The Iby and Aladar Fleischman Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

Indoor localization - Camera

פרויקט מס' 22-1-1-2738

דו"ח סיכום

מבצעים:

לורן סולובודינסקי 208786574

אביטל זיסמנוב 318502754

מנחים:

ארקדי רפאלוביץ אוניברסיטת ת"א

מקום ביצוע הפרויקט: אוניברסיטת תל אביב

תוכן עניינים

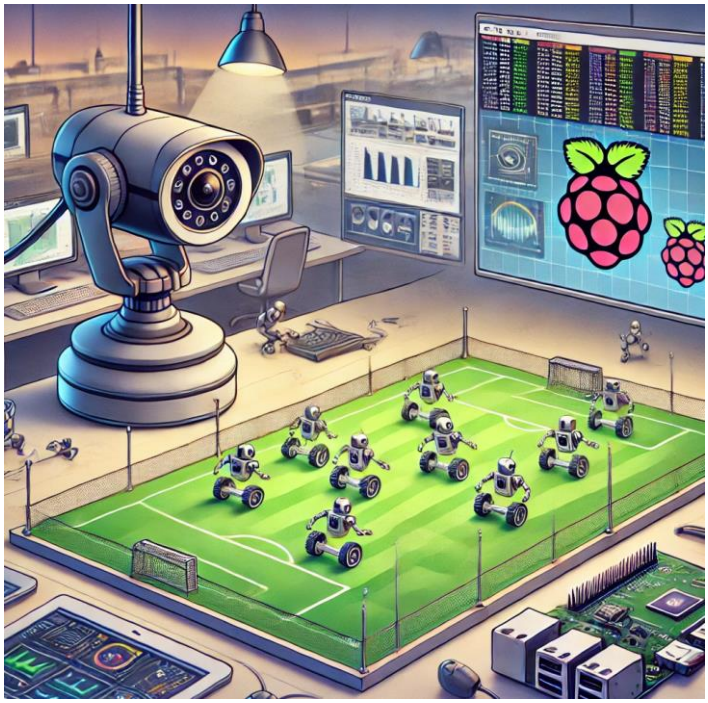
רשימת איורים

רשימת טבלאות

תקציר

מטרת הפרויקט היא לפתח מערכת עקיבה אחר רובוטים שתהיה זולה באופן משמעותי לעומת מערכות עקיבה אופטיות קיימות, ובפרט להחליף את מערכת ה Opti Trek-הנמצאת במעבדת הבקרה של אוניברסיטת תל אביב. המערכת כוללת מצלמה הממוקמת בצורה אסטרטגית כך שהיא מכסה את כל שטח הפעולה של הרובוטים. כל רובוט מסומן בברקוד ייחודי, המאפשר זיהוי חד ערכי ומעקב אחר מספר רובוטים בו זמנית. המצלמה מחוברת לבקר Raspberry Pi המכיל אלגו' זיהוי ועכיבה חדש המיועד לבצע את עיבוד המידע המתקבל ולשלוח אותו למחשב / לרובוט. המערכת תספק תשתית לניטור ובקרה על תנועות הרובוטים, ותשפר את יכולות הבקרה במעבדה, כמו כן, הגדלנו ראש ובמלך הפרויקט ניסינו להוסיף רובוסטיות למערכת כך שללא תלות בתנאי סביבה, משאבים נוכל להתקין את המערכת בתצורת "Plug N Play" ולקבל זיהוי ועכיבה בצורה גבוה ומהירה.

שרטוט הממחיש איך אמור להראות סט-אפ הפרויקט:



איור 1 – שרטוט הממחיש את תוצר הפרויקט אותו נחבר במעבדה

1 הקדמה

כאמור, מטרת הפרויקט היא לפתח מערכת מעקב רובוטים מדויקת וזולה באמצעות כלים של ראייה ממוחשבת בלבד. כפי שנאמר, כל רובוט יצויד במרקר ייחודי, הכולל צורה מורכבת שתוכננה באמצעות ארדואינו, נורות לד, נגדים והדפסת תלת מימד. האלגוריתם לגילוי רובוטים יהיה מותאם לזיהוי, איתור ומעקב בזמן אמת של כל רובוט נע. הפרויקט יכלול כיול מצלמה ופיתוח אלגוריתם עיבוד תמונה חדש, המותאם למרקר החדש שהורכב.
המוטיבציה העיקרית מאחורי פרויקט זה היא להפוך את טכנולוגיית המעקב אחר רובוטים לנגישה ומשתלמת. מערכות מעקב אופטיות קיימות יקרות מאוד, מה שמגביל את השימוש בהן בסביבות חינוכיות ובפרויקטים של סטודנטים. על ידי פיתוח פתרון בעלות נמוכה, אנו מקווים להקל על יישום מערכות מעקב רובוטים במעבדות חינוכיות וביישומים רחבים יותר. יוזמה זו שואפת לגשר על פער העלות תוך שמירה על ביצועים חזקים, ולעודד חדשנות ואימוץ רחב יותר של הטכנולוגיה בסביבות חינוכיות ומחקריות. לשם כך, הוספנו לעצמנו אתגר והוא לדרוש Robustness לסט-אפ העבודה, כלומר לתמוך בכמה שיותר סטאפים ללא צורך בכיולים וללא תלות בתנאי סביבה.

שלבי הפרויקט יכללו הקמת תשתית למערכת, ביצוע מדידות על המרקר החדש, פיתוח אלגוריתם זיהוי ומעקב ייחודי, וכיול המצלמה בהתאם לדרישות הביצועיות (Benchmarks) של הפרויקט.

חשיבות הטכנולוגיה הזו ניכרת במגוון תחומים, כמו לוגיסטיקה, שם יש שימוש נרחב ברובוטים לניהול חבילות ומלאי במחסנים. מערכות עקיבה דומות מסייעות לוודא תנועה נכונה של הרובוטים, למנוע תאונות, ולהבטיח שהחבילות מגיעות ליעד הנכון.

דרך העבודה שלנו כללה את השלבים הבאים:

- הרצת תשתית ישנה בכדי להבין את ביצועיה.
- סקירת חומרה מתאימה:
  - נסיון עבודה עם מספר מצלמות וכיולם – לבסוף נבחרה מצלמת fisheye.
  - בחירת RPi שיתמוך בתשתית ROS2 ובאלגו'.
- יצירת מרקר אלקטרוני ובדיקתו.
- פיתוח אלגוריתם המזהה את המרקר ונותן מיקום בזמן אמת של הרובוט וניסון הרצתו על Rpi.

נציין, שלבסוף, נוכח הנסיבות, ויתרנו על שימוש בRpi והרצנו את המערכת ממחשבים ניידים + מצלמת רשת.

2 רקע תיאורטי

כיול מצלמה

כיול המצלמה התבצע על ידי מציאת הפרמטרים הפנימיים והחיצוניים של המצלמה, המתארים את המעבר מהעולם התלת-ממדי (3D) לתמונה הדו-ממדית (2D). המודל שבו השתמשנו היה "Pinhole" שמתאר את התהליך בעזרת מטריצות של סיבוב ותרגום, וכן מקדמי עיוות כמו עיוות רדיאלי ומשיקי, המייצגים את השפעת העדשה על התמונה. בעצם במודל זה מתייחסים למצלמה כאל חור קטן שמאפשר לקרני אור לעבור דרכו ולהקרין את התמונה על מישור מאחוריו. המודל מתאר את היחס בין הנקודות במרחב התלת-ממדי (העולם האמיתי) לבין הנקודות במישור התמונה ומאפשר להבין את היחסים בין אורכי המוקד של העדשה לבין גודל התמונה וזווית הראייה.

פרמטרים פנימיים של המצלמה הם הפרמטרים שמתארים את המאפיינים הפנימיים של המצלמה והעדשה, והם כוללים: אורך מוקד (המרחק בין העדשה למישור התמונה), מרכז התמונה ועיוותים.

פרמטרים חיצוניים של המצלמה מתארים את המיקום והכיוון של המצלמה ביחס לאובייקטים של העולם האמיתי וחהם כוללים: סיבוב (זווית הסיבוב של המצלמה ביחס למערכת הקואורדינטות של העולם) ותרגום (מיקומם המצלמה במרחב ביחס לאובייקטים).

בעדשות רחבות, כמו עדשות fish-eye קיים עיוות משמעותי שנובע מצורת העדשה. העיוות הרדיאלי מתרחש כאשר קרני האור החודרות לעדשה מתכופפות בצורה לא שווה, כך שבקצוות התמונה נוצרות סטיות, בעוד במרכז התמונה העיוות פחות משמעותי. עיוות משיקי מתרחש כאשר העדשה אינה ממוקמת במקביל למישור התמונה, מה שגורם לשינוי בגודל ובצורה של האובייקטים. לכן תהליך הכיול כולל מציאת הפרמטרים הפנימיים והחיצוניים של המצלמה על מנת לתקן את העיוותים האופטיים ולהשיג תיאור מדויק של המציאות בתמונה. המטרה היא לקבוע את הפרמטרים המדויקים כך שהקשר בין התמונה שהמצלמה יוצרת לבין המיקום והכיוון האמיתיים של האובייקטים בעולם יהיה ידוע ומדויק.

זיהוי המרקר

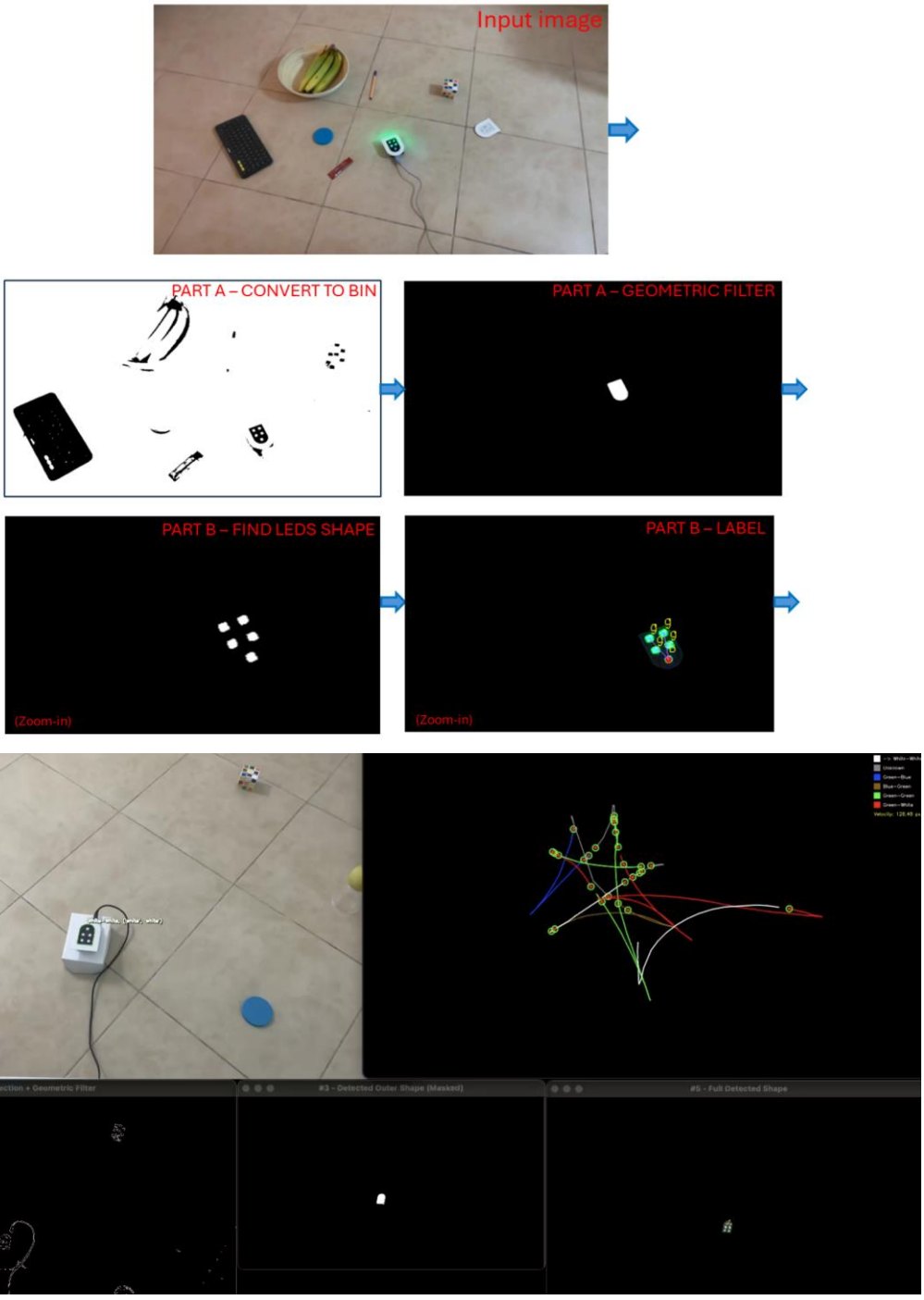
נפרט על טכניקות שונות מעולם עיבוד התמונה בהן השתמשנו:

- Binary Thresholding - טכניקה להמרת תמונה לגווני אפור לתמונה בינארית. כל הפיקסלים בתמונה שמעל ערך סף מסוים יקבלו ערך של 1 (לבן), וכל הפיקסלים שמתחת לסף יקבלו ערך של 0 (שחור). סף בינארי הוא כלי פשוט אך יעיל להפרדת אובייקטים מהרקע בתמונה.
- Bilateral Filter - טכניקה להחלקת תמונה תוך שמירה על הקצוות. הפילטר פועל על שני פרמטרים – הבדל במיקום והבדל בערכי הפיקסלים. הוא מחשב את הממוצע של פיקסלים קרובים, אך מעניק משקל גבוה יותר לפיקסלים בעלי ערך אינטנסיביות דומה, ובכך נמנע הטשטוש בקצוות האובייקטים.
- Canny Edge Detector - אלגוריתם לגילוי קצוות בתמונה. הוא מבצע מספר שלבים: החלקת התמונה (לרוב עם פילטר גאוסי), חישוב שיפועי התמונה, איתור הקצוות הפוטנציאליים על ידי מציאת המקומות עם שיפוע חד, ולאחר מכן תהליך של סף כפול לדיכוי קצוות חלשים.

- Suzuki-Abe Algorithm - אלגוריתם לזיהוי קווי מתאר (Contours) בתמונה. הוא מתבסס על שיטת "Chain Approximation" כדי למצוא גבולות של אזורים בתמונה בינארית. האלגוריתם מזהה קווי מתאר חיצוניים ופנימיים של אובייקטים, כולל חורים או אזורים סגורים בתוך אובייקטים.

3 סימולציה

להלן תוצאות הסימולציות שהרצנו לבדיקת האלגוריתם לפי השלבים המתוארים בפרט המימוש:



איור 2 - תוצאות סימולציה

בתמונה הראשונה רואים מספר מסכים המכילים את שלבי הזיהוי והרכישה של הרובוט בהנתן פריטים שמקשים על הזיהוי, ואילו בתמונה השנייה רואים בנוסף לתמונה הראשונה גם מסך המראה את זיהוי התיוג של רובוטים שונים כאשר הם מסומלצים ע״י שינוי צבעי הLEDים של הרובוט (זה בעצם Tracked Paths של כל רובוט).

לינק[4] מכיל סרטון המראה בזמן אמת את הסימולציה ונמצא בנספחים.

4 מימוש

בהתאם לדיאגרמת הבלוקים של הפרויקט, נתאר בפרק זה את מימוש התוכנה הכולל את כיול המלצמה ואת אלגוריתם הזיהוי. מימוש החמרה יכלול את הברקוד האלקטרוני.

4.1 תיאור חומרה

חומרת המערכת מכילה 3 אלמנטים מרכזיים עליהם נפרט:

4.1.1 מצלמה

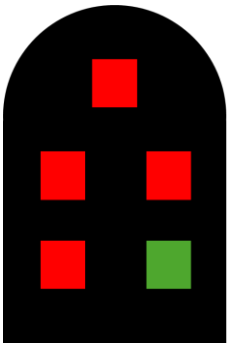
בתחילת הדרך השתמשנו במצלמת 6 מ"מ -MP3 שסופקה לפרויקט, מבית RPi, אשר נחשבה למצלמה איכותית המתאימה לצרכים שלנו. עם זאת, נתקלנו בקשיים במהלך כיול מוקד העדשה בשל תקלה טכנית במצלמה שלא אפשרה לנו להגיע לאיכות התמונה הרצויה. בעקבות זאת, החלטנו לבדוק מספר מצלמות RPi נוספות שהיו זמינות לנו במעבדה. לאחר בחינה של מספר אפשרויות, מצאנו שמצלמת רשת של מחשבי המעבדה, יחד עם מצלמת RPi2 מדגם PT361060M3MP12, מספקות פתרון יציב יותר עבור הצרכים שלנו, עם איכות תמונה משביעת רצון ומחיר נמוך יחסית.

4.1.2 בקר

כדי להוסיף רובסטיות למערכת, החלטנו לבנות את המערכת כך שתהיה Stand-Alone, כלומר שכל תהליך עיבוד המידע יתבצע על גבי הבקר עצמו, ללא צורך במחשב חיצוני. בתחילה עבדנו עם בקר RPi5, שכן הוא עמד בדרישות המפרט הטכני הנדרש להרצת האלגוריתם שלנו. עם זאת, גילינו בהמשך ש-ROS2, אינו נתמך עדיין על גבי RPi5. עקב כך, נאלצנו לעבור ולעבוד עם בקר RPi3, שהיה מתאים יותר מבחינת התמיכה והסביבה התפעולית למרות הביצועים הנמוכים יותר שלו לעומת RPi5. נציין שלבסוף ראינו כי הבקר אינו מסוגל להריץ בצורה נורמלית את האלגו' מאחר וזה הגיע ל-0.5fps ולכן ירדנו ממנו (היה צפוי) ועבדנו ממחשב נייד.

4.1.3 מרקר

המרקר שקיבלנו לעבוד איתו היה בתצורה מאוד מסוימת שהיוותה אתגר לאלגו' הזיהוי שייצרנו ונראה כך בתצורת Tombstone:

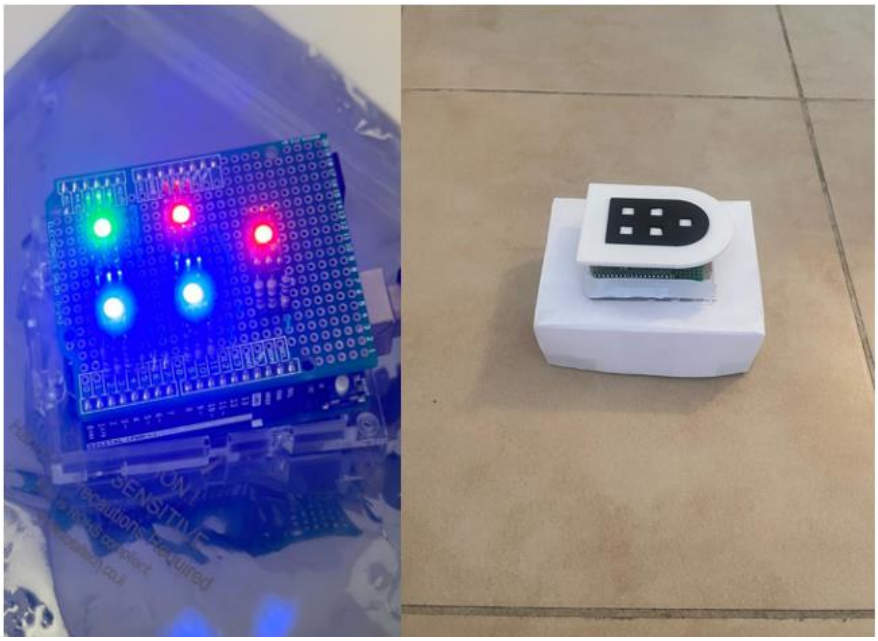


איור 3 - מרקר שיש על כל רובוט עלינו לזהות

המרקר עצמו מכיל מסגרת שחורה, ובתוכה 5 ריבועים שמהווים LEDS בצבעי RGB, כאשר כל קונפיגרוציית צבעים של הLEDS מהווה מספר סריאלי חד חד ערכי של רובוט.

מאחר והייתה בעיה ולא סופק לנו המרקר עצמו היה עלינו לייצר מדמדה רובוט + מרקר, וזאת עשינו באמצעות שימוש בארדואינו, מספר נגדי 670Ohm, RGB LEDS והלחמות שביצענו, בנייה מכנית באמצעות שימוש במדפסת תלת מיממדית של המכסה + דפיוזרים וכמו כן הרכבתם על מכונית על שלט שקנינו.

לבסוף המדמה רובוט היה נראה כך:



איור 4 – בתמונה השמאלית, המדמה רובוט לאחר הלחמות ובעת בפעלת לדים, בתמונה הימנית – המוצר הסופי כאשר הוא חובר מעל לצעצוע מכונית על השלט

4.2 תיאור תוכנה

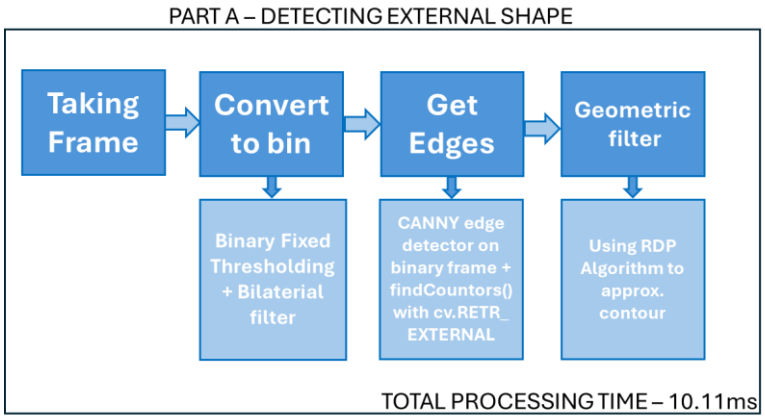
לתוכנה שרצה קיימות מספר דרישות:

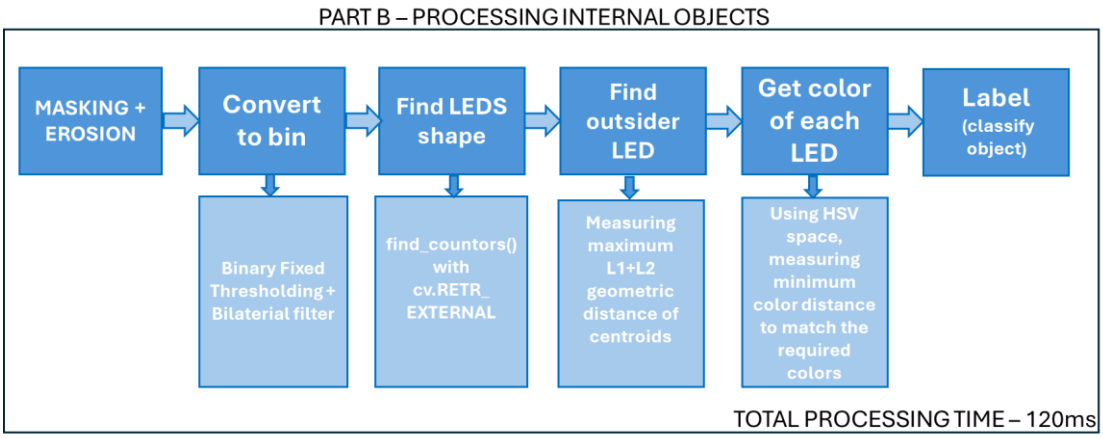
1. יכולת זיהוי ועכיבה גבוהה (מעל 80%).
2. רובוסטיות,כלומר עמידה ב-1 ללא צורך בכיולים, וללא קשר לתנאי סביבה כמו למשל רמת האור בחדר.
3. דל משאבים, כלומר עליו לרוץ על מעבדים חלשים.

בגלל הדרישות, הורדנו את האופציה של שימוש בDeepLearning, אלא להשתמש באלגו' רק בComputer Vision קלאסי כדי להקל על צריכת המשאבים של המעבד.

האלגוריתם בעצמו בנוי מ-2 חלקים עיקריים, המורכבים מתתי חלקים בהם ניסינו לעמוד בדרישות.

החלק הראשון מורכב מסינון וזיהוי הצורה החיצונית של הרובוט, ואילו החלק השני מתעסק בזיהוי פנימי ו-Labeling של הרובוט המזוהה. להלן סכמה בלוקים המסבירה את האלגוריתם:





איור 5 – סיסטם תוכנה

<u>PART A</u>

- **Taking Frame** - מדובר למעשה על ייבוא הפריים שממנו תתחיל כל פעולת עיבוד התמונה. השלב הזה הוא כמובן רק הבסיס הראשוני, שממנו ממשיכים לשלבים של עיבוד ועיבוד נוסף.
- **Convert to Bin** - המרה של התמונה לצבעים של שחור-לבן בעזרת סף בינארי (Binary Thresholding) . בשלב זה, מתבצע גם סינון בילטרלי שנועד לשמור על חדות הקצוות תוך הפחתת רעשים.
- **Get Edges** - יישום של אחד מהשלבים המרכזיים בעיבוד תמונה – זיהוי הקצוות, באמצעות השימוש באלגוריתם Canny edge detector . גלאי זה פועל על המסכה הבינארית שנוצרה בשלב הקודם (כלומר, תמונה שבה כל פיקסל מיוצג כערך בינארי – 0 לשחור ו-1 ללבן). האלגוריתם של Canny מזהה שינויים חדים

ברמות הבהירות שבתמונה, מה שמוביל לזיהוי הקצוות – אזורים שבהם קיים שינוי גדול בין פיקסלים סמוכים. בנוסף, כדי לזהות את קווי המתאר של האובייקטים שבתמונה, משתמשים בפונקציה findContours() שמבוססת על הספרייה OpenCV פונקציה זו מזהה קווי מתאר של עצמים בתמונה ומאפשרת עיבוד והבנת הצורות הגיאומטריות של אותם עצמים. במקרה זה, הפרמטר cv.RETR_EXTERNAL מתמקד בזיהוי קווי המתאר החיצוניים בלבד, תוך התעלמות מפרטים פנימיים, כגון חורים או קווי מתאר פנימיים בכדי לאפטם את זמן הריצה.

- **Geometric Filter -** מתבצע תהליך חשוב שנועד לייעל את המידע הגיאומטרי שהתגלה בתמונה. בשלב זה נעשה שימוש באלגוריתם Ramer-Douglas-Peucker (RDP) שהוא כלי חשוב לצמצום המורכבות של צורות גיאומטריות על ידי הפחתת מספר הנקודות שמרכיבות את קווי המתאר. האלגוריתם פועל כך שהוא בוחר רק את הנקודות הקריטיות שמאפשרות לשמור על צורת המתאר המקורית, אך תוך הקטנה משמעותית של מספר הנקודות שאותן צריך לשמור בזיכרון. זה מאפשר לעבד ולעבוד עם הצורה בצורה קלה יותר, תוך שמירה על דיוק מספק בתיאור המתאר. פעולה זו יעילה במיוחד כשמנסים לזהות אובייקטים בצורה מהירה ונקייה מרעש, מה שמאפשר שלב עיבוד מתקדם יותר, שבו ניתן לזהות את האובייקטים בצורה פשוטה יותר, כגון זיהוי של ריבועים, מעגלים, או כל צורה אחרת שנמצאת בתמונה.

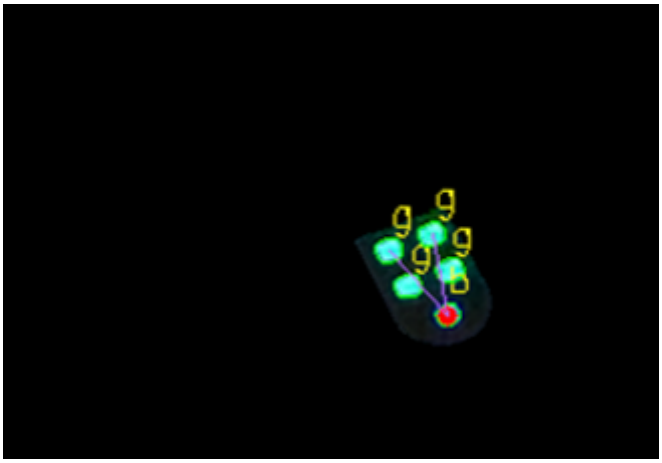
PART B

- **Masking + Erosion -** יצירת מסכה המתמקדת באזור הרלוונטי בתמונה, כלומר באזורים בהם קיימות נורות LED . המסכה עוזרת להפריד את הנורות מהרקע בתמונה. כדי לשפר את איכות המסכה ולהסיר גבולות דקים ולא חשובים, מבוצעת פעולה שנקראת erosion . פעולה זו מחלישה ומעלימה אזורים דקים שנמצאים סביב האובייקטים, כך שמתקבלים גבולות ברורים ומדויקים יותר של הנורות.
- **Convert to bin -** לאחר שהאזור הרלוונטי נמצא תחת המסכה, ממירים את התמונה לייצוג בינארי (שחור/לבן) באמצעות סף קבוע (Binary Fixed Thresholding) יחד עם שימוש בפילטר דו-צדדי (Bilateral Filter) השימוש בפילטר זה חשוב כיוון שהוא שומר על הגבולות החדים של האובייקטים תוך הסרת רעשים חזותיים מהתמונה. התמונה הבינארית המתקבלת משמשת כבסיס לזיהוי אובייקטים בשלב הבא.
- **Find LEDS shape -** בשלב זה, המערכת מזהה את הצורה של נורות ה LED בתמונה. הזיהוי נעשה על ידי שימוש בפונקציה לזיהוי קווי מתאר (Contours) שמאתרת את הגבולות החיצוניים של כל נורה. כאן משתמשים באלגוריתם cv.RETR_EXTERNAL המזהה רק את קווי המתאר החיצוניים של האובייקטים בתמונה, ומאפשר לנו לקבל את הצורה הברורה של כל נורה.
- **Find outsider LED -** לאחר שמצאו את קווי המתאר של הנורות, השלב הבא הוא לזהות את נורת ה- LED החיצונית ביותר מבין כולן. החישוב מתבצע באמצעות מדידה גיאומטרית של המרחק בין המרכזים של כל הנורות. נשתמש בפרמטרים הנקראים L1,L2 שמודדים את המרחקים הללו במרחב הגיאומטרי. על ידי מציאת הנורה הרחוקה ביותר ממרכזי האחרות, המערכת יכולה לזהות מי היא הנורה החיצונית כלומר ה-outsider.
- **Get color of each LED -** בשלב זה, המערכת מזהה את הצבע של כל נורת LED בנפרד. זה נעשה על ידי המרת התמונה למרחב הצבעים HSV (Hue, Saturation, Value) המותאם טוב יותר לזיהוי צבעים מאשר המרחב הסטנדרטי של .RGB לאחר מכן, המערכת מחשבת את המרחק הצבעוני בין כל נורה לבין הצבעים הדרושים, וזאת כדי לזהות במדויק את הצבע של כל נורה.
- **Label -** בסופו של דבר, לאחר שהצבעים של כל נורה זוהו ונמצאו, מתבצע תהליך סיווג (Classification) של האובייקט כולו. הסיווג מבוסס על שילוב הצבעים שהתקבלו עבור הנורות השונות ומאפשר למערכת להחליט איזה סוג של אובייקט מדובר בו, על פי התבניות שהוגדרו מראש.

מספר מילים על החלק של – "Find outsider LED":

כאמור לאחר שזהינו פוטנציאלית רובוט, באמצעות שימוש בקונפיגורציית הצבעים של ה-LEDS עלינו להבין מה מספר הסריאלי של הרובוט. מאחר ורצינו שהמערכת תהיה רובוסטית,לא רצינו להסתמך על חישובים גאומטרים שתלויים בתנאי סביבה ובכיולים (לדוגמה המרחק מהצלמה מהרובוט / כיול המצלמה), לכן מלבד הצבעים היה עלינו לזהות גם סדר הצבעים שמופיע ברובוט. כדי לעשות זאת עשינו טריק, והוא לנסות לזהות נקודות "ground reference" שממנו נוכל לזהות את סדר הצבעים, ולשם כך, נעזרנו בצורה של הרובוט, הצלחנו באמצעות חישוב מתמטי פשוט לזהות את החלון / LED שנמצא הכי

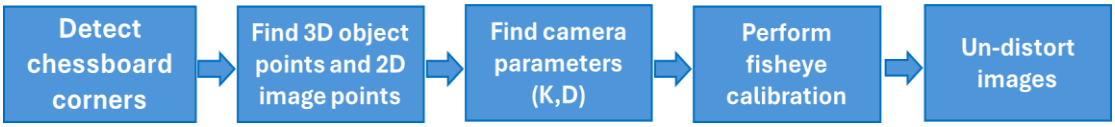
רחוק, עשינו זאת בתצורה המוסברת למעלה (שימוש במרחקים L1, L2), זה נראה כך בלייב:



איור 6 – באדום – זיהוי הOutsider שמהווה Ground Reffernce לסדר הצבעים לצורך Labeling

אנחנו רואים 2 מרחקים בצבע ורוד, ואלו הם L1, L2 איתם אנחנו מוצאים את ה-Outsider.

כיול מצלמה



- **Detect chessboard corners** - בשלב הראשון בקוד, אנו משתמשים בפונקציה cv2.findChessboardCorners כדי לאתר את פינות לוח השחמט בתמונות שצולמו, אם הן קיימות.
- **Find 3D object points and 2D image points** - כאשר הפונקציה מוצאת את הפינות, אנו מוסיפים את הנקודות המתאימות בתמונה הדו-ממדית (imgpoints) ובמרחב התלת-ממדי (objpoints) הנקודות התלת-ממדיות הן אותן נקודות במרחב האמיתי, והן מחושבות לפי הגודל הפיזי של לוח השחמט.
- **Find camera parameters (K,D)** - בשלב הבא, אנחנו מבצעים כיול באמצעות הפונקציה cv2.fisheye.calibrate הכיול הזה מוצא את הפרמטרים הפנימיים של המצלמה, מטריצת המצלמה K ומקדמי העיוות D מה שמאפשר לנו להבין כיצד העיוות משפיע על התמונה.
- **Perform fisheye calibration** - תהליך הכיול מתבצע תוך שימוש בפרמטרים הפנימיים והמקדמים, עם התמקדות מיוחדת בעיוות fisheye כאן השתמשנו בפונקציה cv2.fisheye.calibrate, המותאמת במיוחד לעדשות fisheye .
- **Un-distort images** - לאחר מציאת הפרמטרים K ו-D נשתמש בפונקציה cv2.fisheye.undistortImage כדי לתקן את התמונות ולעשות להן "un-distort" כך שהעיוותים יוסרו והתמונה תוצג בצורה נכונה.

5 ניתוח תוצאות

5.1 השוואות בין תוצאות הסימולציה לעבודה בזמן אמת

במהלך פיתוח האלגוריתם, השתמשנו בטכניקות שונות של ראייה ממוחשבת, והערכנו אותן הן במונחי ביצועים והן במונחי זמן עיבוד, עד שהגענו לאלגוריתם הנוכחי.

להלן טבלה המציגה את ההבדלים בין הטכניקות השונות:

Use Case	Name	Run Time	Perfromance
Thresholding	Binary Thresholding	Good	Good
	Adaptive Thresholding	Bad	Bad
	Otsu's Binarization	Medium	Good
Smoothing	Gaussian	Medium	Medium
	Median	Good	Bad
	Bilaterial	Bad	Good
Getting Edges	Simple Sobel	Good	Medium
	Laplacian of Gaussian	Medium	Medium
	Morphological (Open - Close)	Good	Bad
	Canny Edge Detector	Medium	Good
Getting Countors	Suzuki-Abe algorithm (cv2.findContours())	Good	Good
	Hough Transform	Bad	Bad
Shape Detection - Orientation	Detecting the arch using approxPolyDP	Bad	Medium
	Minimum Area Rectangle	Good	Bad
	Maximum L1+L2 Geometric distance of centeroids	Good	Good
Shape Labeling - Detecting Leds Colors	Simple HSV colors range	Good	Bad
	Minimum color HSV distance from RGB	Medium	Good

טבלה 1 - טבלת תוצאות

מתוך התוצאות של טבלה זו בחרנו להשתמש בשיטות המסומנות בירוק שכן הביאו לתוצאות הכי טובות ומהירות.

6 סיכום, מסקנות והצעות להמשך

הפרויקט עבר שינויים רבים לאור התקופה והמצב אך בסופו של דבר הצלחנו לייצר מערכת עקיבה שמביאה תוצאות ומיקום בזמן אמת בעבור ברקוד אלקטרוני שמשנה קונפיגורציית צבעים כל כמה שניות וזאת בכדי לדמות מספר רובוטים.

נקודות לשיפור להמשך:

- במקור רצינו לממש את הפרויקט על RPI3 אם כי לאחר כמה ניסיונות הוא היה איטי מידי ולכן נמליץ להשתמש בRPI5 שהוא מהיר וחזק יותר.
- שינוי צורת הברקוד לצורה אותה יותר קל לזהות ע"י האלגוריתם (לדוגמא עיגול וקו שמסמן כיווניות).
- ביצוע אופטימיזציה לקיצור זמן ריצה כך שיוכל לרוץ גם על RPI3.
- להשתמש בלדים בעלי אותו Brightness + להשתמש בPWM כדי לעמעמם אותם.
- Caching של מיקום הרובוט בכדי לחסוך בחישובים.
- לצורך זיהוי האוביקט, ניתן לבצע טריק, בגלל שיש רשותנו לדים, נוכל לשלוח פינג בתדר מסוים בכדי לתת אינדקציה על מיקומו של האוביקט.

7 תיעוד הפרויקט

git clone https://github.com/Sloboo/IndoorLocalization.git

- מקורות

קישורים למקורות באינטרנט:

[1] "Suzuki's Contour tracing algorithm OpenCV-Python", https://theailearner.com/tag/suzuki-contour-algorithm-opencv/

[2] "Adaptive Thresholding", https://theailearner.com/tag/adaptive-thresholding/

[3] "Calibrate fisheye lens using OpenCV" , https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0

[4] A recording of the algo running in live - https://vimeo.com/1011770060?share=copy