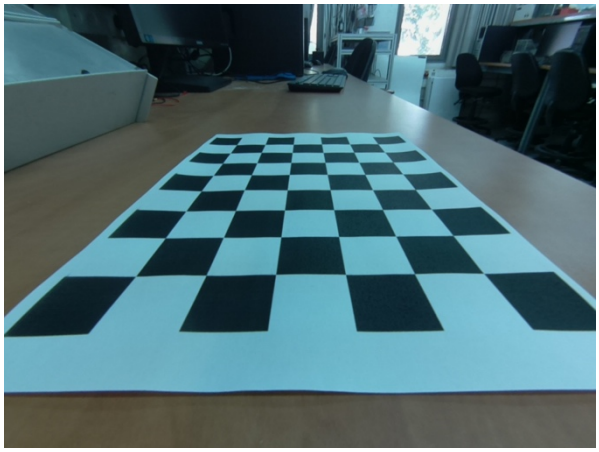
לסיכום, נראה דוגמא של הרצת תהליך הכיול על אחת מהתמונות של לוח השחמט שצולמו. התקבלה תמונה מכוילת ברזולוציה זהה לתמונה המקורית (768x1024), כאשר חלק מהמידע של התמונה המקורית (קצוות התמונה) נחתך.



התמונה המקורית והמעוותת



התמונה המקורית והמעוותת
בתוספת סימון הנקודות
הפנימיות של לוח השחמט אשר
זוהו על ידי הקוד



התמונה החדשה והמתוקנת

איורים 9-11 - תהליך הכיול

זיהוי אובייקט ואומדן מרחק מהרובוט

בחלק זיהוי האובייקט, הרובוט מחזיק בתמונה המכוילת והוא מעוניין לזהות את האובייקטים הנמצאים בה ולהעריך את מיקומם. הוא מבצע זאת לפי סדרת השלבים הבאה כאשר השלבים חוזרים על עצמם עבור כל אחד מהאובייקטים (4 חזרות):

שלב 1
מציאת כל הפיקסלים שעלולים להיות שייכים לאובייקט וניקוי רעשים.

שלב 2
מציאת הגבולות של כל האובייקטים, חישוב השטח שלהם וסינון האובייקטים הלא רצויים.

שלב 3
תחימת האובייקט תחת עיגול מינימלי וזיהוי מיקומו של האובייקט בעולם האמיתי ביחס לרובוט.

שלב 1

יצירת תמונה בינארית לפי ערך סף תוחם ידוע מראש ← ניקוי רעשים מהתמונה הבינארית

לפני בניית הקוד לזיהוי האובייקט לקחנו מספר גדול יחסית של תמונות המכילות את האובייקטים השונים והרצנו קטע קוד שאפשר לנו באופן ידני לתחום את ערכי הHSV ששייכים לאובייקטים השונים. ביצוע של התהליך על מספר גדול של תמונות אפשר לנו למצוא עבור כל אובייקט תחום מינימלי שמכיל את רוב ערכי הHSV האפשריים של האובייקט בתנאי תאורה שונים. כעת, נוכל להשתמש בטווח שמצאנו על מנת לייצר תמונה בינארית כך שכל הפיקסלים שחשודים להיות חלק מהאובייקט יקבלו את הערך 1.

הטבלה הבאה מדגימה את הטווחים עבור ערכי HSV עבור האובייקטים השונים:

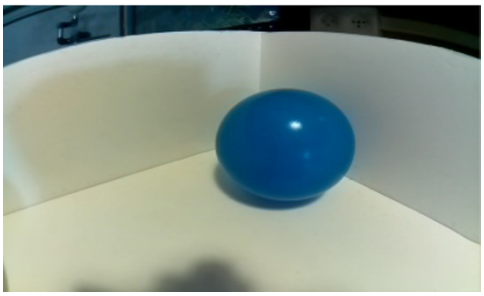
Object color	H value		S value		V value	
	min	max	min	max	min	max
red*	172	20	104	255	104	255
blue	95	113	229	255	62	219
green	64	92	158	255	73	204
yellow	23	42	100	255	97	227

טבלה 1 – טווח ערכי HSV עבור צבעי הכדורים השונים

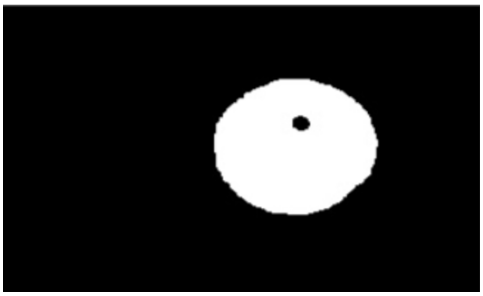
* ערך Hue מחזורי כל $180°$ ולכן הטווח מתחיל בערך 172 ונגמר בערך 20.

הטווח שמצאנו אינו אידאלי והתמונה הבינארית יכולה לכלול בתוכה בועיות קטנות בתוך האובייקט של פיקסלים שלא זוהו או לחילופין עלולים להיות חפצים שונים ברקע שעלולים להכיל בועיות קטנות של פיקסלים אשר נכללים בתוך הטווח שקיבלנו. על מנת לתקן בעיה זו, האלגוריתם עובר על התמונה הבינארית עם חלון ומשנה את ערך הפיקסל שבמרכזו לערך המינימלי שנמצא בחלון, כך שבועיות קטנות של הערך 1 ייעלמו.

לאחר מכן מריצים חלון דומה אלא שהפעם החלון משווה את ערך הפיקסל שבמרכזו לערך המקסימלי שנמצא בחלון, דבר אשר יעלים את הבועיות השחורות בתמונה. לאחר שהתמונה הפכה לבינארית, לא יהיה קיים הבדל בהתנהגות הקוד עבור צבעי הכדורים השונים.



לפני



אחרי

איורים 12-13 – יצירת התמונה הבינארית

שלב 2

מציאת הגבולות של כלל האובייקטים שנמצאו בתמונה הבינארית

מציאת השטח של כלל האובייקטים ושמירת האובייקט עם השטח הגדול ביותר

נתחיל בשימוש בפונקציית findContours שמטרתה למצוא את הגבולות של כלל האובייקטים בתמונה ולסווג אותם לפי היררכיית אב בן [4] .הפונקציה רצה על התמונה הבינארית ובכל מעבר מ0 ל-1 או מ1 ל-0 נקבע גבול חדש של אובייקט. כאשר נמצא גבול חדש הפונקציה רצה על הגבול ושומרת מערך של כל הנקודות אשר שייכות לו. הפונקציה גם שומרת את ההיררכיה של הגבולות, אם קיים אובייקט (נקרא לו אובייקט 1) שכל הנקודות של הגבול שלו מוכלות בתוך טווח הנקודות הסגור על ידי גבול של אובייקט אחר (נקרא לו אובייקט 2) אז אובייקט 1 יקרא הבן של אובייקט 2.

לאחר מעבר על כל התמונה הבינארית נקבל מערך של אובייקטים כאשר עבור כל אובייקט נדע את ערכי הנקודות של הגבול שלו, את המיקום שלו בהיררכית האובייקטים ומי האב והבן שלו במידה והם קיימים. לצורך ייעול הקוד המערך של נקודות הגבול של האובייקט נשמר כך שקווים ישרים בגבול מיוצגים על ידי הקצוות שלהם בלבד וכך גודל המערך קטן בצורה משמעותית. בנוסף, כיוון שכל האובייקטים שלנו מלאים ובצבע אחיד האובייקטים שאנחנו מחפשים בהכרח נמצאים בהיררכיה הגבוהה ביותר (אין להם אבות) ולכן ניתן להתעלם מכל אובייקט שלא נמצא בהיררכיה זו.

אין לנו צורך עוד בנתוני ההיררכיה של האובייקטים ולכן אנחנו משתמשים בפונקציה grab_contours אשר מסננת את נתוני ההיררכיה ומחזירה מערך של הגבולות של כל אובייקט בלבד.

על מנת למצוא את שטחם של האובייקטים השתמשנו באלגוריתם אשר מתבסס על משפט גרין הדיסקרטי [5] אשר מאפשר לקשר בין שטח של צורה להיקף שלה על ידי חיבור וחיסור של השטחים הנוצרים בין הנקודות השונות בהיקף לבין ראשית הצירים לפי הנוסחה הבא:

$$(20)\ \iint f(x,y)dxdy = \sum_{\bar{x}\in\Delta\cdot D} \alpha_D(\bar{x})\cdot F(\bar{x})$$

כאשר $\Delta\cdot D$ הוא ההיקף של הצורה ו-$\alpha_D(\bar{x})$ מקבל ערכים לפי הכיוון שאליו מתקדם ההיקף בנקודה.

ניתן לסדר את הנוסחה גם בצורה נוספת:

$$(21)\ \iint f(x,y)dxdy = \sum_{\rightarrow} -\frac{y}{2} + \sum_{\uparrow} \frac{x}{2} + \sum_{\leftarrow} \frac{y}{2} + \sum_{\downarrow} -\frac{x}{2}$$

כאשר כיוון החץ מייצג לסכום את כל הנקודות בהיקף אשר מתקדמות לכיוון החץ.

האלגוריתם שלנו מנצל את נוסחה זו ומחשב את השטח של הצורה לפי נוסחה שרצה על כל הנקודות בהיקף ובכל פעם בודקת שתי נקודות עוקבות:

$$(22)\ Area = \frac{\sum\left(x_{previous}\cdot y_{current} - x_{current}\cdot y_{current}\right)}{2}$$

נראה ששתי הנוסחאות זהות:

- כאשר ההיקף נע לכיוון ימין ערך ה-*y* של שתי הנקודות זהה ורק ערך ה-*x* גדל ב-1 ולכן:

$$\frac{x_{previous}\cdot y_{current} - x_{current}\cdot y_{current}}{2} = \frac{xy-(x+1)y}{2} = -\frac{y}{2}$$

- כאשר ההיקף נע למעלה ערך ה-*x* של שתי הנקודות זהה ורק ערך ה-*y* קטן ב-1 ולכן:

$$\frac{x_{previous}\cdot y_{current} - x_{current}\cdot y_{current}}{2} = \frac{xy-x(y-1)}{2} = \frac{x}{2}$$

- כאשר ההיקף נע שמאלה ערך ה-*y* של שתי הנקודות זהה ורק ערך ה-*x* קטן ב-1 ולכן:

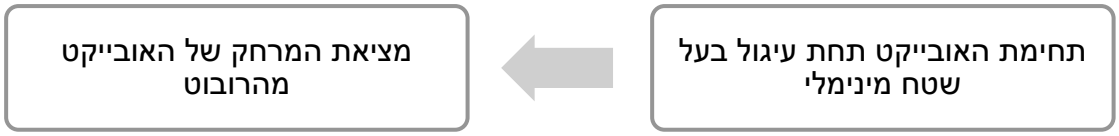
$$\frac{x_{previous}\cdot y_{current} - x_{current}\cdot y_{current}}{2} = \frac{xy-(x-1)y}{2} = \frac{y}{2}$$

- כאשר ההיקף נע למטה ערך ה-*x* של שתי הנקודות זהה ורק ערך ה-*y* גדל ב-1 ולכן:

$$\frac{x_{previous}\cdot y_{current} - x_{current}\cdot y_{current}}{2} = \frac{xy-x(y+1)}{2} = -\frac{x}{2}$$

בעזרת אלגוריתם זה מצאנו את השטחים של כלל האובייקטים ושמרנו רק את האובייקט עם השטח הגדול ביותר, הוא האובייקט שלנו.

שלב 3



בשלב זה, האלגוריתם מחזיק בגבולות של האובייקט בו אנחנו מעוניינים בלבד. נרצה לחסום את האובייקט בתוך העיגול בעל הרדיוס המינימלי שמכיל שאת כל הנקודות של האובייקט. נעשה זאת בעזרת הפונקציה minEnclosingCircle אשר פועלת באופן הבא [6]:

1. אם האובייקט מכיל רק נקודה אחת - העיגול הוא הנקודה הזו בלבד.
2. אם האובייקט מכיל רק שתי נקודות – מרכז הכדור בממוצע בין הנקודות והרדיוס הוא המרחק של הנקודות מהממוצע.
3. האלגוריתם לוקח שלוש נקודות מהאובייקט וחוסם אותן על ידי עיגול מינימלי (מרכז העיגול הוא מפגש האנכים האמצעיים וכל שלוש הנקודות יושבות על ההיקף של העיגול)
4. האלגוריתם לוקח נקודה נוספת מהאובייקט. אם הנקודה מוכלת בתוך העיגול שחושב בשלב 3, חזור על שלב 4. אחרת, חזור לשלב 3 כאשר שלוש הנקודות הן שתיים מהנקודות הקודמות והנקודה החדשה. האלגוריתם חוזר לשלב 3 עבור כל צירוף אפשרי של שתיים מתוך 3 הנקודות הקודמות והנקודה החדשה עד שהוא מוצא עיגול שמכיל גם את הנקודה שהוחלפה.

תנאי העצירה של האלגוריתם הוא שעברנו על כל הנקודות באובייקט והוא מחזיר את רדיוס המעגל הסופי שנקבע ואת המרכז שלו.

בעזרת הרדיוס ומרכזו של העיגול החוסם שקיבלנו ניתן לחשב את מרחקו של האובייקט מהרובוט בעזרת משוואות (9) , (8) שפותחו בחלק התאורטי.

בקרת תנועת הרובוט

כעת נסביר את מימוש הבקרה על הרובוט. הסברנו בחלק התיאורטי כיצד הגענו למציאת הטווח האסור לתנועה, כעת נסביר את הבקרה שהרובוט מבצע על מנת להתחמק מעשית מהמכשול. נזכיר כי מדובר בהתחמקות אוטונומית, משמע, בעת לחיצה על כפתור ההתחמקות האוטונומית המובנה ב UI , הרובוט יפעיל שיקול דעת בעצמו, יתערב בפעולת המשתמש המתפעל את הרובוט ויתחמק מהמכשול.

לצורך מימוש הבקרה החלטנו על שימוש בפונקציית משקל $w(z)$ אשר נבנתה על ידינו בהתאם לצרכי המערכת. נרצה שככל שהרובוט קרוב יותר למכשול, לבקר יהיה משקל גדול יותר בבחירת הפעולות. לכן, נגדיר פונקציית משקל התלויה במרחק מהכדור כך ש:

- $w(z \to 0) = 1$
- $w(z \to \infty) = 0$

לכן, פונקציית המשקל שלנו היא מהצורה:

$$(23)\ w(z) = \frac{1}{1 + \alpha^{z-\beta}}$$

כאשר[6]:

- α – פרמטר המשפיע על מהירות הדעיכה של הפונקציה. $\alpha > 1$
- β – פרמטר המזיז את הפונקציה בציר z.

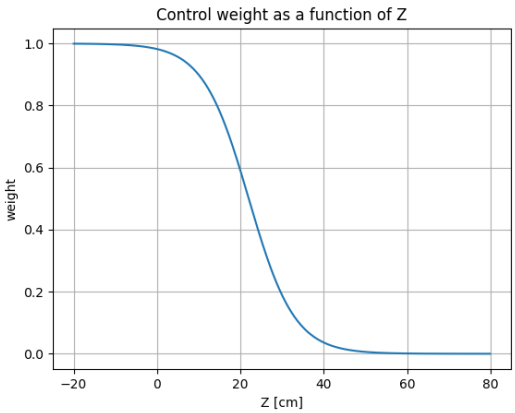
מתוך ניסוי וטעיה, ובדיקה עבור מקרים שונים, ראינו כי ערכי פונקציית המשקל יהיו שונים עבור הבקר ועבור הטווח האסור:

עבור הבקר:

$$\alpha = 1.2\ , \beta = 22$$

כך שמתקבל:

$$w_1(z) = \frac{1}{1 + 1.2^{z-22}}$$



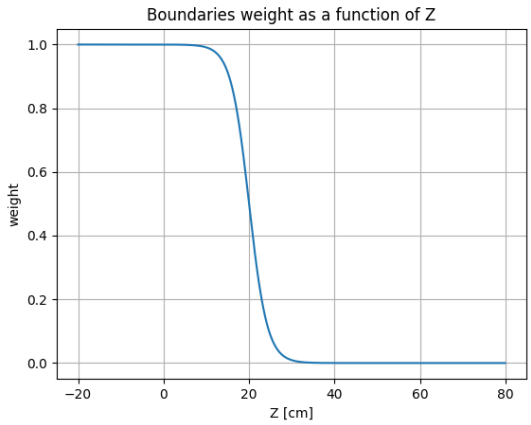
איור 14 – פונקציית משקל עבור הבקר

עבור הטווח האסור:

$$\alpha = 1.6\,, \beta = 20$$

כך שמתקבל:

$$w(z) = \frac{1}{1 + 1.6^{z-20}}$$



איור 15 – פונקציית משקל עבור הטווח האסור

פונקציית המשקל תשפיע על הטווח האסור באופן הבא:

$$(24)\ \delta_{1,corrected} = \delta_1 + w(z) \cdot c$$
$$(25)\ \delta_{2,corrected} = \delta_2 - w(z) \cdot c$$

כאשר c נקבע לפי ניסוי וטעיה.

בהנחה שזווית תנועת הרובוט שנבחרה על ידי המשתמש נמצאת בטווח האסור, הבקר ישפיע על הזווית בהתאם לפונקציה הבאה:

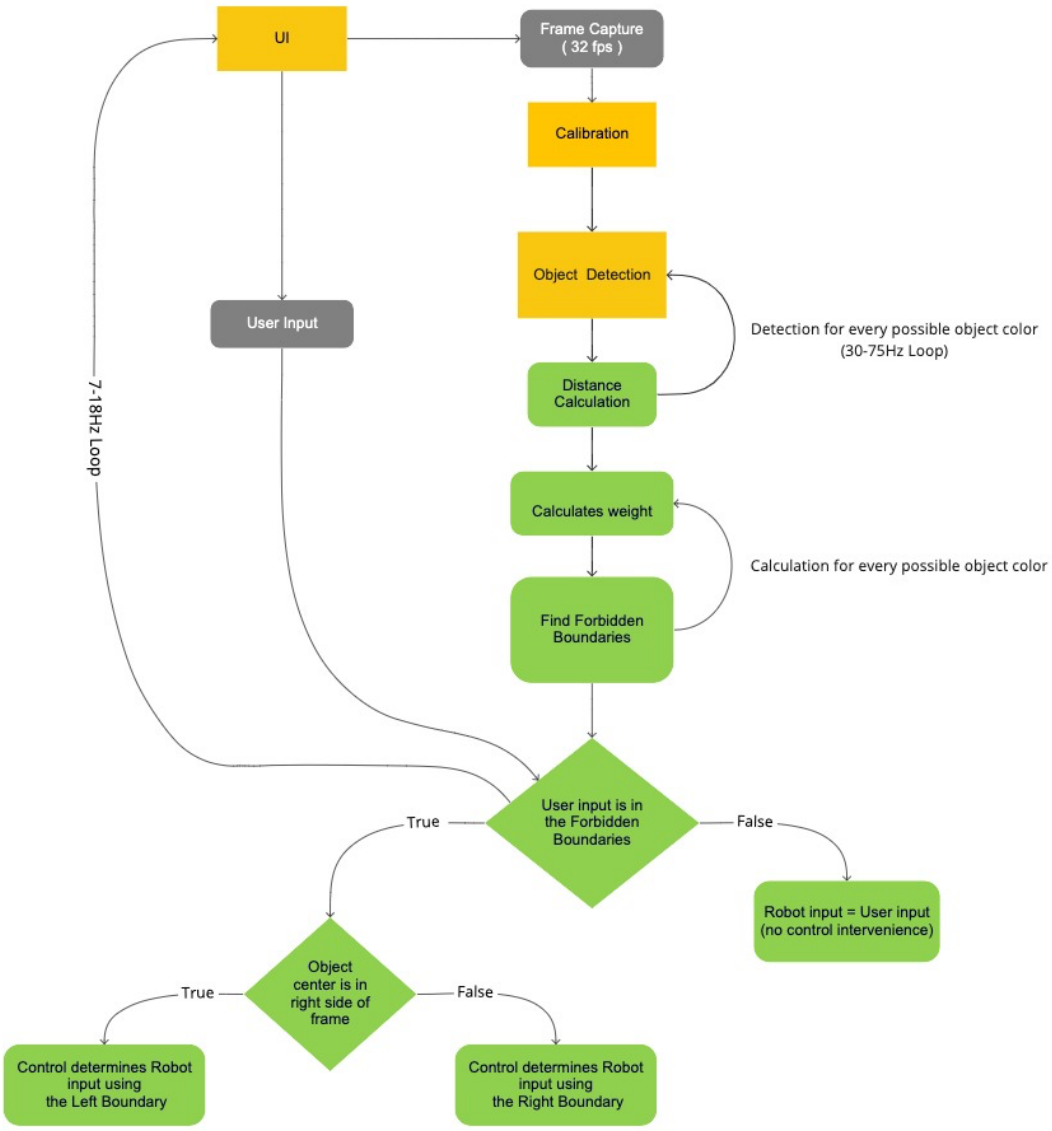
$$(26)\ angle_{corrected} = w(z) \cdot angle_{boundry} + [1 - w(z)] \cdot angle_{user}$$

כאשר $angle_{boundry}$:

- מהווה הקצה השמאלי של הטווח האסור כאשר מרכז האובייקט נמצא מימין לרובוט.
- מהווה הקצה הימני של הטווח האסור כאשר מרכז האובייקט נמצא משמאל לרובוט.

במקרה של מרחק 0 מהכדור, הטווח האסור נקבע להיות $[0°, 180°]$ ולכן הרובוט יתחיל לבצע סיבוב במקום עד שהכדור לא יהיה בגבולות התמונה ומשם הרובוט ממשיך בהתאם לתנאי השטח.

דיאגרמת הבלוקים הבאה מתארת את פעולת הבקר:



איור 16 – דיאגרמת בלוקים עבור פעולת הבקר

ממשק משתמש - UI

ממשק המשתמש אחראי על איחוד כל קטעי הקוד השונים והצגתם בצורה ברורה ונוחה למשתמש.

עבור בניית ה UI נעזרנו בספרית Flask על מנת להקים את השרת שיאפשר הצגת וידאו ובספרית dash על מנת לקשר בצורה נוחה בין קוד הפייתון של האלגוריתם שלנו לבין קוד ה HTML של ה UI.

כפי שניתן לראות מאיור 15, הממשק מציג תצלום בזמן אמת של מצלמת הרובוט ושלושה אובייקטים של קלט מהמשתמש:

- **ג'ויסטיק שאחראי על תנועה הרובוט:**

הזזה שלו על ידי המשתמש מפעילה פונקציה באלגוריתם אשר מקבלת בקלט את הזווית והמרחק של הג'ויסטיק ביחס למרכז, אשר מייצגים את זווית ומהירות הרובוט. ערכים אלו מוצגים בפינה השמאלית העליונה של הממשק ומתעדכנים בהתאם למשתמש.

- **כפתור הפעלת ההתחמקות האוטונומית:**

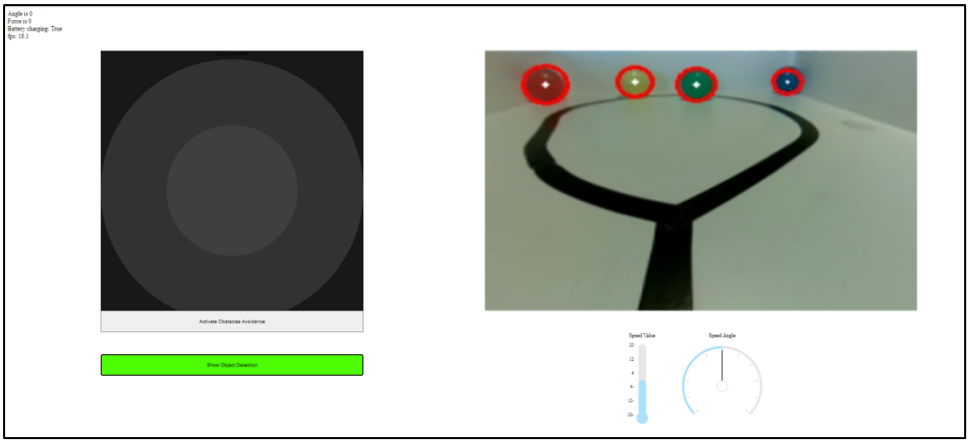
בעת לחיצה, הכפתור הופך לירוק כדי לסמן שההתחמקות האוטונומית הופעלה. בכל עת שהכפתור מופעל, קלט המשתמש מבוקר והרובוט יבצע התחמקות בהתאם לתנאי השטח.

- **כפתור הצגת התמונה המנותחת (לאחר כיול ומדידת ערכים):**

בעת לחיצה, הכפתור הופך לירוק כדי לסמן שמוצגת התמונה המנותחת. בכל עת, הקוד מחזיק בפריים המקורי ובפריים המתוקן. בעזרת כפתור זה המשתמש יכול לבחור כיצד הוא רוצה לראות את המערכת.

בנוסף, מוצגים בפינה השמאלית העליונה פרטים שונים כגון מספר פריימים בשנייה וסטטוס טעינה של הרובוט אשר מתעדכנים בכל 0.7 שניות.

כערך מוסף, החלטנו לחשב ולהציג ב UI את מהירות הרובוט וזווית התנועה שלו בעזרת שימוש במיקום הכדורים בפריימים השונים[7].



איור 17 – UI

[7] ראה הרחבה בפרק סיכום, מסקנות והצעות להמשך, עמוד 32

4 ניתוח תוצאות

בפרק זה נדון בתוצאות הפרויקט שלנו. מימשנו אלגוריתם אשר מבצע התחמקות אוטונומית.

כעת אנו נציג ניסויים שונים אשר סייעו לנו להגיע לתוצר הסופי של הפרויקט:

4.1 ביצועי המערכת מבחינת זמן אמת

המטרה שלנו הייתה לקבל מערכת אשר יכולה להגיב בזמן אמת לשינויים בתנועת הרובוט ולבצע עליו בקרה. לשם כך, עלינו לוודא כי המערכת פועלת בכמה שיותר פריימים לשנייה (frames per second – fps) כדי שתוכל להגיב בצורה מדויקת לשינויים הקורים בזמן אמת. הבנו כי קיים קשר ישיר בין רזולוציית התמונה שנלקחת מהמצלמה היושבת על גבי הרובוט לבין מספר הפריימים בשנייה אותו יכול לעבד הקוד.

לקחנו 6 רזולוציות שונות. נדגיש כי ככל שהרזולוציה קטנה יותר כך גודל המידע שנלקח קטן יותר ולכן הרזולוציה המינימלית נבחרה כך שהקוד עדיין עובד בצורה נכונה.

להלן טבלה המתארת את הקשר בין הרזולוציה של התמונה לבין מספר הפריימים בשנייה:

Resolution	fps
1024×768	7.7
640×480	11.8
512×384	13.6
480×360	14.6
416×312	16.6
320×240	17.9

טבלה 2 – בדיקת רזולוציות

הרזולוציה ניתנת לשינוי בקוד באופן חופשי.

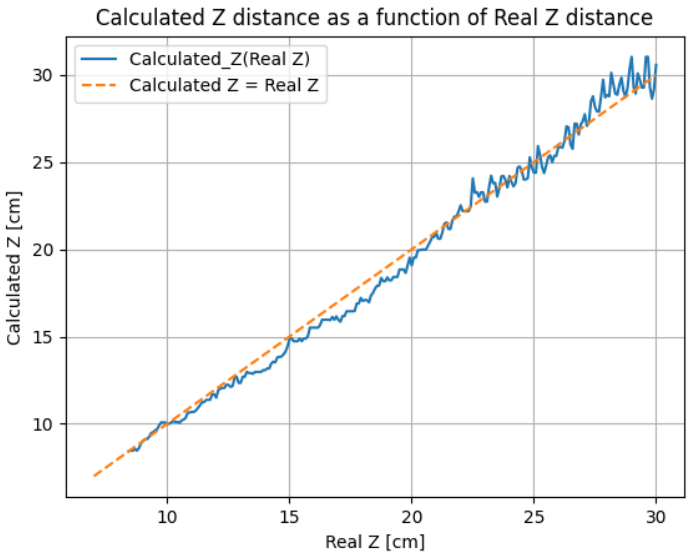
בחרנו לעבוד ברזולוציה שהפיקה fps מקסימלי והיא 240 x 320.

4.2 ביצועי המערכת מבחינת אומדן מרחק ממכשול

על המערכת לאמוד את המרחק של המכשול על מנת שהרובוט יוכל להתחמק בהתאם לתנאי השטח. לשם כך כתבנו קוד אשר ידע לחשב את המרחק המדויק של הרובוט מהמכשול.

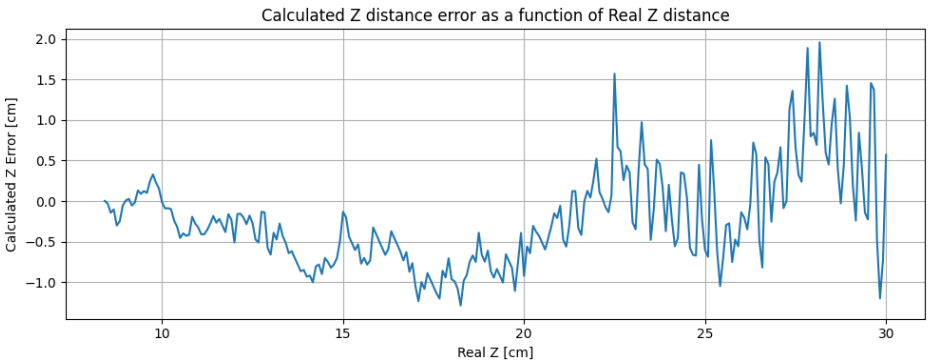
<u>תוצאות ניסוי אומדן המרחק בציר Z [אנכי]:</u>

לקחנו כדור ומיקמנו אותו במרחק $30[cm]$ בציר Z האנכי ביחס לרובוט. הזזנו את הכדור בציר זה לכיוון מרחק $0[cm]$ בקצב אחיד עד אשר לא נראה יותר על ידי המצלמה. השוונו את המרחקים שחישב האלגוריתם למרחקים האמיתיים וקיבלנו את הגרף הבא:



איור 18 –ניסוי אומדן מרחק ממכשול בציר אנכי

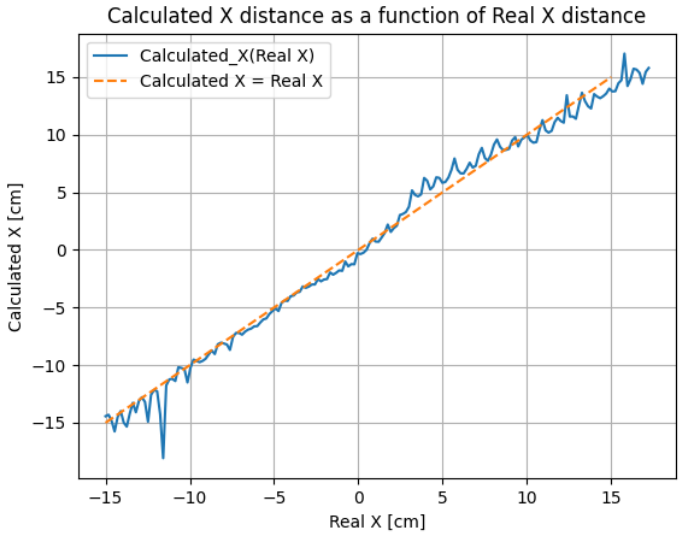
מאיור 17 ניתן לראות כי המרחקים המחושבים על ידי הרובוט תואמים למרחקים האמיתיים עד כדי שגיאה של $\pm 2[cm]$ אשר מספקת עבור ביצוע ההתחמקות.



איור 19 –שגיאת אומדן מרחק ממכשול בציר אנכי

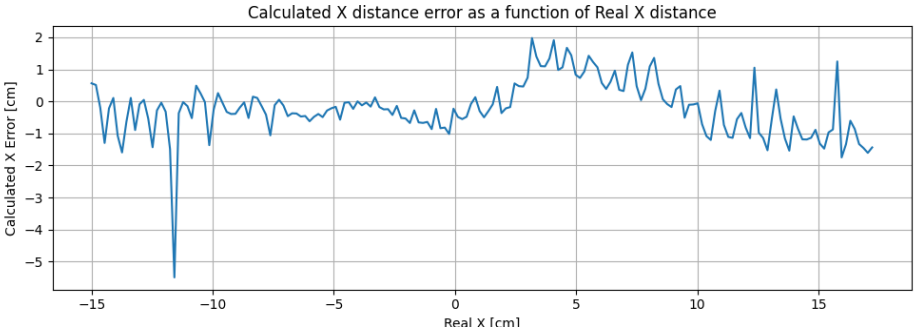
תוצאות ניסוי אומדן המרחק בציר X [אופקי]:

לקחנו כדור ומיקמנו אותו במרחק $-15[cm]$ בציר X האופקי ביחס לרובוט. הזזנו את הכדור בציר זה לכיוון מרחק $+15[cm]$. השוונו את המרחקים שחישב האלגוריתם למרחקים האמיתיים וקיבלנו את הגרף הבא:



איור 20 –ניסוי אומדן מרחק ממכשול בציר אופקי

מאיור 19 ניתן לראות כי המרחקים המחושבים על ידי הרובוט תואמים למרחקים האמיתיים עד כדי שגיאה של $\pm\mathbf{2}[\boldsymbol{cm}]$ עם חריגה יוצאת דופן שנבעה ככל הנראה מהסתרה לא מכוונת של הכדור על ידי מבצע הניסוי. בכל זאת, תוצאות אלה מספקות עבור ביצוע ההתחמקות.



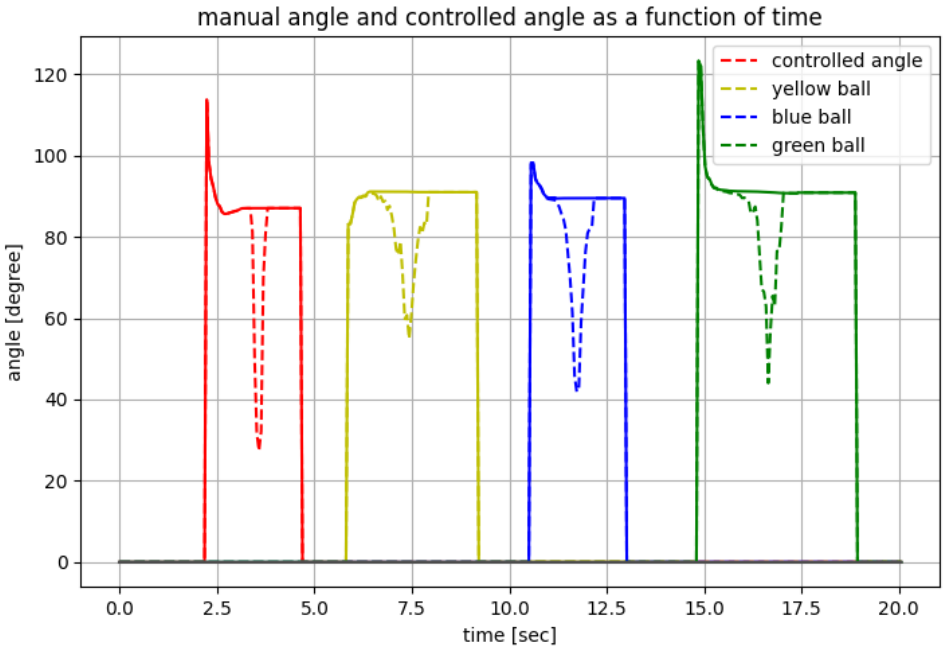
איור 21 –שגיאת אומדן מרחק ממכשול בציר אופקי

4.3 ביצועי המערכת מבחינת ביצוע התחמקות

כעת נראה כי אכן מתבצעת התחמקות כאשר קיים צורך.

מיקמנו את כל ארבעת הכדורים במרווחים שונים מהרובוט כך שהתערבות של האלגוריתם תהיה הכרחית למניעת התנגשות והרובוט יתחמק בזה אחר זה מכל מכשול.

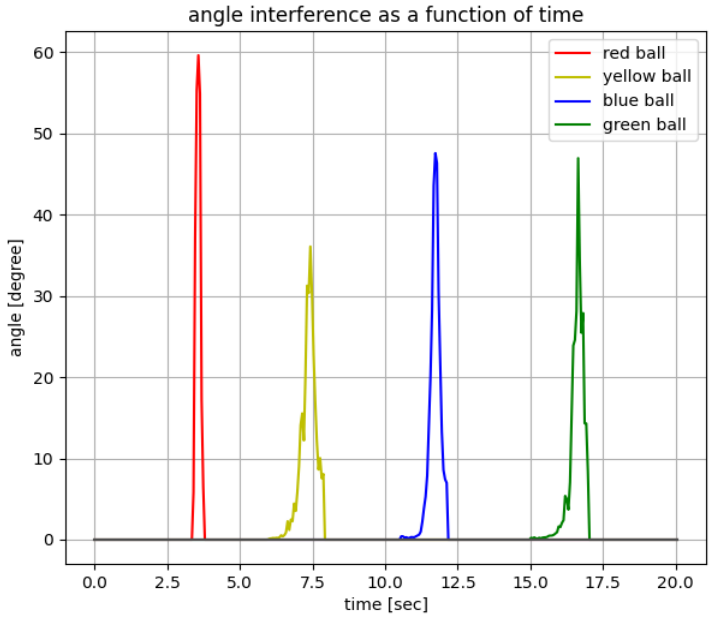
בגרף הבא מתוארים ביחד זווית המשתמש כפונקציה של הזמן וזווית הבקר כפונקציה של הזמן.



איור 22 – זווית המשתמש והבקר כפונקציה של הזמן

ניתן לראות בדיוק את הרגע בו הבקר עוקף את פקודת המשתמש, משנה את זווית התקדמות הרובוט ומונע התנגשות במכשול.

כאשר מבצעים חיסור בין שני הגרפים מתקבל הגרף הבא אשר מתאר את ההפרש בין הזווית שהכניס המשתמש לבין הזווית שקבע הבקר.



איור 23 –הפרש הזוויות כפונקציה של הזמן

ניתן לראות כי כאשר אין צורך בהתערבות הבקר, ההפרש עומד על אפס. כאשר הבקר מתערב, הזווית משתנה וכאשר הרובוט יוצא מסכנת התנגשות, הבקר מפסיק את ההתערבות וההפרש בין הזוויות חוזר להיות אפס.

נשים לב כי צורת ההתחמקות זהה בקירוב עבור כל צבעי הכדורים. נסביר זאת על ידי כך שלאחר שבחרנו טווח ערכי HSV עבור כל הצבעים והתמונה הפכה לבינארית, לא יהיה קיים הבדל בהתנהגות הקוד עבור צבעי הכדורים השונים.

5 סיכום, מסקנות והצעות להמשך

לאחר ניתוח תוצאות הפרויקט ניתן לראות כי עמדנו במטרות שהגדרנו. כתבנו אלגוריתם אשר מאפשר התחמקות אוטונומית ממכשולים וממשק משתמש אשר מאפשר שליטה נוחה במערכת.

במהלך הפרויקט שמנו לב לכמה היבטים שאנו חושבים כי ניתן לשפר בעתיד:

- בעת נסיעה איטית של הרובוט, יתכנו מצבים בהם קלט המשתמש יבוקר כראוי אך עקב מגבלותיו הפיזיות של הרובוט, כלומר מבנה הגלגלים שלו (תנועה זחלית) , הרובוט עלול להתקשות להגיע לזווית ההתקדמות המבוקרת דבר אשר יכול לגרום להתנגשות.

- סוללת הרובוט לעיתים התנהגה בצורה לא צפויה אשר הובילה למגבלות בעת עבודה מרחוק. נציע פיתוח אלגוריתם אשר מקבל כקלט את מתח הסוללה, מחשב את אחוזי סוללת הליתיום הנותרים לפי תכונותיה הפיזיקליות ומציג אותם למשתמש.

- עקב העובדה כי הרובוט מכיל מצלמה קדמית אחת בלבד עם טווח ראיה מוגבל לעד $\mathbf{160°}$, בכל פעם שאובייקט נמצא מחוץ לטווח ראיה זה או מוכל רק בחלקו, יתכן מצב של התנגשות משום שקלט המשתמש לא יבוקר בצורה ראויה.

לצורך פתרון בעיה זו אנו מציעים לפתח את האלגוריתם כך שיוכל לזכור את מיקום האובייקטים שהרובוט ראה בעבר ולהתחשב בהם בעת בקרת תנועת הרובוט. עבור מימוש פתרון על הרובוט לדעת היכן הוא ממוקם במרחב וכך הוא יוכל להעריך את מרחקו מכדורים אשר לא בטווח הראיה שלו.

כערך מוסף, התחלנו לפתח את יכולת זו עבור הרובוט. הקלטים שעומדים לרשותנו בשביל פיתוח המערכת הם:

- מרחקי האובייקטים מהרובוט עבור כל פריים.
- הפרש הזמנים בין כל פריימים.

בעזרת קלטים אלו, חישבנו את מהירות תנועת האובייקט בתמונת הרובוט בין 2 פריימים אשר נלקחו בפרק זמן ידוע ומתוכם חילצנו את מהירות הרובוט[8].

בפיתוח עתידי, יהיה ניתן לחלץ מתוך מהירות הרובוט את מיקומו במרחב וכך להעריך את מרחקו מאובייקטים אשר לא נמצאים בטווח הראיה שלו.

[8] ראה איור 17 – UI, עמוד 26

מקורות

מאמר:

[1] L. Fernandez, V. Avila, and L. Gonçalves. "A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors", Federal University of Rio Grande do Norte, May 2017.

[2] Z. Zhang. "A Flexible New Technique for Camera Calibration", Microsoft Corporation, December 1998.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf

[3] G. A. Watson, Numerical Analysis, University of Dundee,1977
https://hwbdocuments.env.nm.gov/Los%20Alamos%20National%20Labs/General/32166.pdf

[4] S. Suzuki, K. Abe "Topological Structural Analysis of Digitized Binary Images by Border Following", Computer Vision, Graphics, And Image Processing N,32-46, 1985

[5] S. Brlek, G. Labelle, A. Lacasse, The discrete Green Theorem and some applications in discrete geometry, Theoretical Computer Science 346 ,200 – 225 ,2005

[6] E. Welzl. “Smallest Enclosing Disks (balls and Ellipsoids)”, The Free University of Berlin,1991

דף נתונים של רכיב:

[7] Hardware Data: Raspberry Pi 4 Model B

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[8] Camera Data: IMX219-160 8MP Camera with 160° FOV

https://www.waveshare.com/imx219-160-camera.htm

https://www.seeedstudio.com/IMX219-160-Camera-160-FOV-Applicable-for-Jetson-Nano-p-4603.html

קישורים למקורות באינטרנט:

[9] "Camera Calibration: Explaining Camera Distortions"

https://ori.codes/artificial-intelligence/camera-calibration/camera-distortions/

[10] "Create calibration pattern"

https://docs.opencv.org/4.x/da/d0d/tutorial_camera_calibration_pattern.html

קישורים עבור Github :

[11] "ZumoPi" by Arkadi Rafalovich

https://github.com/TAU-Robotics/ZumoPi

[12] "ZumoPi Obstacle Avoidance" by Ido Arad and Meitar Amir

https://github.com/Meitar2908/ZumoPi-Obstacle-Avoidance