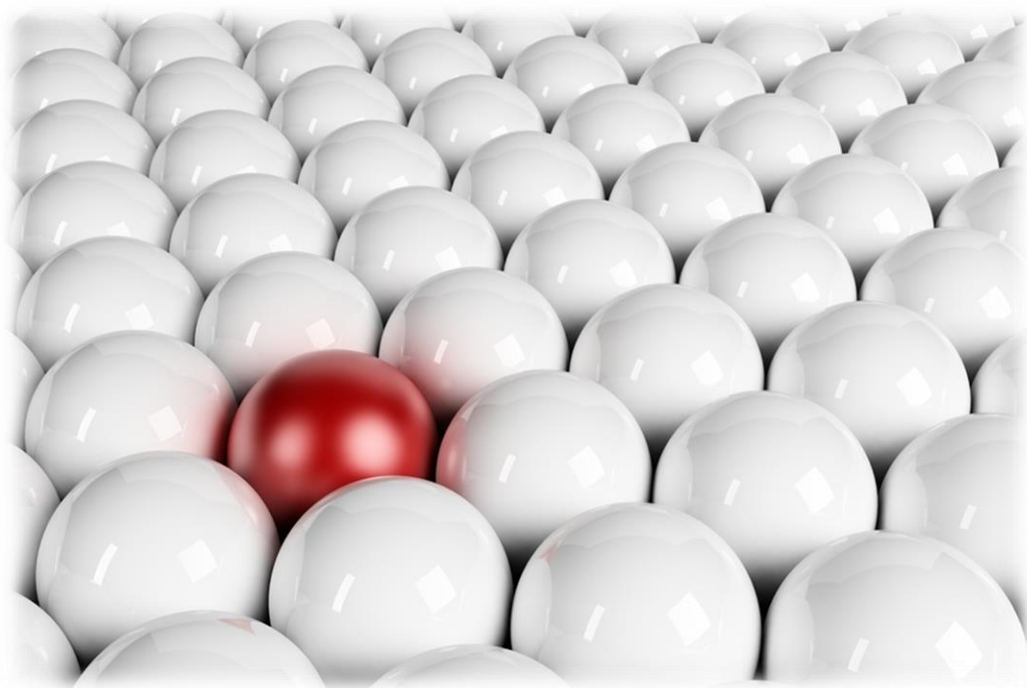


רוביליארד

ספר פרויקט באלקטרוניקה



מגישים: אסרף עידן, טרוק ברק.
מנחים: אדלר אריה ואמסלם רפי.
בית ספר: גינסבורג האורן.
תאריך: 23/06/2013.

תוכן עניינים

2	הרעיון
5	עיבוד תמונה
10	כללי עבודה בקוד
15	מבנה התוכנית
27	פונקציות פנימיות
48	פונקציות חיצוניות
72	תוכניות עזר
80	סיכום

הרעיון

בחלק זה נדבר על הרעיון שמאחורי הפרויקט, ועל תהליך הלמידה שקדם לו:

- ❖ מהיכן הגיע הרעיון לבנות רובוט שמשחק ביליארד?
- ❖ במה רעיון זה היה כרוך?
- ❖ כיצד למדנו עיבוד תמונה?
- ❖ למה התוכנית כתובה דווקא ב-Matlab?
- ❖ ועוד.

מהו הרעיון מאחורי הפרויקט?

הרעיון מאחורי הפרויקט הוא בנייה של תוכנה העושה שימוש בעיבוד תמונה, אשר מדריכה ושולטת על רובוט המותאם למשחק ביליארד. כלומר – המטרה היא בניית רובוט שמשחק ביליארד, אשר מונחה ע"י תוכנה העושה שימוש בעיבוד תמונה.

הרובוט מחובר למחשב באמצעות תקשורת טורית אלחוטית (RS-232 to Wifi), עליו רצה תוכנת השליטה, הכתובה בשפת Matlab מבית חברת Mathworks.

הפרויקט הוא משותף לשני זוגות:

- ❖ אנו (עידן וברק) – אחראים על כתיבת התוכנה הרצה על המחשב.
- ❖ טל ויואב – אחראים על בנייתו של הרובוט וכתיבת הקוד ב-VHDL, האחראי על תנועתו של הרובוט, ועל פירוש ההוראות הנקלטות מהמחשב.

החלק שלנו (ברק ועידן) בפרויקט, הוא תכנון התוכנית העושה שימוש בעיבוד תמונה, כדי לזהות את מצב שולחן המשחק, לחשב את המהלכים האפשריים, ולהדריך את הרובוט כיצד לבצעם.

מהיכן הגיע הרעיון לפרויקט?

בתחילת השנה, לאחר שסיימנו ללמוד את הבסיס של שפת VHDL, הגיע הזמן שכל תלמיד במגמה יתחיל לעבוד על פרויקט הגמר שלו, ולכן, ישבנו כולנו בכיתה עם המורה רפי, והעלנו רעיונות לפרויקטים שניתן לבנות.

היו הרבה הצעות לפרויקטים, אך הצעה אחת משכה את תשומת ליבנו כמעט מיידית – יואב, תלמיד בכיתה, הציע לבנות רובוט שישחק ביליארד. רפי מאוד התלהב מהרעיון, אך אמר כי לשם כך יש צורך בעיבוד תמונה כדי שהפרויקט יוכל לעבוד.

מאחר ואנו (ברק ועידן) ספרנו לרפי עוד קודם לכן כי אנו רוצים לעסוק בעיבוד תמונה בפרויקט שלנו, הוא הציע לנו לעבוד ביחד עם טל ויואב על הרעיון שלהם, ולבנות להם את התוכנה לעיבוד תמונה שתשלוט על הרובוט אותו הם יתכננו.

הסיבה שרצינו לעבוד על פרויקט של עיבוד תמונה, היא ששנינו מאוד אוהבים הנדסת תוכנה, שנינו התעניינו מאוד בתחום של עיבוד תמונה, ולשנינו נאמר שבפרויקט יש הרבה מתמטיקה – מקצוע שאנו אוהבים ללמוד.

וכך יצא שאנו (ברק ועידן) התחברנו לצוות של יואב וטל, והתחלנו לעבוד יחד על פרויקט משותף.

כיצד למדנו עיבוד תמונה?

על-אף שלי ולברק ידע רב בהנדסת תוכנה, בתחילת שנה לא ידענו כמעט דבר על ענף העיבוד תמונה, ולכן היה עלינו ללמוד כיצד מעבדים תמונה וקולטים ממנה מידע.

לכן, התחלנו ללמוד עיבוד תמונה בבית ובמעבדה בזמן השיעורים המוקדים לבניית הפרויקטים.

כמו-כן, המורה רפי קבע שיעורי עזר מיוחדים לכל התלמידים בכיתה שבחרו לעסוק בעיבוד תמונה, שבמהלכן הוא לימד אותנו על מבנה התמונה, על פילטרים מועילים, ועל שיטות שונות לאיסוף מידע מהתמונה.

השיעורים התקיימו כמעט בכל יום במהלך השבוע, במעבדה, ומשעה שבע-וחצי בבוקר עד השעה שמונה ועשרה, למשך כחודש בערך. היינו מחויבים לשיעורים אלו, למרות עומס השעות של יב', והשתדלנו להגיע לכל השיעורים.

הידע והניסיון שלנו בנושא גדלו יחד עם כתיבת הפרויקט, והוא עבר מספר רב של שיפורים ושינויים לשם העצמת יעילותו ומהירותו, ובשלב מסוים אף כתבנו את הפרויקט (שהיה כבר גמור) מחדש.

למה Matlab?

הנדסת תוכנה הכוללת עיבוד תמונה דורשת מסגרת תוכניתית המאפשרת ומספקת כלים לנושא זה.

לכן, המורה הכיר לנו את שפת (וסביבת העבודה) Matlab מבית חברת Mathworks. שפה זו מיועדת לחישובים מבוססי מטריצות ומציגה תמיכה וכלים המיועדים לעיבוד תמונה.

רפי הסביר לנו את הרעיונות הבסיסיים מאחורי העבודה עם השפה, ודי במהירות למדנו כיצד לעבוד איתה ולבצע איתה דברים יותר מורכבים.

כמו-כן, רפי הכיר לנו את הספרייה הפתוחה לעיבוד תמונה OpenCV, אשר מותאמת למספר רב של שפות תכנות, ומאפשרת (תחת שפת C) לכתוב תוכנות שרצות בצורה עצמאית, מה שלא מתאפשר בשפת Matlab מאחר וזוהי שפת סקריפטינג, העושה שימוש במפרש ולא במהדר.

כמו-כן, מאחר והספרייה קיימת עבור שפת C, המהירות בה ניתן להריץ קטעי קוד המכילים עיבוד תמונה היא מדהימה, והספרייה היא המותאמת ביותר לעיבוד תמונה בזמן אמת (Real-time Image Processing). כמו-כן, ספרייה זו היא העיקרית בה נעשה שימוש בתעשייה.

למרות שהספרייה מצוינת, ומאפשרת עיבוד תמונה ברמה הגבוהה ביותר, באותו זמן (תחילת השנה), הידע שלנו בעיבוד תמונה היה מאוד מצומצם, ולכן היה לנו מאט קשה להבין כיצד לעבוד איתה, ולכן החלטנו בסופו של דבר לעבוד עם שפת Matlab.

כיום, לאחר שצברנו הרבה ידע בנושא עיבוד התמונה, אנו מבינים יותר את שיטת העבודה עם OpenCV, ונשמח לעבוד איתה ולעשות בה שימוש בעתיד.

ספריית עיבוד תמונה נוספת בה נתקלנו בשלב מאוחר של הפרוייקט, היא ספריית SimpeCV לשפת הסקריפטינג Python. לעידן מעט ידע בפייטון, ולכן התנסנו מעט בשימוש בספרייה זו, אך מאחר ובשלב זה כבר היינו קרובים לסיום התוכנית ב-Matlab, החלטנו לוותר על השימוש בספרייה זו.

למרות-זאת, מדובר על ספרייה מצוינת שמאפשרת להשיג דברים מדהימים במספר קצר מאוד של שורות קוד, ונהיננו לשחק איתה וליצור איתה יישומים פשוטים.



עיבוד תמונה

הקדמה קצרה אודות מהי תמונה, ומהו תחום העיבוד תמונה.
בחלק זה נסביר כיצד אנו מזהים עצמים בתמונה, כיצד תמונה בנויה, ומהם השימושים לעיבוד תמונה בתעשייה.

מהי תמונה?

הפרויקט שלנו מבוסס על הענף בתחום ההנדסה הקרוי "עיבוד תמונה", או באנגלית – "Image Processing".

אך לפני שנצלול אל תוך הקוד, ונסביר בדיוק איך כל פונקציה עובדת, עדיף שנסביר קודם מה זה למעשה "עיבוד תמונה", ואפילו לפני-כן – מהי תמונה?

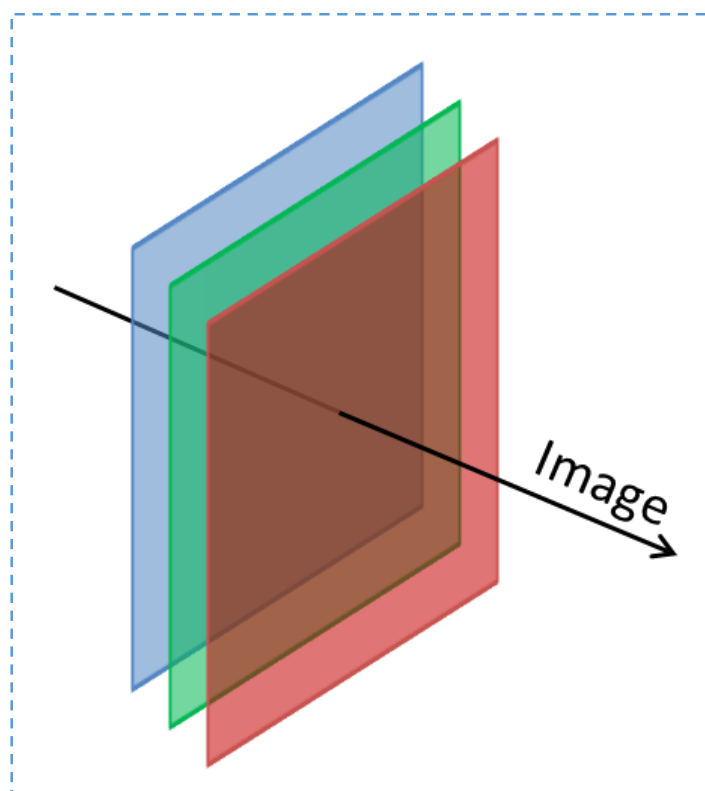
תמונה ע"פ ההגדרה היום-יומית, היא:

"המציאות התלת-ממדית כשהיא מוצגת על משטח דו-ממדי כגון ציור או צילום" (מקור – ויקימילון).

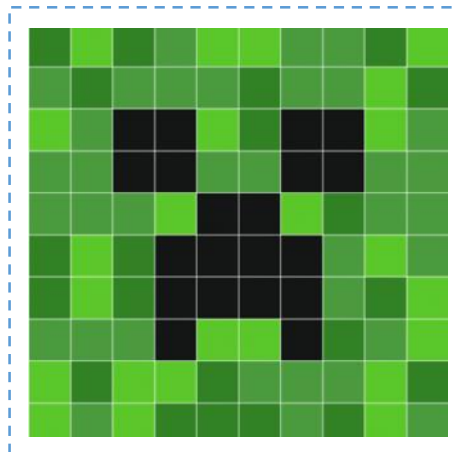
אך ההגדרה של המושג תמונה בעולם המחשוב היא מעט מורכבת יותר:

תמונה היא וקטור של מטריצות בעלות גדלים זהים, המכילות את המידע אודות הצבעים בתמונה. ישנם מספר פורמטים לשמירת ערכי הצבעים של פיקסל, כאשר הפורמט העיקרי בו נעשה שימוש בתעשייה הוא RGB – אדום, ירוק, כחול, והוא היחיד שניתן לראות פיסית.

את הפורמט הנ"ל ניתן לראות בקלות כאשר מתקרבים אל צג המחשב – ניתן להבחין בנקודות בצבעי אדום-ירוק-כחול המרכיבות את התמונה.



תמונה מורכבת מאוסף של נקודות הקרויות "פיקסלים" (Pixel), וכל פיקסל הוא בעל צבע מסוים הנקבע ע"פ התאים החופפים המתאימים לו במטריצות.



מהו ענף העיבוד תמונה?

עיבוד תמונה הוא ענף בתחום ההנדסה שעוסק בעיבוד של תמונות, במניפולציה עליהן, ובאסיפת מידע רלוונטי מתוכן.

ענף זה משפיע ותורם בתחומים רבים בתעשייה:

- ❖ קיימות מערכות העושות שימוש בעיבוד תמונה למציאת פגמים בשבבים.
- ❖ מכשירים SRI, מכשירים למדידת דופק, ועוד כלים רפואיים רבים עושים שימוש בכוח של עיבוד תמונה, בכדי להגיע לאסוף מידע רב ואמין אודות חולים ונבדקים.
- ❖ קונסולות המשחק הנוכחיות הביאו את עיבוד התמונה לסלון, והמשחקים עושים שימוש במצלמות מתקדמות כדי לספק לשחקן חוויית משחק חדשנית ומלאת תנועה.
- ❖ ועוד...

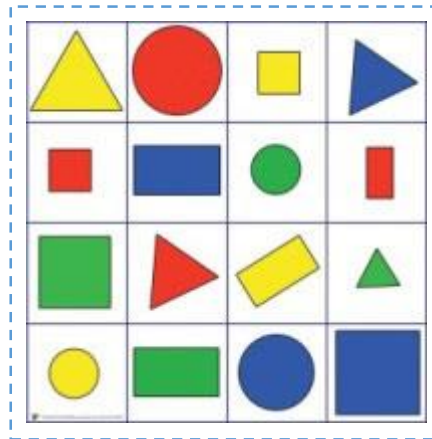
כיצד ניתן לאסוף מידע מתוך תמונה?

כפי שכבר אמרנו, תמונה היא וקטור של מטריצות, והיא מחולקת לפיקסלים, ולכל פיקסל ערכים המגדירים את צבעו.

בעזרת העברתם של פילטרים שונים, ניתן: להשפיע על הצבעים בתמונה, על חדות התמונה, לטשטש תמונה, למצוא צורות מסוימות ועוד הרבה יותר.

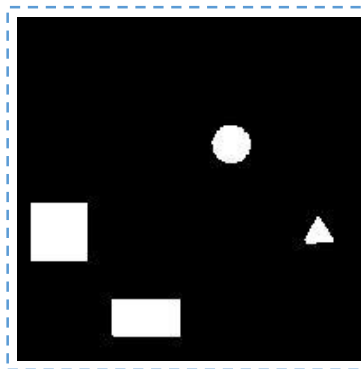
בכדי לזהות נוכחות של צבע בתמונה, ניתן לסנן את הפיקסלים בתמונה לפי הערכים הקובעים את צבעם, כך שמתקבלת תמונה שהיא למעשה מפה בינארית של נוכחותו של צבע זה בתמונה.

נניח לדוגמא שאנו רוצים למצוא את הריבוע הירוק בתמונה הבאה:



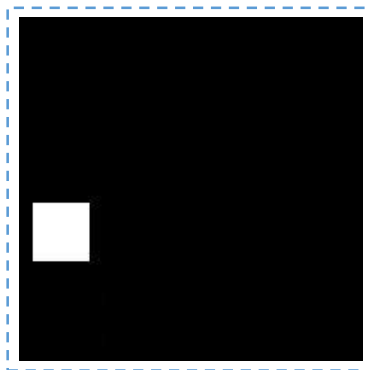
למעשה, נתחיל בכך שנמצא את נוכחות הצבע הירוק בתמונה, כפי שהסברנו זה עתה: נגדיר כי הפיקסלים שערכם יהיה "אמת", יהיו פיקסלים שערך הירוק אצלם גבוה, ואילו ערכי האדום והכחול נמוכים (בשיטת RGB).

המפה הבינארית שתתקבל תראה כך:



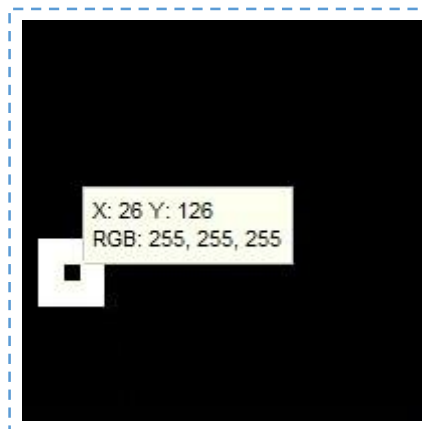
לאחר-מכן, נריץ על התמונה פילטר (או בשפה המקצועית, נבצע קונבולוציה של פילטר על התמונה) בצורת ריבוע. הרצתו של הפילטר על התמונה תחזיר מפה בינארית חדשה, שבה רק הריבוע יופיע, ושאר הגופים הירוקים ימחקו.

כך נראית המפה הבינארית המפולטרת:



אם נרצה למצוא את מיקום מרכזו של הריבוע הירוק בתמונה, על-ידי הרצה של אלגוריתם חישובי הקרוי Centroid, ניתן למצוא בקלות את המרכז של כל גוף או צורה בתמונה בינארית.

מרכז הריבוע שמצאנו, בתמונה:



כעת, לאחר שהסברנו מעט אודות מבנה התמונה, ואסיפת מידע מתמונה, הגיע הזמן שניגש לתוכנית שכתבנו.

כללי עבודה בקוד

בכדי לכתוב קוד גמיש, קריא, מסודר ויעיל, החלטנו על סטנדרטים וכללי עבודה בקוד. כללים אלו נוגעים בצורה בה הקוד כתוב, באלגוריתמים החישוביים מאחורי התוכנית, בפורמטים מוכרים ועוד.

בחלק זה של הספר נרחיב על כללי העבודה והסטנדרטים השונים בקוד, והוא יהווה רקע והקדמה לפני שנציג ונסביר את הקוד עצמו. בחלק זה יוסברו מספר מושגים שיופיעו פעמים רבות לאורך הספר.

מיקומים של גופים כמבנים (struct):

אחד הסטנדרטים השימושיים והשמישים ביותר בקוד, הוא סטנדרט הנוגע לאופן בו מיקומים של גופים בתמונה שמורים בתוכנית:

קבענו כי המיקומים ישמרו בצורה של מבנה שאותו הגדרנו ולו קראנו "Pos Struct", המכיל שני משתנים: ערך ה-X וערך ה-Y של מיקום הגוף. סטנדרט זה תורם לסדר בתוכנית ולהבנת התחביר של הפונקציות השונות המרכיבות אותה.

שמות של משתנים ופונקציות:

לשם שמירת הסדר בקוד, החלטנו מראש על הפורמטים בהם נקבעים שמותיהם של משתנים ופונקציות בתוכנית:

- ❖ בשמות משתנים – מחליפים רווח בקו תחתון ושומרים על שמות קצרים ובאותיות קטנות.
- ❖ בקבועים – אותיות גדולות בלבד, שמם קו תחתון במקום רווח (Capital letters).
- ❖ בפונקציות – אין רווחים, כל מילה מתחילה באות גדולה, מלבד המילה הראשונה.

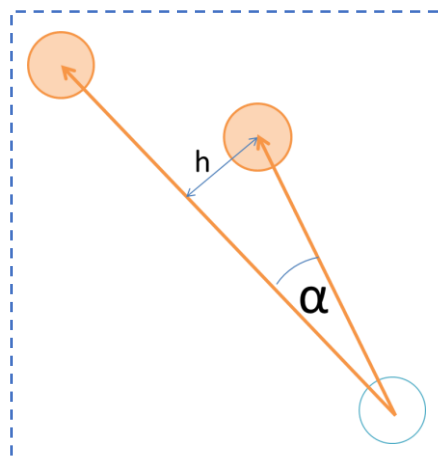
שימוש בווקטורים ומטריצות לשם יעילות התוכנית:

השפה בה כתבנו את התוכנית היא שפת סקריפט (Scripting) בשם Matlab שפותחה ע"י חברת Mathworks, ובנויה על גבי השפות C++ ו-Java.

מאטלאב מתמחה בחישובים מבוססי וקטורים ומטריצות (ומכן שמה: Mat = Matrix), והיעילות והמהירות הרבה בהן היא מבצעת חישובים על משתנים מסוגים אלו, היא ייחודית ורצינית ביותר.

מאחר והתוכנה שלנו עושה שימוש בעיבוד תמונה בזמן אמת, יש לוודא כי החישובים אותם מבצעת התוכנה הם מהירים ככל הניתן, כך שמספר הבדיקות בשנייה יהיה גדול ככל האפשר.

הצורך במהירות החישובים הביאו לכך שהחלטנו כי החישובים בתוכנית יהיו מבוססי וקטורים ומטריצות, איפה שרק ניתן, וכך יצא שרוב התוכנית והפונקציות הקשורות אליה עושות שימוש בעיקר בווקטורים.



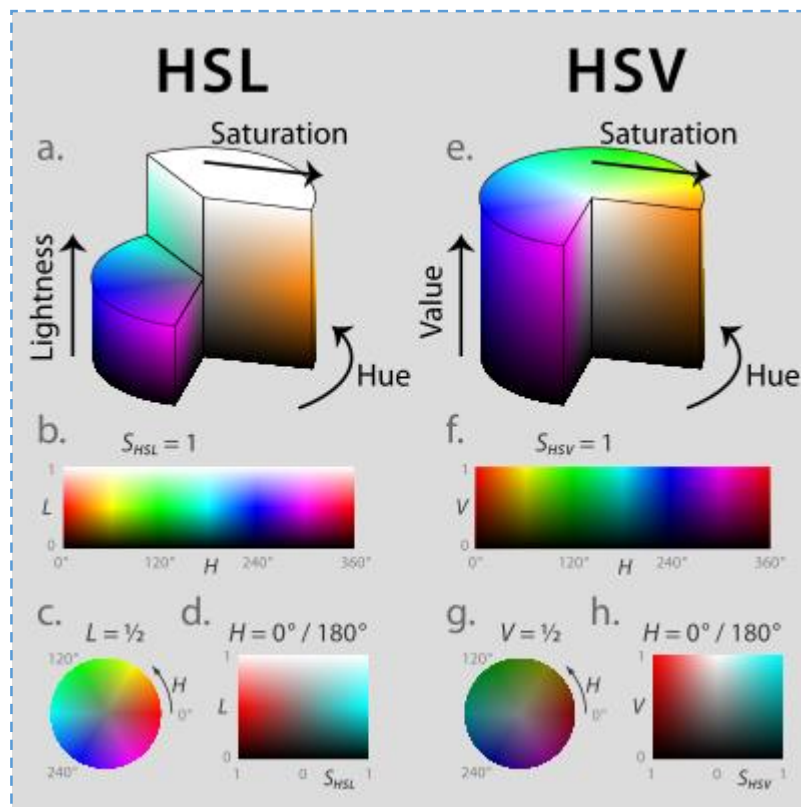
שימוש בפורמט HSV במקום בפורמט RGB:

הפורמט המוכר והשמיש ביותר בעולם ההנדסה לשמירת ערכי הצבעים של פיקסלים בתמונה הוא פורמט ה-RGB. בפורמט זה כל צבע מיוצג על-ידי שילוב של רמות צבע של שלושה צבעי בסיס: אדום, ירוק וכחול.

הבעייתיות שבשימוש בפורמט זה, היא ששינויים קלים ביותר בתאורה גורמים להבדלים עצומים בערכי ה-RGB של הפיקסלים בתמונה, וזה עניין מאוד בעייתי בעיבוד תמונה: במידה והערכים בתמונה הנקלטת שונים מהערכים השמורים בתוכנה, הזיהוי של נוכחות הצבע בתמונה לא יהיה מדויק.

לכן, החלטנו לעבוד עם פורמט אחר לשמירת ערכי הצבע של פיקסלים בתמונה, והוא פורמט ה-HSV. בפורמט זה הצבע של פיקסל בתמונה נשמר ע"י שלושה ערכים: גוון, עוצמה ובהירות.

הפורמט מתואר בבירור בתרשים הבא:



היתרונות של פורמט זה הם שמדובר בפורמט שבו כל צבע הוא חד-חד-ערכי, ושכאשר התאורה משתנית הגוון נשאר קבוע, מה שמקל מאוד על זיהוי של צבע מסוים במספר רמות תאורה. יתרון נוסף הוא שניתן לחזות בקלות את ערכי ה-HSV של צבע מסוים ברמת דיוק סבירה.

מאחר וראינו כי לפורמט ה-HSV יש רק יתרונות על שיטת ה-RGB, החלטנו כי זהו הפורמט בו נעבוד.

מסד הנתונים לשמירת דגימות הצבעים:

מאחר וערכי העוצמה והבהירות של צבע משתנים בהתאם לתאורה בה צולמה התמונה, ובעקבות קשיים בזיהוי תקין של הצבעים, החלטנו לשמור דגימות של כל אחד מהצבעים בהם אנו עושים שימוש.

מאחר ויש צורך שהדגימות ישמרו בצורה מסודרת, ושיהיה קל לגשת אליהם ולטפל בהם, החלטנו כי את הדגימות עבור כל צבע נשמר בטבלה על מסד נתונים מבוסס MySQL.

שפת SQL היא קלה ונוחה לעבודה עם טבלאות וערכים רבים, ומאחר ומדובר על שפה גמישה ופונקציונאלית ניתן להריץ שאילתות (queries) לשם קבלה של מידע בעל ערך רב מתוך הטבלאות.

כמו-כן, היותה של MySQL פלטפורמה פתוחה וחינמית לשפת SQL, אפשר לנו להקים את מסד הנתונים על שרת אכסון חינמי, בתוך זמן קצר.

מסד הנתונים מכיל את דגימות הצבעים שבהם אנו נעזרים על מנת לזהות צבע מסוים בתמונה, כאשר כל צבע שמור בטבלה שנקראת על שמו. בכל טבלה קיימים הגבולות התחום של צבע, אותם גילינו ע"י הדגימות (שאופן לקיחתן מוסבר בחלק "תוכניות עזר").

טבלה של הצבע מגנטה (Magenta) לדוגמה:

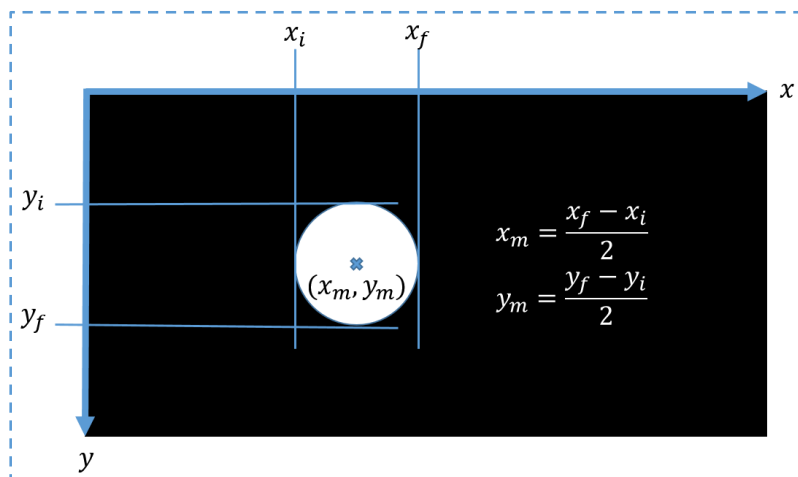
	n	h_min	s_min	v_min	h_max	s_max	v_max
1	0.872222	0.453125	0.443137	0.955399	0.616667	0.54902	
2	0.872222	0.496815	0.607843	0.955399	0.653179	0.705882	
3	0.872222	0.453125	0.443137	0.955399	0.660819	0.705882	
4	0.872222	0.50303	0.6	0.955399	0.653179	0.698039	
5	0.872222	0.5125	0.611765	0.955399	0.627219	0.690196	
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

להשתמש בכמה שפחות פונקציות מוכנות מראש:

החלטנו כי את כל הפונקציות שבהן אנו עושים שימוש נכתוב בעצמנו, עד כמה שאפשר.

ישנה פונקציה אחת מוכנת שכן עשינו בה שימוש במהלך התוכנית שלנו וזאת הפונקציה Centroid: פונקציה שמוצאת את מרכזי הגופים בתמונה בינארית.

אין בעיה בלכתוב פונקציה שתמצא את המרכז של גוף אחד בתמונה בינארית. הצלחנו לכתוב פונקציה שתתפקד כמו Centroid אך היא הייתה איטית ולא מדויקת ולכן נשארו עם הפונקציה המוכנה שבאה עם המאטלאב. בשביל למצוא מרכז גוף צריך בסה"כ את הגבולות שלו בתמונה:



התקשורת בין הרובוט למחשב

התוכנית מתקשרת עם הרובוט באמצעות תקשורת טורית, המתבצעת דרך ערוץ התקשורת הטורית במחשב (RS-232 Port). על שיטת התקשורת הזו המליץ לנו המורה, וקלה מאוד לשימוש, ולכן החלטנו להשתמש בה.

התוכנית שולחת מילים (הוראות) בקצב של 115200 באוד (Baud-rate), שאורכן שמונה סיביות (8Bit), כך שקיימות מאתיים-חמישים-ושש מילים אפשריות לשליחה.

מאחר ואיננו רוצים שהרובוט יהיה מוגבל בתנועתו בגלל כבל, אנו משתמשים במשדר ומקלט אלחוטיים (RS-232 to WiFi), המספקים תקשורת אמינה גם בטווחים ארוכים.

תמונה של הרכיב:





מבנה התוכנית

כאן נסביר את כל חלקיה של התוכנית, כיצד היא פועלת וכיצד היא מתוכננת.

כמו-כן, את כל הקבצים הקשורים לתוכנית ולפונקציות ניתן למצוא בכתובת: <http://bit.ly/1a1vjkD>.

Main Function (Robilliard)

מבנה התוכנית:

```
function Robilliard(src_img)
    % Connect to database:
    server_conn = connectToDatabase();

    % Constant (global) values
    global MAX_DEGREE;
    global BALLS_RADIUS;
    global server;
    global COM1;

    MAX_DEGREE = 70;
    server = server_conn;
    COM1 = openSerialBus();

    if (nargin ~= 0)
        frame = src_img;
    else
        % Setuiping the camera.
        global source_capture;
        source_capture = setupCamera(1);
        start(source_capture);
        pause(10);

        frame = isMoving(source_camera, 15);
    end

    % << Game loop starts here!!! >>
    % Detecting the magenta colored stickers.
    [hf,sf,vf] = getHSV(frame);
    crop_cords = findItems (server, hf,sf,vf, 'magenta', 2.5);
    if (size(crop_cords, 2) ~= 2)
        error('ERROR: couldn't detect stickers properly.');
```

```
end
```

```
% Cropping out the table out of the frame.
width = abs(crop_cords(2).x - crop_cords(1).x);
height = abs(crop_cords(2).y - crop_cords(1).y);
table_image = imcrop(frame, [crop_cords(1).x, crop_cords(1).y,
width, height]);
```

```

% Recalculating the HSV values (of the table image).
[h,s,v] = getHSV(table_image);

% Finding the location of the holes.
holes_pos = findItems (server, h,s,v, 'black', 2.5);
if (size(holes_pos, 2) ~= 6)
    error('ERROR: the holes were not detected properly.');
```

end

```

% Finding the locaion of the white ball.
[white_ball_pos, BALLS_RADIUS] = findItems (server, h,s,v,
'white', 2.5);
if (size(white_ball_pos, 2) ~= 1)
    error('ERROR: white ball was not detected properly.');
```

end

```

% Finding the location of the red balls
red_balls_pos = findItems (server, h,s,v, 'red', 2.5);
num_of_red_balls = size(red_balls_pos, 2);

% Display the number of red balls in the console.
message = sprintf('There are %d red balls.', num_of_red_balls);
disp(message);

% Calling the strategy-list building function.
[norm_list, alt_list] = buildStrategiesLists
(white_ball_pos,...
    red_balls_pos, holes_pos, num_of_red_balls);

% Sorting out the normal strategies list by value.
norm_list = sortStrategyList(norm_list);

%% Drawing out the different strategies.
norm_handle = figure('Name', 'Normal Strategies');
imshow(table_image);

alt_handle = figure('Name', 'Alternative Strategies');
imshow(table_image);

drawStrategiesList(norm_list, norm_handle);
drawStrategiesList(alt_list, alt_handle);

if (nargin == 1)
    disp('Done debugging the function.');
```

return;

end

```

% Guiding the robot.
[robot_pos, ~, arm_length] = getRobot(hf,sf,vf);
```

```

while(1)
    try
        temp = size(norm_list(1).plan,2)-1;
        white_target_pos = norm_list(1).plan(temp);

        catch
            error('ERROR: Strategy list is empty.');
```

```

        end

        hit_white_pos = calcHitPoint(white_ball_pos,
white_target_pos, 2*BALLS_RADIUS);

        target_pos = calcHitPoint(hit_white_pos,white_target_pos,
arm_length);

        hit_white_pos = normalizePos(hit_white_pos, crop_cords(1));
        target_pos = normalizePos(target_pos, crop_cords(1));

        [target_side, target_side_id] = findSide(target_pos,
crop_cords(1), crop_cords(2));

        if (strcmp(target_side, 'in_frame'))
            norm_list(1) = [];
            continue;
        end

        break;
    end

    [~, robot_side_id] = findSide(robot_pos, crop_cords(1),
crop_cords(2));

    num_sides = target_side_id - robot_side_id;
    if (num_sides < 0)
        num_sides = num_sides + 4;
    end

    %Report to the robot how many sides he needs to go.
    controlRobot(num_sides, 3);

    waitForRobotReport();

    vec_line = buildVector(target_pos, hit_white_pos);
    target_line = buildLine(target_pos, hit_white_pos);

```

```
directRobot(target_line, target_pos, vec_line);  
  
controlRobot('hit', 0);  
  
% << Game loop ends here >>  
  
stop(source_capture);  
  
end
```

לקריאה עיונית:

. <http://pastebin.com/jfYXbHNX>

הסבר על התוכנית:

הפונקציה הנ"ל הינה הפונקציה הראשית של התוכנית, כלומר – פונקציה זו היא שרצה בעת הפעלת התוכנית, ממנה מתבצעות הקריאות לשאר הפונקציות הקשורות לתוכנית, והיא האחראית לתיאום בין פונקציות אלו.

מאחר וזו הפונקציה הראשית של התוכנית, מעתה והילך נקרא לה בשם זה – "התוכנית".

התוכנית מקבלת בהפעלתה ארגומנט אופציונאלי אחד ויחיד: תמונה. בתמונה זו נעשה שימוש כאשר אנו רוצים לנפות באגים בתוכנית (Debug) מבלי לעשות שימוש במצלמה.

במידה והתוכנה אינה מקבלת תמונה בהפעלתה, היא תגדיר ותכין את המצלמה לשימוש, ותרוץ באופן רגיל (שימוש במצלמה ללכידת פריימים).

במהלך הסבר אופן פעולת התוכנית נעשה שימוש בדוגמאות, הממחישות את אופן פעולת החלקים השונים בתוכנית.

כל הדוגמאות מבוססות על התמונה הבאה:



הכרזה והגדרה של משתנים גלובאליים –

תחילה, התוכנית מכריזה ומגדירה את המשתנים הסטטיים (קבועים) בתוכנית, כמשתנים גלובאליים. כלומר – כמשתנים שניתן לגשת אליהם מכל מקום בתוכנית. המשתנים הסטטיים בתוכנית הם:

- ❖ **Server** – מכיל את אובייקט החיבור לשרת מסד הנתונים, בו נשמרות דגימות הצבעים.
- ❖ **Source_capture** – משתנה גלובאלי זה מכיל את אובייקט המצלמה, מוגדר ומוכן לשימוש.
- ❖ **COM1** – אובייקט החיבור לערוץ התקשורת הטורית במחשב (RS-232), לשליטה על הרובוט.
- ❖ **MAX_DEGREE** – הזווית המקסימאלית המותרת בין וקטורי התנועה של הכדורים.
- ❖ **BALLS_RADIUS** – הרדיוס של הכדורים, אותו נגלה ונשמור בהמשך התוכנית.

משתנים אלו מוגדרים כגלובאליים מאחר ונעשה בהם שימוש רחב לאורך כל הפונקציות בקובץ הראשי, והגדרתם ככאלו חוסכת בזיכרון ושומרת על המראה המסודר של הקוד.

קליטת פריים לעיבוד –

לאחר מכן, התוכנית בודקת האם התקבלה תמונה כארגומנט:

- ✓ במידה וכן, התוכנית תיכנס למצב ניתור באגים, ותעשה שימוש בתמונה שספקנו - התוכנית לא תגדיר ולא תעשה שימוש במצלמה במהלך התוכנית. במצב זה אין כלל חיבור לרובוט.
- ✗ במידה ולא, התוכנית תגדיר את המצלמה ותרוץ כרגיל.

כאשר התוכנית איננה במצב של ניתור באגים, בשלב זה היא תיצור את אובייקט המצלמה.

לאחר מכן, התוכנית תוודא כי הכדורים על השולחן אינם נעים, בעזרת שימוש בפונקציה החיצונית לזיהוי תנועה (isMoving) – במידה והכדורים נעים התוכנית תמתין עד שאלו יפסיקו לנוע.

על הפריים (תמונה) המוחזר מפונקציה זו, בו הכדורים במצב סטטי, יעשו כל הבדיקות והחישובים בתוכנית, לשם מציאת אסטרטגיות המשחק האפשריות במצב הקיים.

גזירת השולחן מתוך התמונה:

מאחר ואנו מעוניינים כי החישובים יעשו רק על הגופים (כדורים) הנמצאים על השולחן (ובשטחו), בתחילה עלינו לגזור את השולחן החוצה מן התמונה.

בתמונה הגזורה יופיע רק שטחו של השולחן, וכך נבטיח כי אף גוף אשר נמצא מחוץ לשטחו לא יפריע לדיוק ונכונות החישובים.

כדי לגזור את השולחן התוכנה עושה שימוש בשתי מדבקות בצבע מגנטה (Magenta), אשר מודבקות על שתי פינות מנוגדות של השולחן. התוכנה מוצאת את מיקומן של מדבקות אלו (בתמונה המקורית), ובעזרתן אנו גוזרים את השולחן החוצה מן התמונה המקורית.

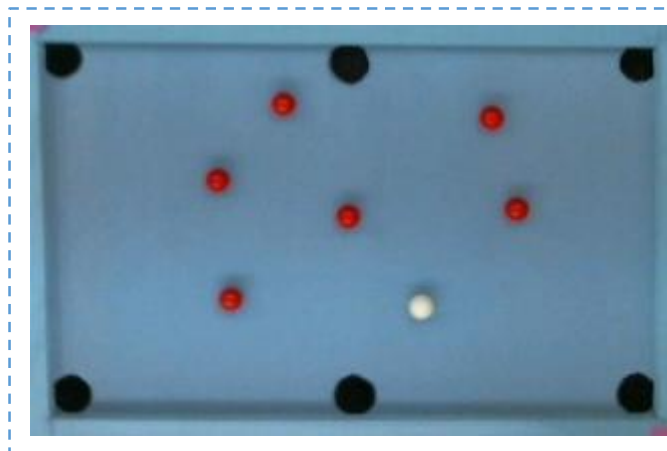
להלן התמונה הבינארית המציגה את מיקומי המדבקות בתמונה:



התוכנית מוודאת כי אכן נקלטו בדיוק שתי מדבקות בתמונה, אחרת יכולות לצוץ בעיות באופן הרצת ועבודת התוכנית.

את מיקומי המדבקות התוכנה שומרת: בהמשך יעשה בהם שימוש לצורך נירמול (Normalization) המיקום בו צריך לעמוד הרובוט.

כך תראה התמונה הגזורה:



זיהוי הכדורים והחורים:

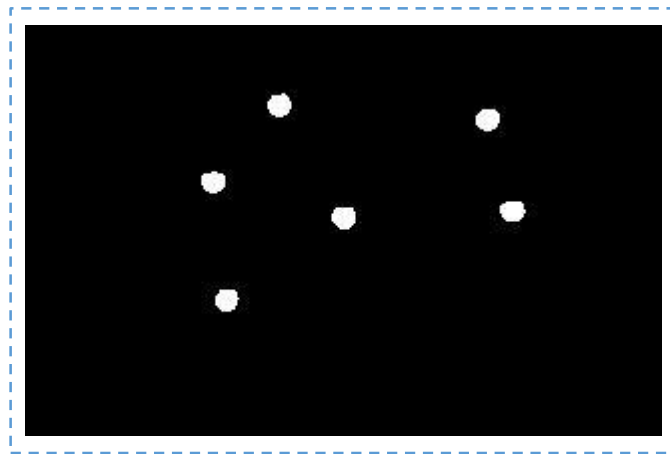
השלב הבא הוא זיהוי ושמירת מיקומם של הכדורים והחורים בשולחן, וכמו-כן גם את גודל הרדיוס של הכדורים.

לשם השגת מטרה זו, התוכנית עושה שימוש בפונקציה `findItems`. התוכנית מספקת לפונקציה את הפרמטרים הדרושים למציאת גופים מסוג מסוים, ושומרת את הגופים המוחזרים.

התוכנית חוזרת על תהליך זה עבור שלושה סוגים של גופים: כדור לבן, כדורים אדומים וחורים. התוכנית שומרת את המיקומים במבנה `Pos` (עבור הכדור הלבן היחיד), או במערך של מבנים מסוג זה (עבור גופים שמספרם גדול מאחד).

כמו-כן, התוכנית מוודאת בחלק זה כי נמצא רק כדור לבן אחד בתמונה, כי מספר החורים שנמצאו שווה לשש (מספר החורים התקני בשולחן ביליארד), ומדפיסה את מספר הכדורים האדומים על השולחן.

מיקומי הכדורים בתמונה הגזורה:



בניית אסטרטגיות המשחק:

לאחר שהתוכנית זיהתה ושמרה את מיקומי הגופים בתמונה, כעת יש למעשה לחשב את אסטרטגיות המשחק האפשריות (לפי המצב המתואר בתמונה), ולשם כך מתבצעת קריאה לפונקציה הייעודית `buildStrategiesLists`.

תפקידה של הפונקציה הנ"ל הוא לנהל את כל תהליך הבנייה של אסטרטגיות המשחק הזמינות, והיא מחזירה לתוכנית שתי רשימות (מערכים) של אסטרטגיות:

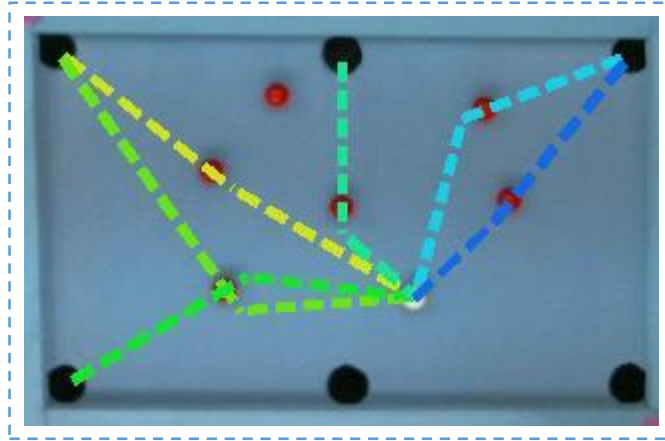
- ✓ רשימה של אסטרטגיות בסיסיות – אלו אסטרטגיות שבהן לא מעורב שרשור.
- ✓ אסטרטגיות מתקדמות – באסטרטגיות אלו מעורב שרשור (=רצף של פגיעות בין כדורים).

אופן פעולתה של פונקציה זו יוסבר בהמשך.

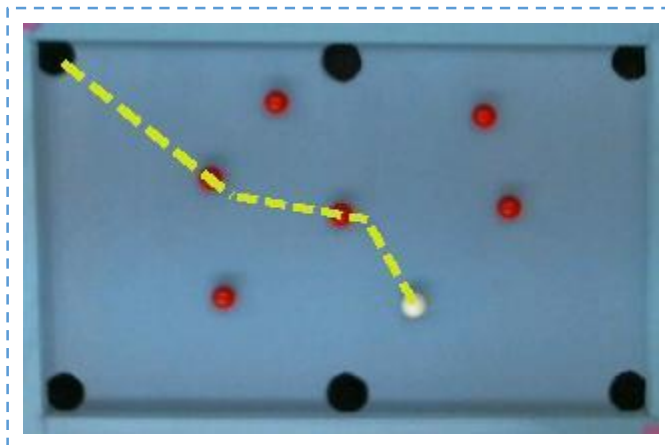
הצגת האסטרטגיות על המסך:

לאחר-מכן, התוכנית מציגה את (כל) האסטרטגיות על המסך, בעזרת שתי חלונות המציגים שני עותקים של התמונה הגזורה של השולחן. בכל חלון מוצגים וקטורי התנועה של הכדורים המעורבים באסטרטגיה, כל אסטרטגיה בצבע שונה:

❖ בחלון אחד מוצגות האסטרטגיות הבסיסיות:



❖ בחלון השני מוצגות האסטרטגיות המתקדמות:



הצגת האסטרטגיות נועדה לעזור לדבג (to debug) את הקוד ולהבטיח את תקינות אופן פעולת התוכנית.

כעת כל שנשאר זה לקחת את האסטרטגיה הטובה ביותר, לבדוק האם הרובוט יכול לבצעה, ולהורות לרובוט כיצד לעשות כן.

חישוב המיקום אליו הרובוט צריך להגיע:

כעת, לאחר שחישבנו את אסטרטגיית המשחק, יש להורות לרובוט כיצד לשחק אסטרטגיה זו – היכן עליו להגיע, כיצד עליו להגיע לשם, ובאיזו זווית עליו ליהיות.

ראשית, התוכנית מוצאת את מיקומו הנוכחי (התחלתי) של הרובוט ואת אורך הזרוע שלו (בפיקסלים). יש צורך באורך הזרוע (בפיקסלים) לשם חישוב המיקום אליו צריך הרובוט להגיע בכדי לבצע את המהלך.

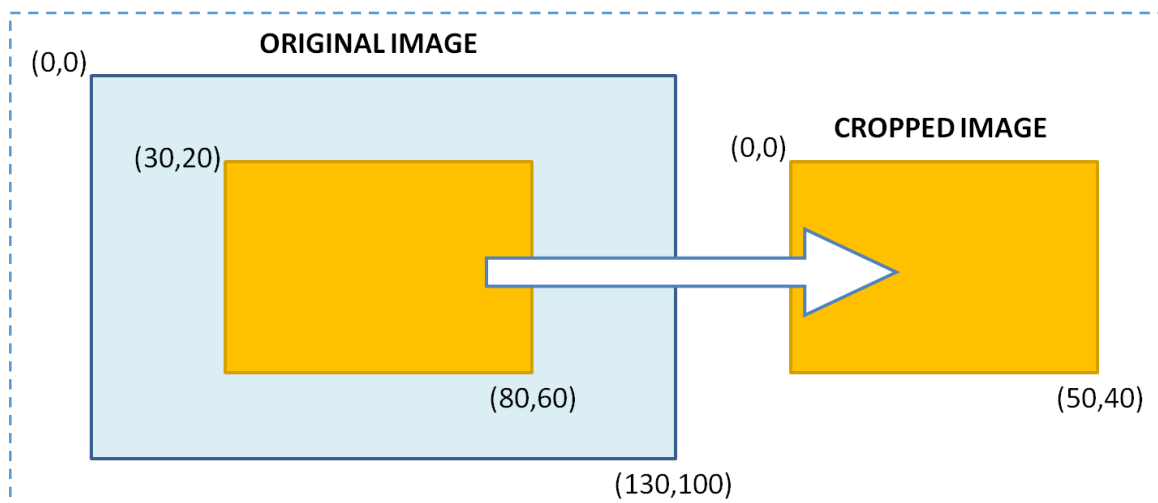
לאחר-מכן, התוכנית מחשבת את המיקום בו הרובוט צריך ליהיות בכדי לבצע את המהלך. התוכנית מוודאת כי נקודה זו אינה נמצאת בשטחו של השולחן (מאחר והרובוט אינו יכול לעמוד שם):

- ✓ במידה וכן, התוכנית תמחק את האסטרטגיה הנבדקת מרשימת האסטרטגיות (מאחר ואינה אפשרית), ותבדוק את האסטרטגיה הבאה ברשימת האסטרטגיות.
- תהליך זה מתרחש כלולאה עד שנמצאת אסטרטגיה תקינה (אפשרית).
- ✗ במידה ולא, נמצאה אסטרטגיה מתאימה, והתוכנית תמשיך בתהליך ביצועה.

כעת יש ברשותנו את הנקודה בה על הרובוט (לכאורה) לעמוד, אך ישנה בעיה: הנקודה המחושבת נמצאת ביחס למערכת הייחוס של התמונה הגזורה, ולא של הפריים המקורי.

לכן, התוכנית מבצעת נירמול של הנקודה למערכת הייחוס של הפריים המקורי, בעזרת שימוש בפונקציה המיועדת לכך (NormalizePos) ובנקודות הגזירה אותן מצאנו בתחילת התוכנית.

אופן פעולת הפונקציה עוסק בפרק המוקדש לה, והשוני בין מערכות הייחוס מתואר בתרשים הבא:



לאחר שהתוכנית התאימה (נירמלה) את הנקודה למערכת הייחוס של הפריים המקורי, יש ברשותנו את המיקום בו על הרובוט לעמוד לשם ביצוע האסטרטגיה, וכעת יש להדריך את הרובוט כיצד להגיע לנקודה זו.

מציאת הרובוט:

לשם מציאת מיקום הרובוט וזווית פנייתו ביחס למערכת הייחוס של הפריים, התוכנית עושה שימוש בפונקציה הפנימית למציאת הרובוט (getRobot).

פונקציה הנ"ל מריצה חיפוש על התמונה למציאת מדבקות צבעוניות היושבות על הרובוט, בעזרת הפונקציה החיצונית למציאת גופים בתמונה (findItems).

הדרכת הרובוט לצלע השולחן אליה הוא צריך להגיע:

לאחר שחישבנו את המיקום אליו הרובוט צריך להגיע, ומצאנו את מיקומו ההתחלתי של הרובוט, התוכנית מבצעת קריאה לפונקציה הפנימית שמטרתה להדריך את הרובוט לביצוע המהלך הנבחר (directRobot).

אך לפני ביצוע הקריאה עצמה, התוכנית מחשבת מספר משתנים הדרושים לה: ראשית, התוכנית בונה את הישר העובר דרך הכדור הלבן והמיקום אליו צריך הרובוט להגיע. בנוסף, התוכנית בונה את הווקטור העובר דרך נקודות אלו.

לשם החישובים הנ"ל התוכנית עושה שימוש בפונקציה לבניית ישר (buildLine), ובפונקציה לבניית ווקטור (buildVector).

התוכנית מבצעת קריאה לפונקציה הנ"ל, ומספקת לה את המשתנים הדרושים אותם היא זה עתה חישבה, בנוסף למיקום אליו צריך הרובוט להגיע בתמונה. הפונקציה תדריך את הרובוט בתנועתו, עד לביצוע המהלך.

כעת, התוכנית ממתינה עד לסיום הדרכת הרובוט – עד לסיום ביצוע המהלך.

לאחר ביצוע המהלך:

כעת לאחר שהמהלך בוצע ישנן שתי אפשרויות:

- ❖ ניתן להכניס לקוד התוכנית לולאת משחק (Game-Loop), כך שהתוכנית תחזור על עצמה והרובוט ימשיך לשחק עד שיגמרו המהלכים האפשריים.
- ❖ התוכנית מגיעה לסופה – היא סוגרת את החיבורים למצלמה, לשרת ולערוץ התקשורת הטורית; ולאחר מכן יוצאת.



פונקציות פנימיות

בפרק זה נסביר על פונקציות שאינן פונקציות חיצוניות.
כלומר – פונקציות אלו ייחודיות לפרוייקט שלנו, ולא ניתן להשתמש בהם בפרוייקטים אחרים.

buildStrategiesLists

מבנה הפונקציה:

```
function [norm_list, alt_list] = buildStrategiesLists(white_pos,...
    reds_pos, holes_pos)

function [list] = sub_func(list)
    last_place = size(list,2);
    list(last_place+1).plan = strategy_profile.plan;
    list(last_place+1).profit = strategy_profile.profit;
end

norm_list = [];
alt_list = [];

for i = 1:size(reds_pos)
    for j = 1:6

        empty_profile.plan(1) = holes_pos(j);
        empty_profile.profit = i*j;

        [playable, strategy_profile] = planStrategy(white_pos,
            reds_pos(i), holes_pos(j), reds_pos,...
            empty_profile, 0, 1);

        if (playable == false)
            continue;

        else
            last_profile = size(strategy_profile.plan,2);
            strategy_profile.plan(last_profile+1) = white_pos;
            if(size(strategy_profile.plan,2) == 3)
                norm_list = sub_func(norm_list);
            else
                alt_list = sub_func(alt_list);
            end
        end
    end
end
end
end
```

לקריאה עיונית:

. <http://pastebin.com/K4HCyEvS>

הסבר על הפונקציה:

מטרתה של פונקציה פנימית זו היא לבנות את שתי הרשימות המכילות את אסטרטגיות המשחק הזמינות, ולהחזירן. למעשה, פונקציה זו אחראית על בדיקתן של כל הקומבינציות האפשריות של כדור אדום וכור, סיווגן של אסטרטגיות זמינות לפי סוגן, ושמירתן לצורך שימוש מאוחר יותר.

הפונקציה מקבלת כארגומנטים את:

- מיקום הכדור הלבן.
- מערך המכיל את מיקומיהם של כל הכדורים האדומים בתמונה.
- מערך המכיל את מיקומיהם של החורים של השולחן בתמונה.

אופן פעולתה של הפונקציה הוא כדלהלן:

לפונקציה ישנה תת פונקציה אחת שאחראית על עדכונה של רשימת אסטרטגיות ברגע שנמצאת אסטרטגיה חדשה: תת הפונקציה מוסיפה את האסטרטגיה החדשה שנמצאה (שמסופקת כארגומנט) לרשימה המיועדת (שמסופקת כארגומנט).

הפונקציה עובדת במסגרת של לולאות מקוננות אשר מבצעות בדיקות עבור כל קומבינציה אפשרית של כדור וכור.

בתור המסגרת הנ"ל, מתרחשת קריאה לפונקציה הפנימית לתכנון אסטרטגיה – `planStrategy`. (הערה: פונקציה זו היא הפונקציה הכבדה ביותר בתוכנית, והרחבה עליה ניתן למצוא בפרק העוסק בה).

במידה ונמצא מהלך עבור השילוב הנוכחי של כדור וכור, הפונקציה מסווגת את המהלך לפי סוגו – מהלך בסיסי או מורכב, ובונה את מבנה (`struct`) המהלך, אשר מכיל מספר פרטים אודות מהלך זה. (הערה: הסבר אודות מבנה זה ניתן למצוא בחלק בספר "כללי עבודה בקוד").

את מבנה המהלך שנבנה הפונקציה דוחפת לרשימה המתאימה לסוג המהלך.

לבסוף, לאחר שהפונקציה סיימה לעבור על כל הקומבינציות, ושמרה את כל האסטרטגיות האפשריות ברשימות, היא מחזירה אותן ויוצאת.

planStrategy

מבנה הפונקציה:

```
function [playable, profile] = planStrategy (source_pos, target_pos,
dest_pos, reds_pos, profile, i, prev_k)

    global BALLS_RADIUS;

    global MAX_DEGREE;
    temp_max_degree = MAX_DEGREE - 20*i;

    hit_pos = calcHitPoint (target_pos, dest_pos, 2*BALLS_RADIUS);
    vec_source_hit = buildVector (source_pos, hit_pos);
    vec_target_dest = buildVector (target_pos, dest_pos);

    angle = calcVecAngle(vec_source_hit, vec_target_dest);
    if (angle > temp_max_degree)
        playable = false;
        return;
    end

    % Check source_hit
    if (temp_max_degree==MAX_DEGREE)
        [clear,k] = checkRoute (source_pos, target_pos, reds_pos,
3*BALLS_RADIUS);
        if (clear==false)
            playable = false;
            return;
        end
    end

    % Check target_dest
    [clear,k] = checkRoute (target_pos, dest_pos, reds_pos,
3*BALLS_RADIUS);
    if (clear == true)
        playable = true;
        last_step = size(profile.plan, 2);
        profile.plan(last_step+1) = hit_pos;
        return;
    end
end
```

```

else
    [playable, profile] = planStrategy(source_pos, reds_pos(k),
        dest_pos, reds_pos, profile, i+1, k);

    last_step = size(profile.plan,2);
    temp = profile.plan(last_step);

    if (temp_max_degree==MAX_DEGREE)
        hit_pos = calcHitPoint(target_pos, temp,
2*BALLS_RADIUS);

        vec_source_hit = buildVector (source_pos, hit_pos);
        vec_target_dest = buildVector (target_pos, dest_pos);
        angle = calcVecAngle(vec_source_hit, vec_target_dest);
        profile.angle = angle;

    else
        hit_pos = calcHitPoint(reds_pos(prev_k), temp,
2*BALLS_RADIUS);
    end

    profile.plan(last_step+1) = hit_pos;

end
end
end

```

לקריאה עיונית:

. <http://pastebin.com/rMhLcrZ6>

הסבר על הפונקציה:

מטרתה של פונקציה פנימית זו, היא לנסות ולבנות אסטרטגיה עבור קומבינציה מסוימת של כדור אדום וחור, ובמקרה של הצלחה, להחזיר את אובייקט המהלך מוכן לשימוש.

הקריאה לפונקציה פנימית זו מתבצעת מתוך הפונקציה הפנימית לבניית רשימות המהלכים (buildStrategiesLists).

הפונקציה היא רקורסיבית (Recursive) ולא איטרטיבית (Iterative), כלומר – הפונקציה קוראת לעצמה, ועושה שימוש בתנאי סופי כדי להפסיק את רצף הרקורסיות, במקום לעשות שימוש באיטרציות לולאה.

הסיבה שהפונקציה כתובה בצורה רקורסיבית, היא שצורת פעולתה דורשת לבצע את חישובים מסוימים על התוצאות של אותם החישובים (הסבר מעשי והגיוני יותר יבוא בהמשך).

לשם ביצוע תפקידה, הפונקציה מקבלת מספר ארגומנטים:

- **source_pos** – המיקום של כדור המקור, אשר מתנגש בכדור אחר – כדור המטרה. במקרה הבסיסי ביותר כדור זה יהיה הכדור הלבן, ובמקרה של אסטרטגיה מורכבת הכדור יכול להיות כדור אדום המעורב בשרשור.
- **Target_pos** – המיקום של כדור המטרה, אשר בו מתנגש כדור המקור, ואשר נע אל יעד מסוים.
- **Dest_pos** – היעד אליו צריך להגיע כדור המטרה. במקרה הבסיסי ביותר, או בחלק החיצוני ביותר של הרקורסיה, מיקום זה יהיה המיקום של החור אליו אנו רוצים בסופו של דבר להכניס את הכדור האדום.
- **Reds_pos** – מערך של המיקומים של כל הכדורים האדומים על השולחן. במערך זה יעשה שימוש כשהפונקציה תבדוק האם מסלול התנועה של כדור מסוים פנוי.
- **Profile** – התוכנית המתארת את המהלך נכון לעכשיו. משתנה זה נחוץ לפעילות הרקורסיה, ובקריאה חיצונית משתנה זה מחיל בתחילה רק את מיקומו של החור.
- **i** – מספר המתאר עומק הרקורסיה.
- **Prev_k** – מספרו של הכדור בהתנגשות האחרונה שחושבה, במערך המכיל את מיקומי הכדורים האדומים. משתנה זה נחוץ לרקורסיה בלבד, והוא לא יבוא לידי ביטוי באסטרטגיה בסיסית. בקריאה החיצונית, ערכו של משתנה זה אינו חשוב.

הפונקציה מחזירה שני משתנים:

- **Playable** – משתנה זה קובע כי המהלך תקין.
- **Profile** – פרופיל המשחק המעודכן ביותר נכון לחלק ברקורסיה בו הפונקציה נמצאת.

משתנים אלו דרושים גם עבור הרקורסיה, וגם עבור התוכנית עצמה.

אופן פעולת הפונקציה:

בתחילה, התוכנית מחשבת את נקודת הפגיעה של המקור במטרה, ולאחר מכן בונה את הווקטור בין המקור ונקודת הפגיעה, ואת הווקטור בין המטרה והיעד.

לאחר-מכן, הפונקציה בודקת כי הזווית בין שני ווקטורים אלו אינה גדולה מזווית מקסימאלית מוגדרת. הגודל של זווית מקסימאלית זו תלוי בעומק הרקורסיה, והוא מתחיל, בחלק החיצוני ביותר של הרקורסיה, בשבעים מעלות.

במידה והזווית בין הווקטורים גדולה מהזווית המקסימאלית, לא ניתן לבצע מהלך זה, והפונקציה מחזירה פרופיל אסטרטגיה ריק, וכי המהלך אינו בר-ביצוע.

במידה והזווית תקינה, הפונקציה עוברת לבדוק האם ישנו כדור אחר המפריע לתנועתו של כדור המקור לנקודת הפגיעה, או לתנועתו של כדור המטרה לנקודת היעד.

במידה ואין הפרעה באף אחד משני המסלולים הנ"ל, האסטרטגיה היא אסטרטגיה בסיסית, תקינה, והפונקציה מחזירה כי קומבינציה זו של כדור וחור היא בר-משחק, ואת הפרופיל של האסטרטגיה.

במידה וישנה הפרעה בין כדור המקור לנקודת הפגיעה, נקבע כי מהלך זה אינו תקין. הסיבה שתוכנית קובעת כי מהלך זה אינו תקין, היא למנוע כפילויות של אסטרטגיות מורכבות, שנבנות מקומבינציות שונות של כדור אדום וחור, אך השרשור בהם זהה.

במקרה של הפרעה במסלול בין כדור המטרה והיעד, הפונקציה נכנסת לרקורסיה פנימית יותר, לה היא מספקת מיקומים מעט שונים, כך שהבדיקות והחישובים יעשו כדי למצוא מצב של שרשור.

תהליך זה חוזר על עצמו כל עוד קיימת הפרעה בין כדור המטרה והיעד שלו, עד לעומק של ארבע חזרות.

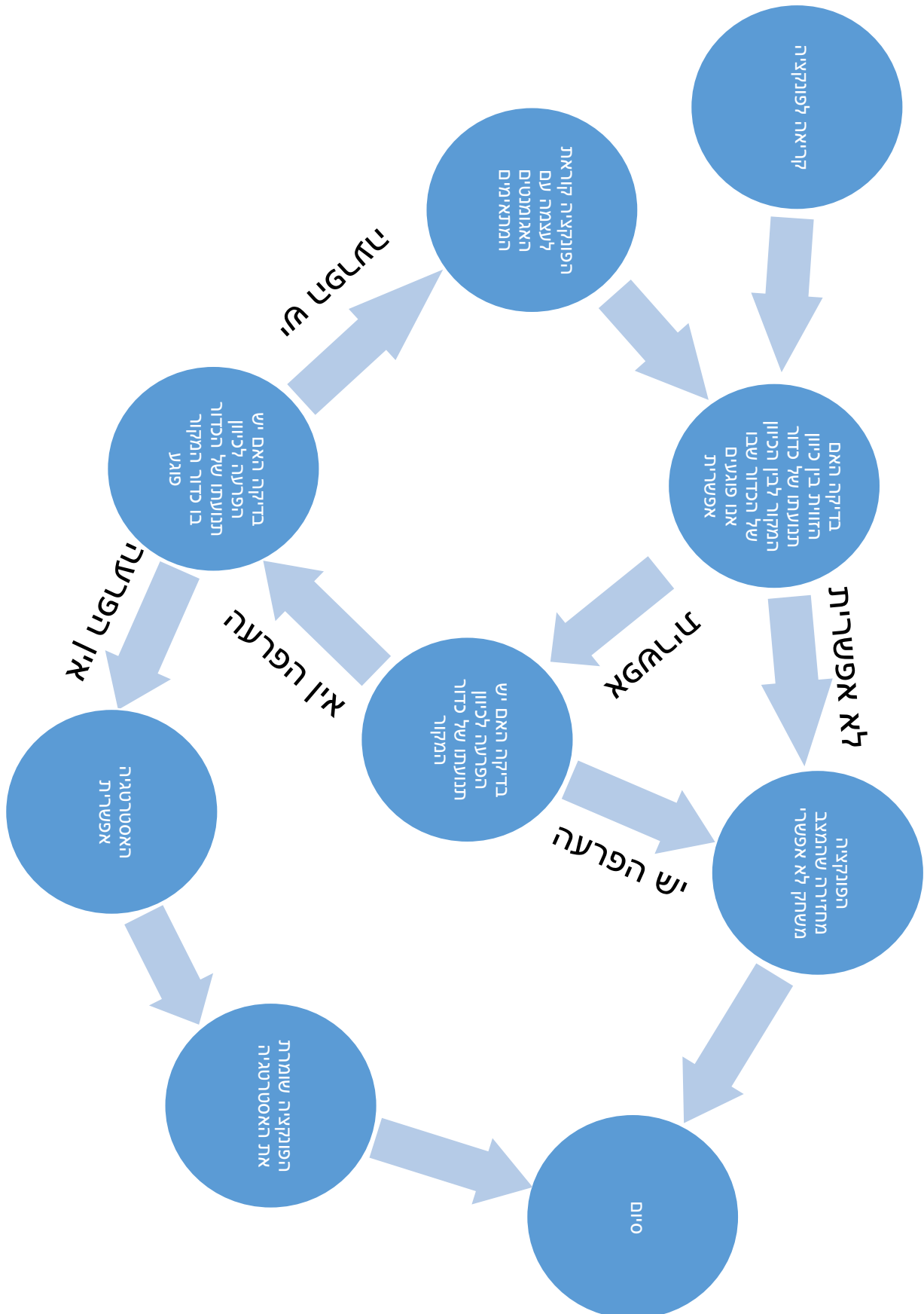
בחלק הפנימי ביותר של הרקורסיה, במידה ונמצא כי מסלול התנועה של כדור המטרה הנוכחי והיעד שלו נקי, הפונקציה קובעת כי מהלך זה הינו תקין ובר-משחק, מעדכנת את פרופיל המהלך, מחזירה כי המהלך בר-משחק (playable = true).

בגלל שהחלק הפנימי ביותר של הרקורסיה מחזיר שהמהלך תקין, גם שאר החלקים ברקורסיה יעשו כך – הם יעדכנו את פרופיל המהלך, ויחזירו כי המהלך בר-משחק.

כאשר נגיע למקור הרקורסיה (לחלק החיצוני ביותר), הפונקציה תוסיף את הכדור הלבן לפרופיל המהלך, תחזיר אותו (בנוסף לכך שהמהלך בר-משחק), ותצא.

בעמוד הבא ניתן לראות תרשים המתאר את אופן פעולתה של הפונקציה.

אופן פעולתה של הפונקציה כתרשים:



calcHitPoint

מבנה הפונקציה:

```
function [hit_pos] = calcHitPoint (target_pos, dest_pos, distance)
    t = distance/calcDistance(target_pos, dest_pos,
    'point_to_point');
    hit_pos.x = target_pos.x - t*(dest_pos.x - target_pos.x);
    hit_pos.y = target_pos.y - t*(dest_pos.y - target_pos.y);

end
```

לקריאה עיונית:

<http://pastebin.com/xbKHReMF>

הסבר על הפונקציה:

מטרת הפונקציה היא לחשב את המיקום בו גוף ראשון יפגע בגוף שני, כך שהגוף השני ינוע לכיוון נקודה מסוימת, אותה סיפקנו.

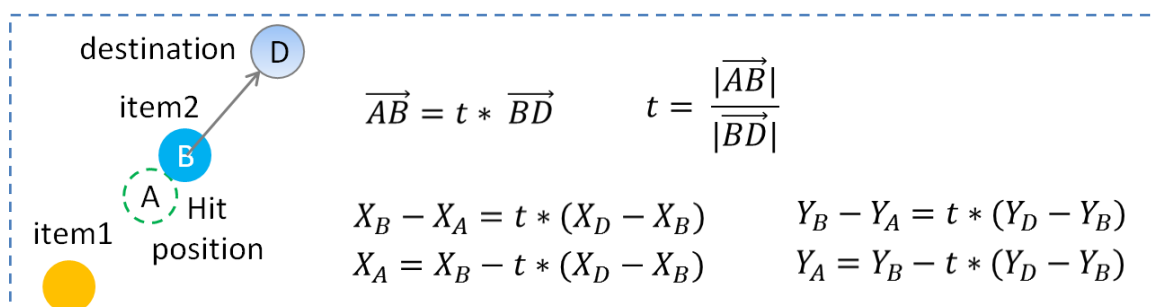
בפונקציה זו אנו עושים שימוש במספר מקרים, כאשר בחלקן המטרה אינה זהה אך דומה.

הפונקציה מקבלת שלושה ארגומנטים:

- **Target_pos** – המיקום של הגוף השני, בו יפגע הגוף הראשון.
- **Dest_pos** – הנקודה שלכיוונה ינוע הגוף השני.
- **Distance** – המרחק בין מרכזי הגופים ברגע פגיעתם.

לצורך חישוב נקודת הפגיעה הפונקציה עושה שימוש בפונקציה לחישוב מרחק (calcDistance).

התרשים הבא מתאר את אופן פעולת הפונקציה:



הפונקציה מחזירה את נקודת הפגיעה (כמבנה Pos) ויוצאת.

CheckRoute

מבנה הפונקציה:

```
function [clear, kMin] = checkRoute (source_pos, dest_pos,...
    block_pos_array, min_distance)

num_of_block = size(block_pos_array, 2);
clear = true;
dMin = Inf;
kMin = -1;

for k=1:num_of_block

    if (block_pos_array(k).x == dest_pos.x &&
        block_pos_array(k).y == dest_pos.y)
        continue;
    end

    vec_source_block = buildVector(source_pos,
        block_pos_array(k));
    vec_source_target = buildVector(source_pos, dest_pos);
    angle = calcVecAngle(vec_source_block, vec_source_target);

    vec1_abs = calcDistance(vec_source_block);
    vec2_abs = calcDistance(vec_source_target);

    if ( angle<90 && vec2_abs>vec1_abs )
        h = abs(vec1_abs * sind(angle));
        if (h < min_distance)
            d = calcDistance(block_pos_array(k), source_pos,
                'point_to_point');
            if (d < dMin)
                dMin = d;
                kMin = k;
            end
        end
    end

    if (dMin ~= Inf)
        clear = false;
    end
end
```

לקריאה עיונית:

. <http://pastebin.com/byTsRW6C>

הסבר על הפונקציה:

מטרת הפונקציה היא לוודא כי מסלול התנועה של כדור הוא פנוי, וכי אין כדור אחר המפריע לו.

במידה וכדור אחד חוסם את מסלול תנועתו של כדור אחר (לדוגמה – כדור אדום חוסם את המסלול בין הכדור הלבן לכדור אדום אחר), הפונקציה הפנימית שאחראית על בניית האסטרטגיה (buildStrategy) תקבל את המידע על חסימה זו, ותנסה למצוא אסטרטגיה אלטרנטיבית.

הפונקציה checkRoute מקבלת מספר ארגומנטים:

- **Source_pos** – מיקומו הנוכחי של הכדור שאותו רוצים להניע.
 - **Dest_pos** – המיקום שאליו רוצים להניע את הכדור.
 - **Block_pos_array** – מערך המכיל את מיקומי כל הכדורים (האדומים) על השולחן. במערך זה יכול להימצא כדור החוסם את מסלול התנועה, ולכן יש לבדוק את כל הכדורים ולוודא כי הם אינם חוסמים מסלול התנועה.
 - **Min_distance** – זהו המרחק המינימלי בו צריך להימצא כדור אחר כדי שהוא לא יחשב כחוסם את מסלול התנועה. מרחק זה תלוי ברדיוס הכדורים המחוברים.
- תחילה, הפונקציה מחשבת את מספר הכדורים האדומים על השולחן ע"י חישוב הגודל של המערך המכיל את מיקומיהם. משתנה זה נחוץ לשם הגדרת מספר האיטרציות (חזרות) של לולאת הבדיקה – חזרה אחת עבור כל אחד מן הכדורים במערך.

לאחר מכן התוכנית נכנסת ללולאת הבדיקה:

בתחילת האיטרציה (iteration = חזרה) הפונקציה בודקת האם הכדור החוסם (תיאורטית) הנבדק הוא למעשה הכדור אותו אנו מעוניינים להניע. במידה וכן הלולאה עוברת לחזרה הבאה, כלומר – מדלגת על הבדיקה עבור הכדור החוסם הנבדק הנוכחי.

לאחר מכן, הפונקציה בונה וקטורים בין הכדור המקורי לכדור החוסם, ובין הכדור המקורי והיעד.

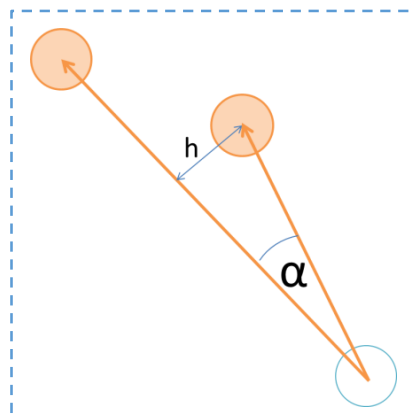
בעזרת הוקטורים הנ"ל הפונקציה מוודאת כי הכדור החוסם נמצא בתוך התחום בו נמצא מסלול התנועה.

במידה והכדור החוסם אכן נמצא בתחום הנ"ל, הפונקציה ממשיכה בבדיקה.

בהמשך הבדיקה הפונקציה מחשבת את המרחק של הכדור החוסם ממסלול התנועה (שהוא מסלול ישר), ובמידה והמרחק הזה קטן מהמרחק המינימלי שסיפקנו לפונקציה (min_distance), נקבע כי כדור זה אכן חוסם את מסלול התנועה של הכדור אותו אנו רוצים להניע.

ברגע שהפונקציה מגלה כדור חוסם, היא שומרת את מספרו במערך ויוצאת מן הלולאה.

לבסוף, הפונקציה מחזירה האם המסלול נקי, ובמידה והוא אינו נקי היא מחזירה גם את מספר הכדור החוסם, אותו היא שמרה קודם לכן.



normalizePos

מבנה הפונקציה:

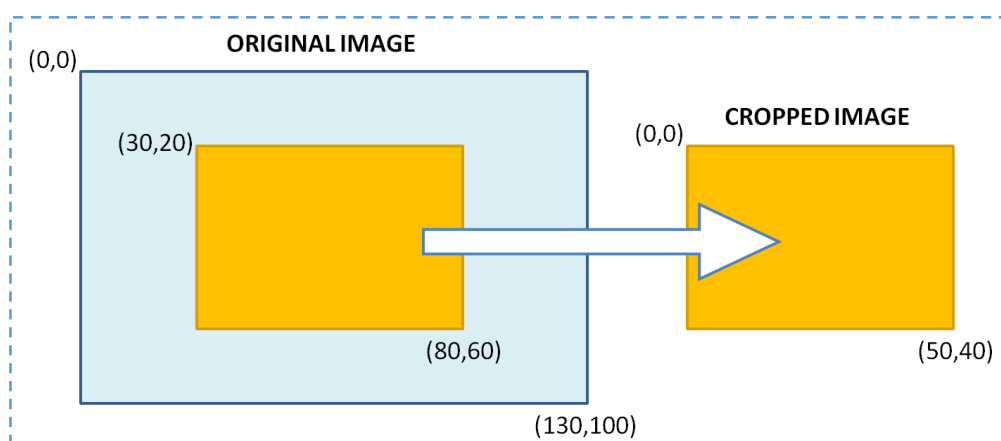
```
function [fixed_pos] = normalizePos(original_pos, index)
    fixed_pos.x = original_pos.x + index.x;
    fixed_pos.y = original_pos.y + index.y;
end
```

לקריאה עיונית:

<http://pastebin.com/hYUAKfk4>

הסבר על הפונקציה:

תפקיד הפונקציה הוא לנרמל קורדינטות תמונה הגזורה, לקורדינטות בתמונה המקורית. כלומר – בגלל שהתמונה הגזורה אינה זהה לתמונה המקורית, מערכות הייחוס של כל אחת מהן הן שונות, כפי שניתן לראות בתרשים הבא:



מאחר ואנו עושים את החישובים על התמונה הגזורה (כדי שהחישובים יהיו מדויקים יותר), ולאחר מכן עלינו להשתמש בתמונה המקורית כדי לשלוט על הרובוט, יש להמיר את נקודת היעד של הרובוט למערכת הייחוס של התמונה המקורית.

הפונקציה מקבלת שני ארגומנטים:

- **original_pos** - קורדינטה על התמונה הגזורה.
- **index** - ראשית הצירים של התמונה הגזורה, במערכת הייחוס של התמונה המקורית.

הפונקציה מחזירה את הקורדינטה שעברה נירמול (כמבנה Pos) ויוצאת.

getRobot

מבנה הפונקציה:

```
function [robot_pos, stick_pos, arm_length] = getRobot(h,s,v)

    function [object_pos] = func1(server, h,s,v, color, radius)
        object_pos = findItems (server, h,s,v, color, radius);
        if (size(object_pos, 2) ~= 1)
            error('ERROR: The robot was not detected
properly. ');
        end
    end

    global server;

    robot_pos = func1(h,s,v, 'yellow', 2);
    stick_pos = func1(h,s,v, 'green', 2);

    arm_length = calcDistance(robot_pos, stick_pos,
'point_to_point');

end
```

לקריאה עיונית:

. <http://pastebin.com/Qthp6Uvs>

הסבר על הפונקציה:

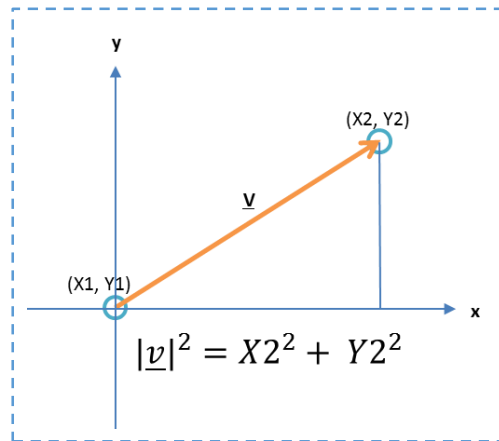
תפקידה של פונקציה זו הוא לאסוף מידע על הרובוט ברגע קריאתה. אנו עושים שימוש בפונקציה זו בזמן אמת לשם מעקב אחרי הרובוט בזמן תנועתו והכוונתו למיקום בו עליו לעמוד כדי לבצע את המהלך.

הפונקציה מקבלת כארגומנט את ערכי ה-HSV של התמונה, ומחזירה מידע אודות מיקום הרובוט בפריים, את מיקום קצה הזרוע המכה, ואת אורך הזרוע.

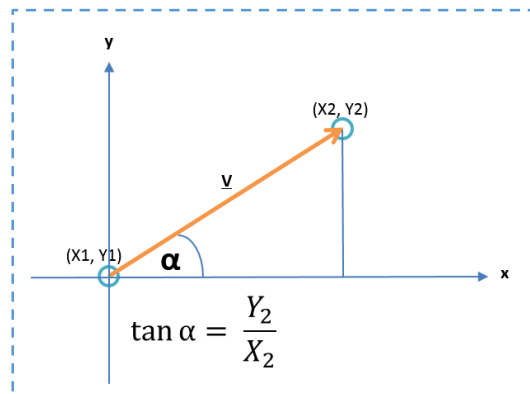
לשם מציאת מיקום הרובוט בפריים, הפונקציה עושה שימוש בשתי מדבקות צבעוניות המודבקות במרכז הרובוט ובקצה הזרוע של הרובוט. הפונקציה משתמשת בפונקציה החיצונית למציאת גופים בתמונה (findItems) לשם מציאת מיקומי המדבקות.

את מיקום הרובוט עצמו, ואת מיקום קצה הזרוע אנו יודעים לפי מיקומי המדבקות, אך לשם מציאת הזווית של הרובוט ביחס למערכת הייחוס של התמונה, עלינו לבצע חישובים נוספים.

אנו בונים וקטור בין מיקומי המדבקות בעזרת הפונקציה החיצונית לבניית וקטור (buildVector).



לאחר מכן, אנו עושים שימוש בפונקציה החיצונית לחישוב זווית של וקטור (calcVecAngle) על הווקטור שחישבנו, והזווית שזו מחזירה היא הזווית של הרובוט ביחס למערכת הייחוס של התמונה.



לבסוף, הפונקציה מחזירה את המידע שהיא אספה, ויוצאת.

directRobot

מבנה הפונקציה:

```
function directRobot(target_line, target_pos, vec_line)

    function [h,s,v] = getHSVfromCapture()
        trigger(source_capture);
        frame = getdata(source_capture);
        [h,s,v] = getHSV(frame);
    end

    function [object_pos] = getObject(h,s,v, color, radius)
        object_pos = findItems (server, h,s,v, color, radius);
        if (size(object_pos, 2) ~= 1)
            error('ERROR: The robot was not detected properly.');
        end
    end

    function stopAndWait()
        controlRobot(COM1,'stop',1);
        pause(0.5);
    end

    global source_capture;
    global server;
    global COM1;

    TILL_DISTANCE = 10;
    dist_robot_line = Inf;
    controlRobot(COM1,'forward',1);
    while(dist_robot_line > TILL_DISTANCE)
        [h,s,v] = getHSVfromCapture();
        rob_pos = getObject(h,s,v, 'yellow', 2.55);

        dist_robot_line = calcDistance(rob_pos, target_line,
'point_to_line');
    end

    stopAndWait();
```

```

TILL_DEGREE = 10;
alpha = Inf;
controlRobot(COM1,'rotate_right',1);
while(alpha > TILL_DEGREE)
    [h,s,v] = getHSVfromCapture();
    robot_pos = getObject(h,s,v, 'yellow', 2.55);
    stick_pos = getObject(h,s,v, 'green', 2.55);

    vec_robot_stick = buildVector(robot_pos, stick_pos);
    alpha = calcVecAngle(vec_robot_stick, vec_line);
end

stopAndWait();

TILL_DISTANCE = 10;
dist_robot_target = Inf;
controlRobot(COM1,'forward',1);
while(dist_robot_target > TILL_DISTANCE)
    [h,s,v] = getHSVfromCapture();
    robot_pos = getObject(h,s,v, 'yellow', 2.55);
    dist_robot_target = calcDistance(robot_pos, target_pos,
    'point_to_point');
end

stopAndWait();
disp('Robot in position.');
```

end

לקריאה עיונית:

. <http://pastebin.com/P6ik4acR>

הסבר על הפונקציה:

מטרתה של פונקציה זו היא לשלוט על הרובוט ולהדריך אותו לשם ביצוע המהלך הנבחר.

פונקציה זו מקבלת שלושה ארגומנטים, הדרושים לחישובים המעורבים בהדרכת הרובוט:

- **target_line** – הישר העובר דרך הכדור הלבן והמיקום אליו צריך להגיע.
- **target_pos** – המיקום של הרובוט בפריים.
- **Vec_line** – הווקטור בין המיקום שאליו הרובוט צריך להגיע והמיקום של הכדור הלבן.

הדרכת הרובוט לצלע השולחן:

הדרכת הרובוט מתחילה בהוראה לרובוט לעבוד במצב של "טייס-אוטומטי", כך שהוא יסתובב מסביב לשולחן בצורה עצמאית, עד שיגיע לצלע השולחן שלידה נמצא המיקום אליו צריך הרובוט להגיע.

הסיבה שאנו עושים שימוש במצב זה, היא שמספר העיבודים בשנייה שהתוכנית שלנו מסוגלת לעשות אינו גדול מספיק כדי לעקוב אחרי תנועה מהירה של הרובוט מסביב לשולחן, מאחר והשפה איננה מותאמת ליישומי עיבוד תמונה בזמן אמת (Real-time Image Processing).

לשם העברת הרובוט למצב של "טייס-אוטומטי" התוכנית שולחת לרובוט מילה בינארית של שמונה סיביות (8-Bit), אשר מכילה שלושה משתנים:

- ❖ את המצב של טייס אוטומטי, מתוך ארבעה מצבים אפשריים, בצורה של שתי הסיביות החשובות ביותר (Most Significant Bits).
- ❖ את המהירות בה הרובוט צריך לנסוע, מתוך ארבע מהירויות אפשריות, בצורה של שתי סיביות (סיביות חמש ושש).
- ❖ את מספר הפאות שעל הרובוט לעבור (0 עד 4). לשם משתנה זה מוגדרות ארבע סיביות, אך תפקידן משתנה בהתאם למצב בו על הרובוט להימצא, וכך גם מספר הפקודות בכל מצב.

אך לפני שנורה לרובוט כמה פאות עליו לעבור, יש לחשב משתנה זה. לכן, הפונקציה מוצאת את הפאות שלידן נמצאים הרובוט והמיקום אליו הוא צריך להגיע. בעזרת נתונים אלו התוכנית מחשבת את מספר הפאות שעל הרובוט לעבור.

לאחר מכאן התוכנית מעבירה לרובוט את הפקודה המורה לו להסתובב מסביב לשולחן, ולעבור את מספר הפאות אותו חישבנו, וממתינה לאישורו של הרובוט כי סיים לבצע הוראה זו.

הצורה בה הרובוט מסתובב מסביב לשולחן, מוסברת בספר הפרויקט של הזוג השני איתו עבדנו על הפרויקט, ואשר היה אחראי על הבנייה של הרובוט.

הדרכת הרובוט למיקום המדויק:

לאחר שהרובוט הגיע לצלע השולחן, יש לכוונו למיקום המדויק בו הוא צריך להימצא, ולזווית המדויקת בה על הרובוט לעמוד כדי לבצע את המהלך.

בחלק זה של התוכנית הרובוט נמצא במצב "עבד", והתוכנית היא ששולטת על כל פעולותיו, ומבקרת אותו. בחלק זה הרובוט רק מבצע הוראות בסיסיות כמו "סע", "עצור", "הסתובב", והוא אינו עושה חישובים בעצמו.

התוכנית שולטת ברובוט בזמן אמת, ומדריכה אותו להגיע במהירות בינונית עד למרחק מסוים מישר התנועה אותו חישבנו ועליו הרובוט צריך להימצא.

ואז, התוכנית מדריכה את הרובוט לנסוע במהירות האיטית ביותר עד שהרובוט יעמוד על הישר.

כאשר הרובוט נמצא על הישר, התוכנית מורה לו להסתובב לאט עד שזוויתו תהיה שווה לזווית וקטור הכיוון. כלומר – הרובוט מסתובב עד שהוא פונה לכיוון וקטור הכיוון אשר חשבנו בתחילת חלק זה של התוכנית.

כאשר הרובוט פונה בזווית הנכונה, התוכנית מורה לו להתקדם שוב באיטיות, כך שהוא יעמוד בדיוק במיקום בו עליו להיות כדי לבצע את המהלך.

ביצוע המהלך:

כעת, כשהרובוט עומד במיקום ובזווית המתאימים לביצוע האסטרטגיה הנבחרת, התוכנית מורה לרובוט לבצע את המהלך (לתת מכה לכדור). כאשר הרובוט מקבל הוראה זו, הוא מפעיל את המנוע הליניארי שמורכב על זרוע הרובוט, נותן מכה לכדור, ומסיים את ביצוע המהלך.

כאן מסתיים החלק של הדרכת הרובוט לביצוע האסטרטגיה, והפונקציה יוצאת.

controlRobot

מבנה הפונקציה:

```
function controlRobot(command, speed)

global COM1;

state{1} = {'stop'};
state{2} = {0,1,2,3};
state{3} = {'forward', 'backward', 'right', 'left'};
state{4} = {'rotate_right', 'rotate_left', 'stick_up', 'stick_down',
            'stick_inc_angle', 'stick_dec_angle', 'hit'};

switch(command)
    case state{1}
        word = 0;
        index = 1;

    case state{2}
        word = 63;
        index = 2;

    case state{3}
        word = 128;
        index = 3;

    case state{4}
        word = 192;
        index = 4;

    otherwise
        error(sprintf('ERROR: %s is not a legal command.', command));
end

[~,temp] = ismember(command,state{index});
word = word + temp + 16 * speed;

fwrite(COM1, word, 'uint8');

end
```

לקריאה עיונית:

<http://pastebin.com/MPkVXG9G>

הסבר על הפונקציה:

מטרתה של הפונקציה הנ"ל היא לשלוח הוראות לרובוט דרך ערוץ התקשורת הטורית (RS-232).
לפונקציה זו שימוש הכרחי בהדרכת הרובוט לביצוע המהלך הנבחר.

הפונקציה מקבלת שני ארגומנטים:

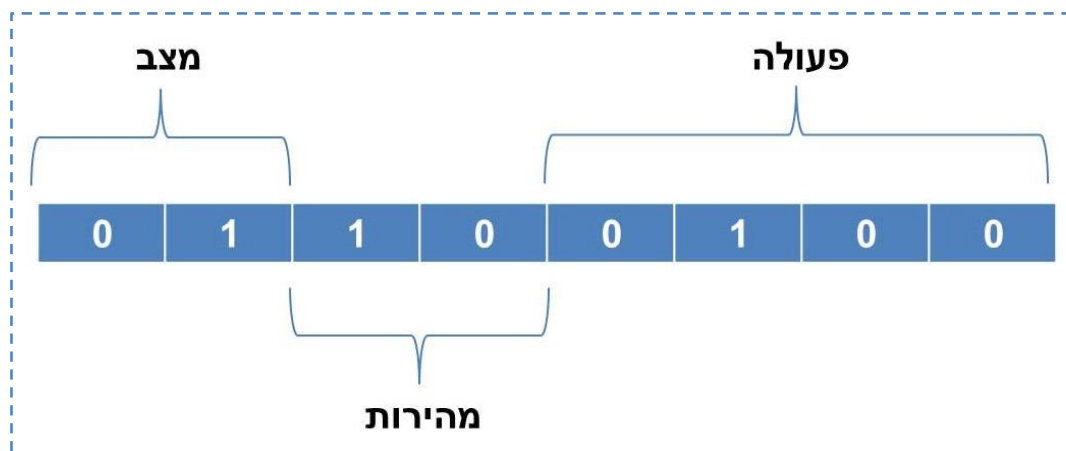
- ❖ **Command** – מחרוזת (string) המכילה את הפקודה שעל הרובוט לבצע (לדוגמה – סע ישר, הסתובב, עצור...), מתוך רשימה מוגדרת של פקודות.
- ❖ **Speed** – המהירות שבה יש לבצע את הפקודה, כאשר ישנן 4 מהירויות אפשריות.

במסגרת העבודה עם הזוג שעבד על הרובוט, החלטנו כי לרובוט יהיו ארבעה מצבים אפשריים:

- **עצירה מוחלטת** – במצב זה הרובוט עוצר לחלוטין במקומו ואינו זז כלל.
- **טייס אוטומטי** – במצב זה הרובוט פועל בצורה עצמאית לצורך תנועתו מסביב לשולחן (הצורך במצב זה מוסבר בחלק "מבנה התוכנית").
- **שליטה כיוונית** – שליטה על כיוון נסיעתו של הרובוט: קדימה, אחורה, שמאלה, ימינה.
- **הכנות אחרונות** – במצב זה נעשה שימוש לשם ההכנות האחרונות לפני ביצוע המהלך: סיבוב הרובוט, הורדת המנוע (הליניארי) המכה, נתינת מכה, ועוד.

את ההוראה שנשלחת לרובוט, המוגדרת כמילה בת שמונה סיביות (8-Bit), תכננו כך שהיא תכיל גם את המצב הרצוי לפעול הרובוט, גם את המהירות לביצוע הפקודה, וגם את הפקודה הרצויה בתוך המצב הרצוי.

מבנה המילה הנשלחת מתואר בתרשים הבא:



אופן פעולתה של הפונקציה:

בתחילת הפונקציה מוכרזות הפקודות הזמינות בכל אחד מן המצבים, בצורה של משתנים ייחודיים לשפה (Matlab) הקרויים "תאים" (Cells). השימוש בסוג משתנה זה נועד להבטיח כי ניתן יהיה לבצע חיפוש קל של פקודות במצבים הקיימים.

לאחר מכן מגיע תנאי רב ערכי (Switch-Case), הבודק עבור הפקודה הרצויה באיזה מצב היא נמצאת.

ואז, באמצעות משחק עם הסיביות, אנו מרכיבים את המילה הנשלחת לרובוט, אשר מכילה את המצב, המהירות והפקודה.

מאחר ואנו מכירים את הבסיס הבינארי בצורה טובה מאוד, ומאחר ושנינו מנוסים בהנדסת תוכנה, המשחק עם הסיביות היה קל ביותר ומהיר.

לבסוף, הפונקציה שולחת את המילה לרובוט, ויוצאת.

waitForRobotResponse

הקוד:

```
function waitForRobotResponse()
    global COM1;
    disp('Waiting for robot response...');

    response = 0;
    while(response == 0)
        response = fread(COM1);
    end

    disp('The robot have reported.');
```

לקריאה עיונית:

<http://pastebin.com/BJSJhtZh>

הסבר על הפונקציה:

תפקידה של פונקציה זו הוא להמתין עד לקבלת אישור מהרובוט כי הוא סיים לנוע מסביב לשולחן, וכי הוא עומד כעת ליד הצלע הנכונה של השולחן – הצלע שלידה נמצא המיקום בו על הרובוט לעמוד בכדי לבצע את המהלך הנבחר.

הפונקציה אינה מקבלת ארגומנטים, ואינה מחזירה משתנים כלל.

מבנה הפונקציה הוא בסיסי ביותר:

בתוך הפונקציה ישנה לולאה אשר חוזרת על עצמה כל עוד לא התקבלה תשדורת מהרובוט. בתוך הלולאה מתבצעת בדיקה של ערוץ התקשורת הטורית במחשב, והבדיקה חוזרת על עצמה כל עוד הערך המוחזר שווה לאפס.

כך הפונקציה ממתינה עד לקבלת התשדורת מהרובוט (ערך כלשהו שונה מאפס), וכשזו מתקבלת הפונקציה יוצאת.



פונקציות חיצוניות

כאן נרחיב על כל פונקציה חיצונית: שאיננה נמצאת בקובץ הפונקציה הראשית.
פונקציות חיצוניות ניתנות כתובות בצורה כללית, ולכן ניתן להשתמש בהן גם בתכונות אחרות.

setupCamera

מבנה הפונקציה:

```
function [source_capture] = setupCamera(CAMERA_ID)
    disp('Checking if the camera is ready:');

    try
        cam = imaghwinfo('winvideo',CAMERA_ID);
        text = sprintf('%s detected! trying to connect...',
            cam.DeviceName);
        disp(text);

        try
            source_capture = videoinput('winvideo', CAMERA_ID,
                'RGB24_1920x1080');

            catch
                % Cleaning previuos connection...
                temp = imagfind;
                stop(temp(size(temp,2)));

                % Retrying to connect...
                source_capture = videoinput('winvideo', CAMERA_ID,
                    'RGB24_1920x1080');

            end

            disp('Connection established successfully!');

            set(source_capture, 'FramesPerTrigger', 1);
            set(source_capture, 'TriggerRepeat', Inf);
            triggerconfig(source_capture, 'manual');

            catch
                error('ERROR: The camera is being used by another
                    program...');

            end

            catch
                source_capture = {};
                error('ERROR: The camera is not connected.');
```

end

לקריאה עיונית:

. <http://pastebin.com/udmaPDe7>

הסבר על הפונקציה:

מטרת הפונקציה היא להתחבר ולהגדיר מצלמה המחוברת למחשב, כך שניתן יהיה להשתמש בה במהלך התוכנית. הפונקציה מקבלת כארגומנט יחיד את מספר הזיהוי של המצלמה אליה רוצים להתחבר.

בתחילה, הפונקציה בודקת האם המצלמה המדוברת אכן מחוברת למחשב:

- ❖ במידה ולא, הפונקציה מדפיסה הודעת שגיאה בקונסול, ויוצאת.
- ❖ במידה וכן, הפונקציה מדפיסה בקונסול את שמה של המצלמה לצורך אימות.

במידה והמצלמה אכן מחוברת, כעת נבדק האם היא נמצאת בשימוש. במידה והיא פנויה החיבור נוצר, המצלמה מוגדרת, והפונקציה מחזירה את אובייקט המצלמה מוכן לשימוש.

במידה והמצלמה אינה פנויה, הפונקציה בודקת האם הגורם לכך הוא חיבור קודם של Matlab למצלמה, או שהמצלמה נמצאת בשימוש ע"י תוכנה חיצונית:

- ❖ במידה והמצלמה אינה פנויה בגלל חיבור קודם של Matlab, זה יסגור את החיבור הקודם, ואז יבוצעו חיבור והגדרה של המצלמה.
- ❖ במידה ונעשה שימוש במצלמה ע"י תוכנה חיצונית, תודפס שגיאה מתאימה על המסך והפונקציה תצא.

כעת נסביר מהן הגדרות המצלמה:

- ❖ המצלמה מוגדרת להחזיר פריימים ברזולוציה של FullHD (1920 על 1080 פיקסלים).
- ❖ עבור כל טריגר הנשלח למצלמה, זו תחזיר פריימ אחד בלבד.
- ❖ המצלמה ממתינה בצורה תמידית לקבלת טריגר.
- ❖ צורת פעולתו של הטריגר מוגדרת כברירת מחדל (Default).

הצורך בהגדרות אלו הוא לשם הכנתה של המצלמה להחזרה מהירה יותר של פריימים עבור קריאה (טריגר) מתקבל. הגדרות אלו מאפשרות פיתוח של מימוש העושה שימוש בעיבוד תמונה בזמן אמת (Real Time Application).

במידה והחיבור וההגדרה עברו בהצלחה הפונקציה תצא ותחזיר אובייקט (object) מצלמה מוכן לשימוש.

isMoving

מבנה התוכנית:

```
function [new_image] = isMoving(source_capture)
    FILTER = 50;
    MAX_CHANGE_PERCENT = 15;
    STATIC_SEQ_COUNT = 5;

    text = sprintf('\nWaiting for a static image sequence...');
    disp(text);

    trigger(source_capture);
    old_image = getdata(source_capture);
    pix_in_image = size(old_image,1) * size(old_image,2);

    while(count < STATIC_SEQ_COUNT)
        trigger(source_capture);
        new_image = getdata(source_capture);

        diff_image = (new_image - old_image) + (old_image -
new_image);

        R = diff_image(:,:,1);
        G = diff_image(:,:,1);
        B = diff_image(:,:,1);

        binImage = (R > FILTER) | (G > FILTER) | (B > FILTER);

        change_percent = sum(sum(bin_image)) / pix_in_image;
        if(change_percent < MAX_CHANGE_PERCENT);
            count = count + 1;
        else
            count = 0;
        end

        old_image = new_image;
    end

    disp('The image sequence is now static.');
```

end

לקריאה עיונית:

. <http://pastebin.com/49B9Gk4Y>

הסבר על הפונקציה:

הפונקציה הראשונה שרצה בכל חזרה של לולאת המשחק היא הפונקציה `isMoving`.

מטרתה של פונקציה זו היא לוודא שהסביבה הנקלטת מהמצלמה היא סטטית, כלומר – אין תנועה של גופים בין פריימים עד לגבול מסוים קבוע מראש.

התוכנית שלנו עושה שימוש בפונקציה זו כדי לוודא שהכדורים הפסיקו לזוז לאחר ביצוע המהלך האחרון ששיחק הרובוט. חשוב לוודא כי הגופים אכן הפסיקו לנוע, כי תנועתם תשפיע על דיוק החישובים ועל נכונות אסטרטגיות המחשבות.

בכדי לוודא כי הסביבה המצולמת אינה בתנועה יש להשוות בין שתי תמונות עוקבות, ולבדוק את אחוז השוני בניהן, כלומר – את היחס בין הפיקסלים השונים (בין שתי התמונות) למספר הפיקסלים בתמונות המקור.

תחילה, הפונקציה מצלמת פריים ראשון מהמצלמה, ומיד מחשבת את מספר הפיקסלים בפריים (תמונה) זה – במשתנה זה יעשה שימוש בהמשך הפונקציה.

לאחר-מכן מגיעה לולאת `while` אשר רצה כל עוד לא היה רצף של חמש פריימים זהים.

מבנה הלולאה הוא כזה:

הפונקציה קולטת פריים שני מהמצלמה, ולאחר מכן בונה תמונת שוני של שני הפריימים, כלומר – תמונה בה ערכי ה-`RGB`, יהוו את ההפרש בערכים אלו בין שתי התמונות.

הבעיה היא שמאחר וערכי ה-`RGB` הם משתנים מסוג `unit8` וחיסור של שני משתנים מסוג זה תמיד יחזיר תוצאה גדולה או שווה לאפס, ולא את הפרש האמיתי בין שני המשתנים. לכן, חשבנו על הנוסחא הבאה:

```
diff_image = (img1 - img2) + (img2 - img1);
```

הנוסחא הנ"ל מחזירה תמונה בה ערכי ה-`RGB` מהווים את ההפרש האבסולוטי בין ערכי ה-`RGB` של שתי התמונות המקוריות, ובכך פתרנו בעיה זו.

השלב השני הוא סינון הפיקסלים בתמונת השוני לפי רמת השוני (גודל ערכי ה-`RGB` של כל פיקסל). הצורך בשלב זה נובע מכך, שעל-אף ששתי תמונות נראות זהות, קיימים הבדלים מזעריים בין שתי התמונות (שמקורם בשינויי תאורה ובגורמים נוספים), ולכן יש לבדוק גם את מידת השוני של כל הפיקסלים בתמונת השוני.

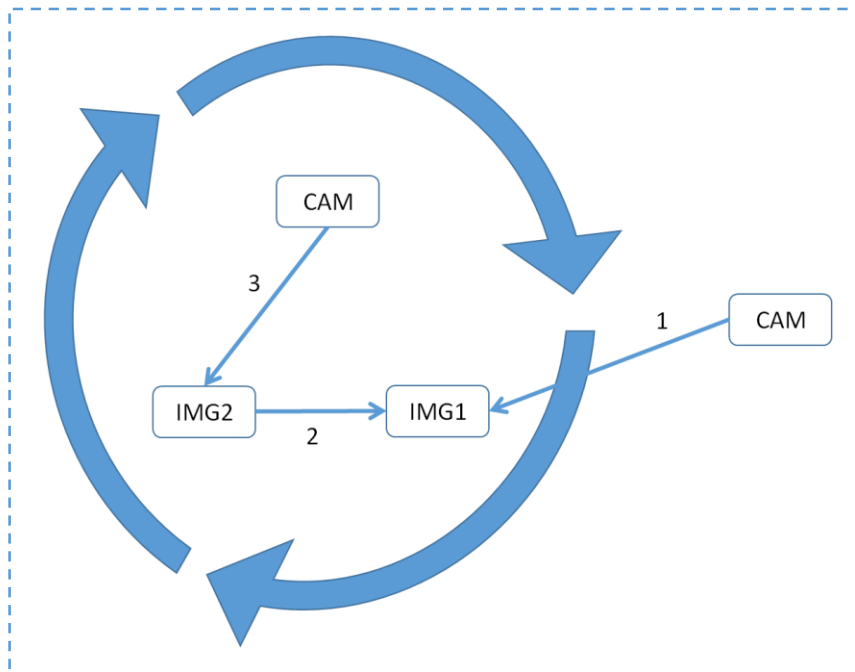
לשם כך ממירים את התמונה לתמונה בינארית ע"י סינון ערכי ה-`RGB` של כל אחד מהפיקסלים בתמונה, כך שבתמונה הבינארית רק הפיקסלים שעברו את סף השוני הקבוע מראש יקבלו ערך 1 (`true`).

בשלב הבא סופרים את מספר הפיקסלים שערכם 1, ומחשבים את היחס בין מספר זה למספר הפיקסלים בתמונה המקורית, וכך מתקבל אחוז השינוי בתמונה. במידה ואחוז זה קטן מאחוז שינוי קבוע מראש, המשתנה `count` (המונה את מספר הפריימים הסטטי ברצף הנוכחי) גדל באחד.

במידה והמשתנה שווה חמש, כלומר – היה רצף של חמש פריימים זהים, מתרחשת יציאה מהלולאה, והפונקציה יוצאת ומחזירה את הפריים האחרון שנקלט.

בכדי לחסוך בזמן ההרצה של כל חזרה של הלולאה, ולהגדיל את מספר החזרות בכל שנייה, אנו מצלמים בכל חזרה רק תמונה אחת נוספת אותה שומרים כתמונה השנייה, ואילו את התמונה השנייה של החזרה הקודמת אנו שומרים כתמונה הראשונה של התמונה הנוכחית.

לשם המחשה, להלן גרף המתאר את התהליך:



getHSV

מבנה הפונקציה:

```
function [h,s,v] = getHSV(rgb_image)
    hsv_image = rgb2hsv(rgb_image);
    h = hsv_image(:,:,1);
    s = hsv_image(:,:,2);
    v = hsv_image(:,:,3);
end
```

לקריאה עיונית:

. <http://pastebin.com/9iNmrwtj>

הסבר על הפונקציה:

מטרת הפונקציה היא החזרת ערכי ה-HSV של תמונה, אותה מקבלת הפונקציה כארגומנט. הסיבה לקיומה של פונקציה זו היא שאת זיהוי הצבעים התוכנה שלנו עושה בעזרת שיטת ה-HSV, ולא שיטת ה-RGB.

ראשית הפונקציה מבצעת המרה של התמונה משיטת RGB לשיטת HSV. לאחר מכן הפונקציה מפרקת את התמונה לערכי HSV, אותם היא מחזירה ולאחר מכן יוצאת.

עוד על הנושא בפרק "עיבוד תמונה".

connectToDatabase

מבנה הפונקציה:

```
function [conn] = connectToDatabase()  
    dbname = 'sql28217';  
    username = 'sql28217';  
    password = 'OUR_PASSWORD';  
    driver = 'com.mysql.jdbc.Driver';  
    dburl = ['jdbc:mysql://sql2.freemysqlhosting.net:3306/'  
    dbname];  
  
    javaclasspath('C:\Program Files\MySQL\Connector J 5.1.24\mysql-  
connector-java-5.1.24-bin.jar');  
  
    conn = database(dbname, username, password, driver, dburl);  
  
end
```

לקריאה עיונית:

<http://pastebin.com/7u252UVN>

הסבר על הפונקציה:

תפקיד הפונקציה הוא להתחבר לשרת מסד הנתונים שלנו (שרת MySQL), בו מאוחסנות דגימות הצבעים.

הפונקציה לא מקבלת ארגומנטים כלל, כלומר – היא אינה מקבלת פרטי חיבור לשרתים, והיא יכולה להתחבר רק לשרת שלנו. בשפה המקצועית פונקציה כגון זו נקראת Hard-Coded.

להלן תמונה הלקוחה מתוך ממשק השליטה של שרת ה-MYSQL:

Object Browser

SCHEMAS

Search objects

▼

sql28217

▼

Tables

▶

black

▶

green

▶

magenta

▶

red

▶

white

▶

yellow

▶

Views

▶

Routines

magenta

Filter:

Edit

File:

Autosize:

	n	h_min	s_min	v_min	h_max	s_max	v_max
▶	1	0.872222	0.453125	0.443137	0.955399	0.616667	0.54902
	2	0.872222	0.496815	0.607843	0.955399	0.653179	0.705882
	3	0.872222	0.453125	0.443137	0.955399	0.660819	0.705882
	4	0.872222	0.50303	0.6	0.955399	0.653179	0.698039
	5	0.872222	0.5125	0.611765	0.955399	0.627219	0.690196
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

הפונקציה מחזירה את אובייקט (Object) החיבור לשרת מסד הנתונים.

detectColor

מבנה הפונקציה:

```
function [binImage] = detectColor(conn, h,s,v, color)

    sql = sprintf('select h_min, min(s_min), min(v_min), h_max,
max(s_max), max(v_max) from %s',...
    color);

    data = cell2mat(fetch(conn, sql));
    hsv_min = data(1:3);
    hsv_max = data(4:6);

    if (strcmp(color, 'red'))
        binImage = (h < hsv_max(1) | h > hsv_min(1)) & (s >
hsv_min(2)) & ...
            (s < hsv_max(2)) & (v > hsv_min(3)) & (v < hsv_max(3));

    elseif (strcmp(color, 'white'))
        binImage = (s < hsv_max(2)) & (v > hsv_min(3));

    elseif (strcmp(color, 'black'))
        binImage = (v < hsv_max(3));

    else % Any other color.
        binImage = (h > hsv_min(1)) & (h < hsv_max(1)) & (s >
hsv_min(2)) & ...
            (v > hsv_min(3));
    end
end
```

לקריאה עיונית:

. <http://pastebin.com/UBS3y0Br>

הסבר על הפונקציה:

מטרתה של פונקציה זו היא למצוא את נוכחותו של צבע מסוים בתמונה. אנו עושים שימוש בפונקציה זו בתוכניתנו בכדי לזהות גופים שונים בתמונה. בדרך-כלל היא נקראת מתוך הפונקציה `findItems`.

הפונקציה מקבלת מספר ארגומנטים, והם:

- **Conn** – אובייקט החיבור למסד הנתונים בו מאוחסנות דגימות הצבעים. יש צורך בדגימות אלו בכדי לדעת את ערכי ה-HSV המתאימים לצבע המבוקש.
- **h,s,v** – ערכי ה-HSV של התמונה שאותה הפונקציה תבחן.
- **Color** – מחרוזת המכילה את שם הצבע אותו יש לחפש בתמונה.

אופן פעולתה של הפונקציה הוא כזה:

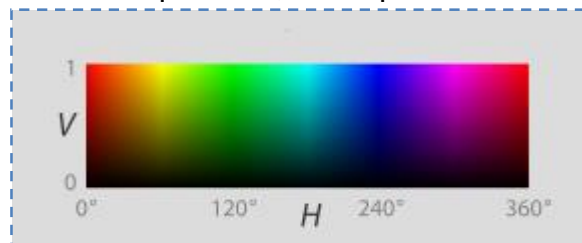
הפונקציה נגשת למסד הנתונים ומחזירה מהטבלה של הצבע הנבחר, את ערכי ה-HSV המגדירים את צבע זה. ערכי ה-HSV נבחרים כך שהטווח להגדרתם יהיה הגדול ביותר.

ההוראה להשגת ערכי HSV אלו ממסד הנתונים היא בשפת SQL:

```
select h_min, min(s_min), min(v_min), h_max, max(s_max), max(v_max) from  
COLOR_NAME;
```

לאחר שהפונקציה יודעת מהם ערכי ה-HSV המגדירים את הצבע הנבחר, היא עושה סינון של הפיקסלים בתמונה לפי ערכים אלו, כך שמתקבלת תמונה בינארית המציגה את נוכחותו של צבע זה בתמונה המקורית.

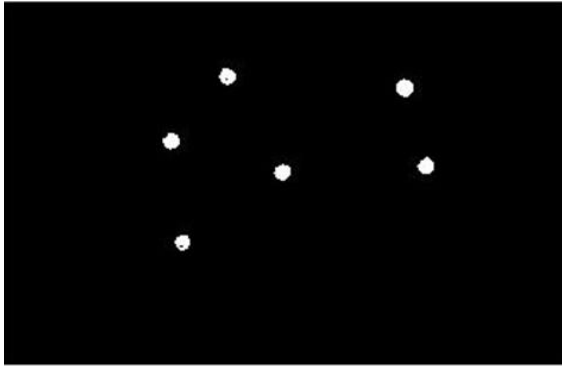
יש לשים לב שעבור הצבעים אדום, שחור ולבן יש כללים מיוחדים לסינון התמונה:
❖ הצבע האדום נמצא בשני טווחי גוון מנוגדים, כפי שניתן לראות בתרשים הבא:



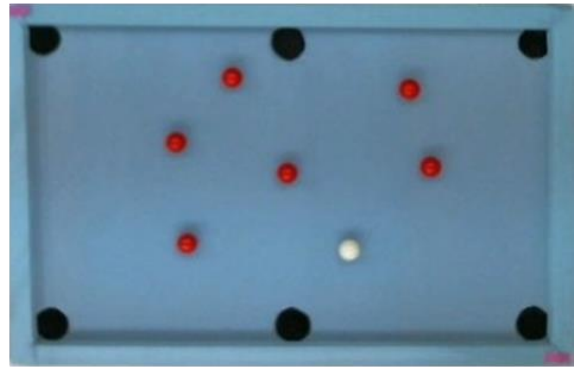
מסיבה זו יש להתייחס באופן שונה לערכי המינימום והמקסימום של טווח ה-H המגדיר את הצבע: ערכי הגוון (H) של הפיקסלים בתמונה צריכים להיות קטנים משלושים (בערך) וגדולים משלוש-מאות ושלושים (בערך).

- ❖ עבור הצבע השחור יש להתייחס אך ורק לערכי ה-V של הפיקסלים, זאת מאחר ולצבע השחור אין עוצמה (S) או גוון (H), אלא רק רמת בהירות (נמוכה).
- ❖ עבור הצבע הלבן יש להתעלם מהגוון של הפיקסלים בתמונה המקורית, זאת מאחר והצבע הלבן הוא חסר גוון.
- ❖ עבור שאר הצבעים יש להתייחס לשלושת ערכי ה-HSV.

להלן המחשה לתוצר הפונקציה, המציג את נוכחות הצבע האדום בתמונה:



התמונה הבינארית המציגה את נוכחות
הצבע האדום בתמונה



התמונה המקורית

לאחר סינון הפיקסלים בתמונה המקורית, הפונקציה מחזירה את התמונה הבינארית המתקבלת, אשר מציגה את נוכחות הצבע הנבחר בתמונה.

findItems

מבנה הפונקציה:

```
function [items_pos, Radius] = findItems (conn, h,s,v, color, radius)
    temp = pi * radius^2;
    min_area_of_objects = round(temp);
    binImage = detectColor(conn, h,s,v, color);

    binImage = bwareaopen(binImage, min_area_of_objects);
    temp = regionprops(binImage, 'centroid');

    for i=1:size(temp, 1)
        items_pos(i).x = temp(i).Centroid(1);
        items_pos(i).y = temp(i).Centroid(2);
    end

    [~,temp] = max(max(binImage,[],2));
    Radius = items_pos(1).y - temp;
end
```

לקריאה עיונית:

. <http://pastebin.com/3pmyJ822>

הסבר על הפונקציה:

מטרת הפונקציה היא למצוא גופים בעלי גודל וצבע מסוים, בתמונה.

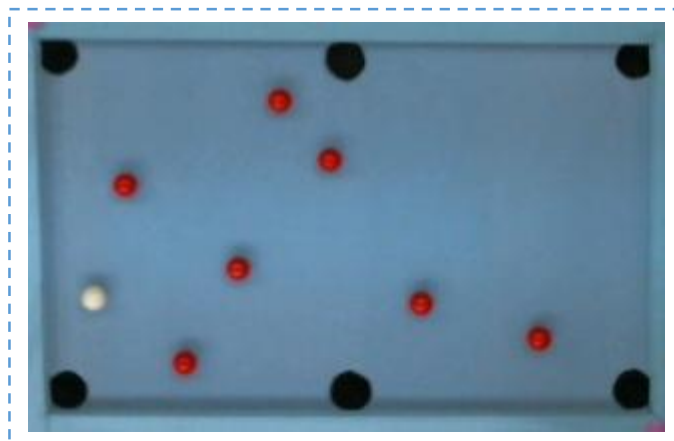
הפונקציה מקבלת מספר ארגומנטים:

- אובייקט חיבור לשרת האחסון של דגימות הצבעים (עוד על נושא זה בפרק "דגימות צבע").
 - ערכי HSV של התמונה בה אנו מחפשים את הגוף.
 - צבעו של הגוף כמחרוזת (מילה).
 - רדיוס מינימאלי של גודל הגוף.
- בתחילה, הפונקציה מחשבת (בעזרת הרדיוס) את השטח המינימאלי של הגוף, בוא יעשה שימוש בהמשך הפונקציה. השטח מחושב ע"י הנוסחה המתמטית הפשוטה למציאת שטח מעגל:

$$A = \pi r^2$$

לאחר מכן, מתרחשת קריאה לפונקציה `detectColor`, המחזירה תמונה בינארית המציגה את נוכחות הצבע שניתן בתמונה עליה נערך החיפוש.

לדוגמה, נבצע חיפוש של הכדור הלבן בתמונה הבאה.



להלן התמונה הבינארית המוחזרת, המציגה את נוכחות הצבע הלבן בתמונה.



ניתן להבחין בבירור בכדור הלבן, אך ניתן גם להבחין בהשתקפויות האור הלבן (שמקורו בנורות פלורוסנט) בכדורים האדומים.

לכן, יש לסנן את הגופים הנראים בתמונה לפי גודלם, כך שתתקבל תמונה בינארית אשר תציג אך ורק את הגופים אותם חיפשנו.

הסינון נעשה ע"י הפונקציה `bwareaopen`, המסננת את הגופים לפי שטחם (מספר הפיקסלים) – רק גופים ששטחם גדול מהשטח המינימאלי אותו חישבנו קודם, יופיעו בתמונה המסוננת.

להלן אותה תמונה ממקודם, לאחר שהעברנו עליה סינון בעזרת הפונקציה המדוברת:

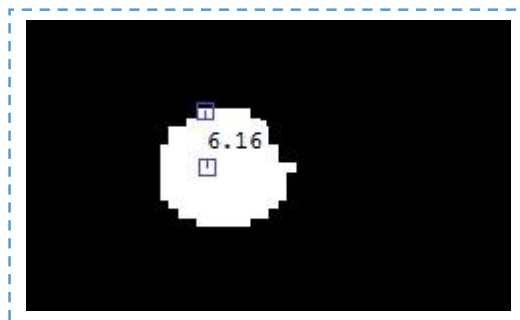


כעת מופיע בתמונה רק הכדור הלבן אותו חיפשנו.

ואז, הפונקציה מריצה על התמונה את האלגוריתם `*Centroid`, המוצא את מרכזי הגופים המופיעים בתמונה המסוננת.

*עוד על נושא זה בפרק "עיבוד תמונה".

כעת, הפונקציה מחשבת את הרדיוס האמיתי של הגוף, ע"י חיסור של ערכי ה-Y של הפיקסל הגבוה ביותר של הגוף הראשון (הגוף השמאלי ביותר) ושל מרכזו.



לבסוף, הפונקציה מחזירה את מרכזי הגופים כמערך של מבני `Pos`, ואת רדיוסם (המשותף).

buildLine

מבנה הפונקציה:

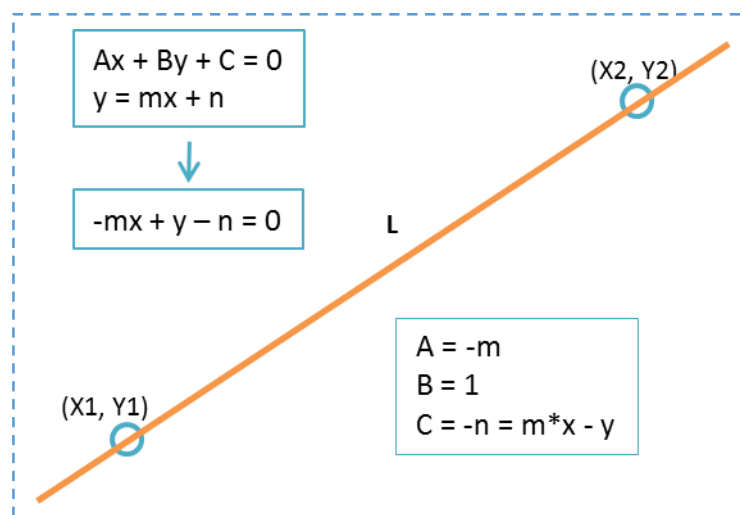
```
function [line] = buildLine(pointA, pointB)
    m = (pointB.y-pointA.y) / (pointB.x-pointA.x);
    line(1) = -m;
    line(2) = 1;
    line(3) = (m*pointA.x - pointA.y);
end
```

לקריאה עיונית:

<http://pastebin.com/wvLcSArr>

הסבר על הפונקציה:

תפקיד הפונקציה buildLine הוא בניית משוואת ישר במישור, המיוצגת ע"י הפרמטרים של הישר. הפונקציה מקבלת כארגומנטים (Arguments) שתי נקודות במישור, השמורות כמבני Pos, ומחזירה את הפרמטרים של הישר במערך. לשם מציאת הפרמטרים של הישר השתמשנו בנוסחה למציאת משוואת ישר במרחב, אותה פישטנו בדרך הבאה:



הסיבה שבחרנו לשמור את הפרמטרים של המישור במערך, היא הקלה במבנה וביעילות התוכנית בהמשך.

לבסוף, הפונקציה מחזירה את המערך המכיל את הפרמטרים של הישר ויוצאת.

buildVector

מבנה הפונקציה:

```
function [vec] = buildVector (start_point, end_pos)
    vec(1) = end_pos.x - start_point.x;
    vec(2) = end_pos.y - start_point.y;
end
```

לקריאה עיונית:

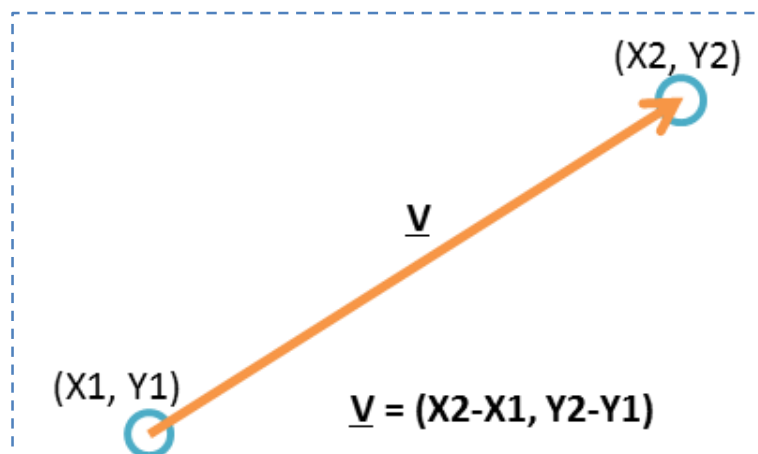
<http://pastebin.com/Q7pdPHQ0>

הסבר על הפונקציה:

תפקידה של הפונקציה buildVector הוא לבנות וקטור בעזרת שתי נקודות.

הפונקציה מקבלת כארגומנטים שתי נקודות במישור, אשר שמורות כמבני Pos. לשם בניית הוקטור הפונקציה מחסרת את ערכי ה-X וערכי ה-Y של הנקודות.

לאחר החישובים, הפונקציה מחזירה את הוקטור בין שתי נקודות אלו, ויוצאת.



calcVecAngle

מבנה הפונקציה:

```
function [angle] = calcVecAngle (vec1, vec2)
    if (nargin==1)
        angle=atand(vec1(2)/vec1(1));

    elseif (nargin==2)
        cosA = (dot(vec1,vec2)) / ((calcDistance(vec1)) *
        (calcDistance(vec2)));
        angle = acosd(cosA);

    else
        error('ERROR: too many input arguments in function
        findVecAngle');

    end
end
```

לקריאה עיונית:

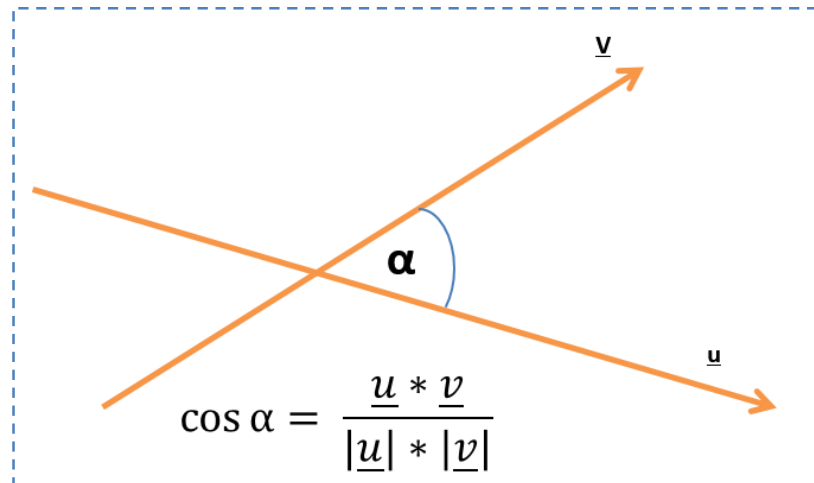
. <http://pastebin.com/AnzPMDSw>

הסבר על הפונקציה:

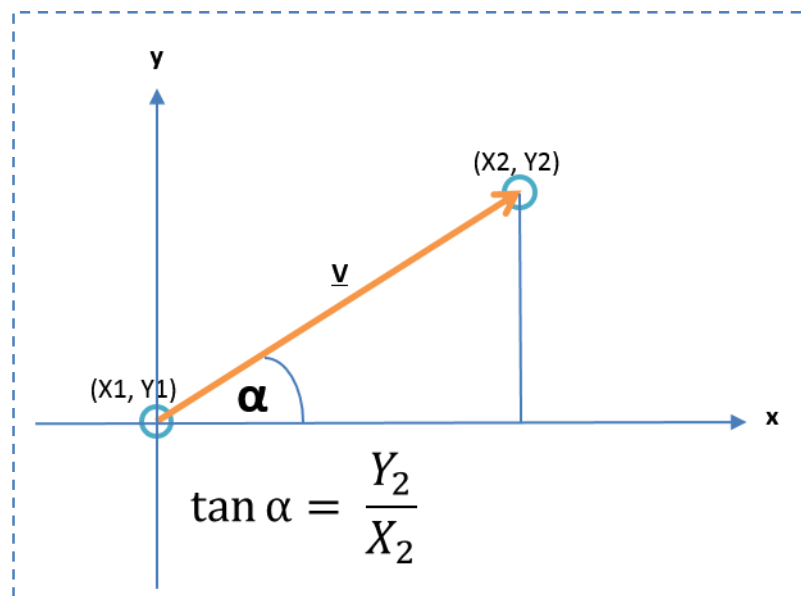
מטרת הפונקציה היא חישוב הזווית בין שני וקטורים במישור, או חישוב הזווית של וקטור ביחס למישור.

הפונקציה יכולה לקבל ארגומנט אחד או שני ארגומנטים, בהתאם למצב הרצוי:

- Vec1 – הווקטור הראשון (חובה).
 - Vec2 – הווקטור השני (תלוי במצב הרצוי).
- במידה והפונקציה מקבלת שני ארגומנטים (שניהם ווקטורים בלבד), היא מחשבת את הזווית בניהם בעזרת הנוסחה המתמטית המוכרת למציאת זווית בין שני ווקטורים:



במידה והפונקציה מקבלת רק ווקטור אחד, היא מחשבת את הזווית שלו ביחס למישור (ביחס לציר ה-X). לשם מציאת הזווית של הווקטור אנו עושים שימוש בערכי ה-X וה-Y של הווקטור. השיטה מתוארת בתרשים הבא:



לבסוף, בסיום החישובים הדרושים, הפונקציה מחזירה את הזווית (במעלות) ויוצאת.

calcDistance:

מבנה הפונקציה:

```
function [distance] = calcDistance (item1, item2, mode)
    % Calculate the absolute value of a vector.
    if (nargin==1 || strcmp(mode, 'vector_abs'))
        distance = sqrt(item1(1)^2 + item1(2)^2);

    % Calculate the distance of two points from each other.
    elseif (nargin==2 || strcmp(mode, 'point_to_point'))
        distance = sqrt((item1.x-item2.x)^2 + (item1.y-item2.y)^2);

    % Calculate the shortest distance of a point from a line.
    elseif (nargin==3 && strcmp(mode, 'point_to_line'))
        a=item2(1); b=item2(2); c=item2(3); clear('item2');
        distance = abs(a*item1.x + b*item1.y + c) / sqrt(a^2 + b^2);
    else
        error('ERROR: not an allowed mode of calcDistance.');
```

end

לקריאה עיונית:

. <http://pastebin.com/VYNEzyu4>

הסבר על הפונקציה:

מטרת הפונקציה היא חישוב המרחק בין שני גופים במישור.

בתוכניתנו אנו עושים שימוש בפונקציה זו מספר פעמים, בכל פעם שיש צורך במציאת מרחקים.

הפונקציה יכולה לקבל מספר משתנה של פרמטרים – בין אחד לשלושה פרמטרים:

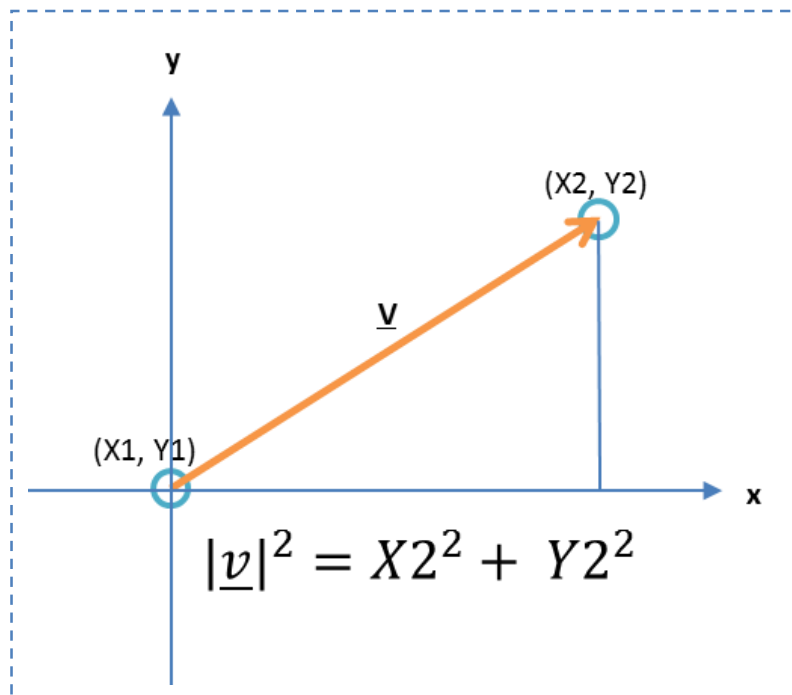
- Item1 - הגוף הראשון (חובה).
- Item2 - הגוף השני (רשות עבור מצב מסוים).
- Mode - המצב הרצוי כמחרוזת (רשות, אך מומלץ לצורך הבהרת אופן השימוש).

גוף (item) יכול להיות אחד מהבאים:

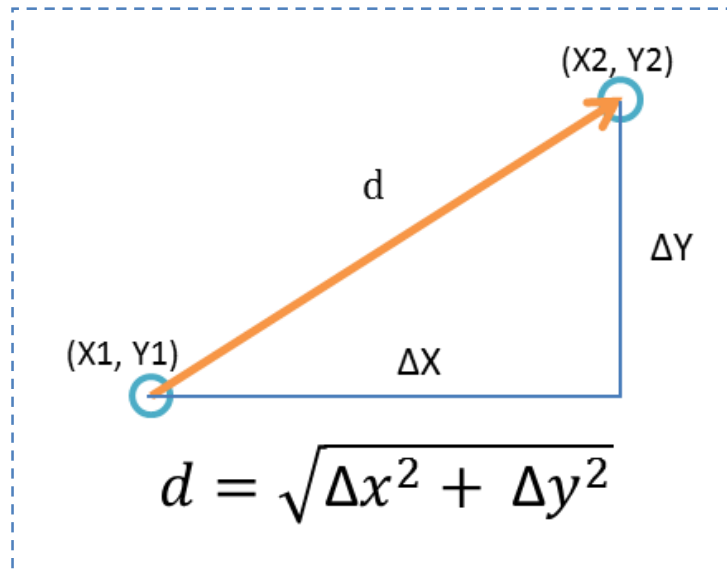
- וקטור במישור.
- נקודה במישור (כמבנה Pos).
- ישר במישור (שמור כמערך המכיל את הפרמטרים של הישר).

לפונקציה שלושה מצבים אפשריים:

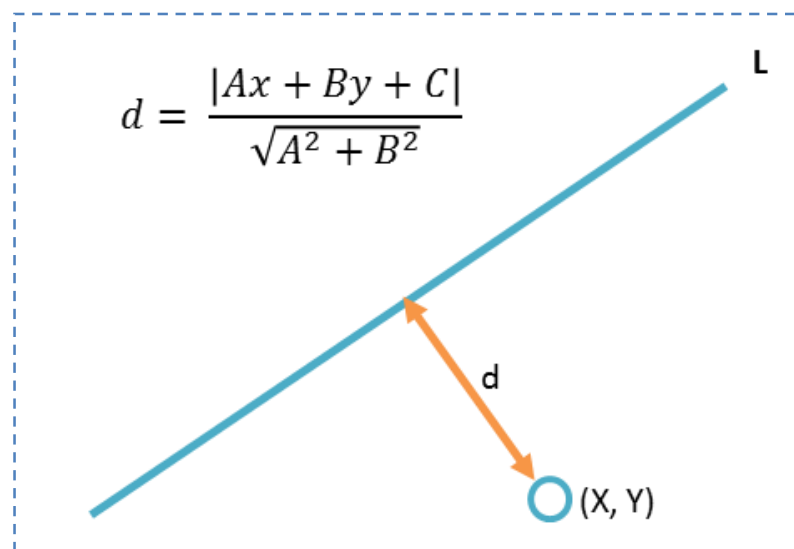
- Vector abs – מציאת הערך המוחלט (Absolute) של וקטור. במצב זה הגוף הראשון (item1) יהיה מסוג וקטור. לשם חישוב הערך המוחלט של הווקטור, הפונקציה עושה שימוש בנוסחה המוכרת "משפט פיתגורס":



- Point to point – מציאת המרחק בין שתי נקודות במרחב. במצב זה שני הגופים (items) יהיו מבני Pos, המייצגים את מיקומם במישור. לשם חישוב המרחק בין שתי הנקודות, הפונקציה עושה שימוש בנוסחה המוכרת למציאת מרחק בין שתי נקודות (נוסחת distance).



- Point to line – מציאת המרחק בין נקודה וישר במישור. במצב זה הגוף הראשון (item1) יהיה מבנה Pos, המייצג את מיקומה של הנקודה במישור, והגוף השני יהיה מערך Line המכיל את הפרמטרים של הישר. לשם מציאת המרחק בין הנקודה והישר הפונקציה עושה שימוש בפונקציה המוכרת למציאת מרחק של נקודה מישר.



לאחר סיום החישובים הפונקציה מחזירה את התוצאה המהווה את המרחק בין הגופים.

במידה והפונקציה מקבלת מצב לא מוכר, או שיש חוסר התאמה בין מספר (או סוג) הגופים המתקבלים והמצב הנבחר, הפונקציה מדפיסה שגיאה בקונסול ויוצאת.

openSerialBus

מבנה הפונקציה:

```
function [COM1] = openSerialBus()
    COM1 = instrfind('Type', 'serial', 'Port', 'COM1', 'Tag', '');

    if isempty(COM1)
        COM1 = serial('COM1');
    else
        fclose(COM1);
        COM1 = COM1(1);
    end

    fopen(COM1);
    set(COM1, 'BaudRate', 115200);

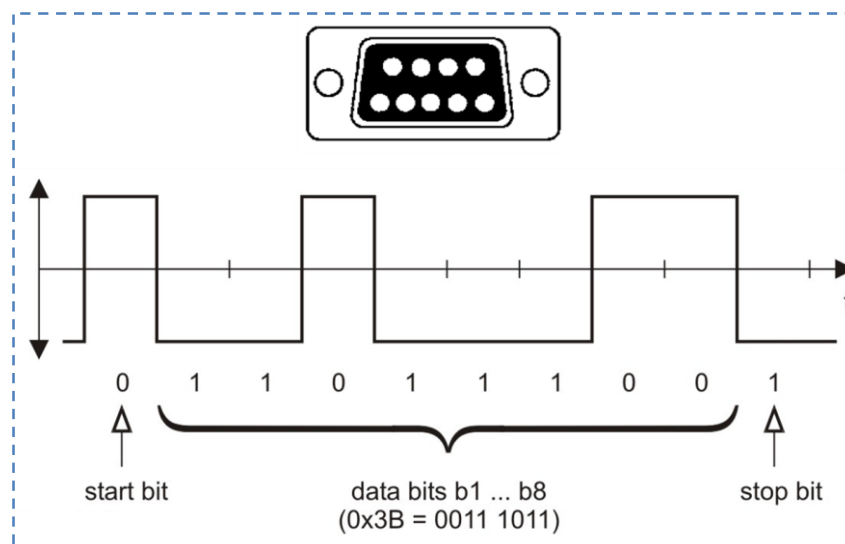
end
```

לקריאה עיונית:

<http://pastebin.com/eANj6DY9>

הסבר על הפונקציה:

תפקיד הפונקציה הוא לפתוח ולהגדיר את יציאת RS-232 במחשב, כך שתתאפשר תקשורת טורית (סריאלית) עם הרובוט (עוד על נושא זה בפרק "תקשורת"). החיבור מוגדר להעביר מילים (תשדורות) של 8 סיביות (8-Bit) בקצב (Baud Rate) של 115200 Baud, מפורט (Port) COM1.



הפונקציה מחזירה את אובייקט החיבור מוכן לשימוש, ויוצאת.

findSide

מבנה הפונקציה:

```
function [side, side_id] = findSide(check_point, upper_left_limit,
    buttom_right_limit)

    if (check_point.x < upper_left_limit.x)
        side = 'left';
        side_id = 1;

    elseif (check_point.x > buttom_right_limit.x)
        side = 'right';
        side_id = 3;

    elseif (check_point.y < upper_left_limit.y)
        side = 'up';
        side_id = 4;

    elseif (check_point.y > buttom_right_limit.y)
        side = 'buttom';
        side_id = 2;

    else
        side = 'in_frame';
        side_id = -1;

    end

end
```

לקריאה עיונית:

<http://pastebin.com/uiZdkupQ>

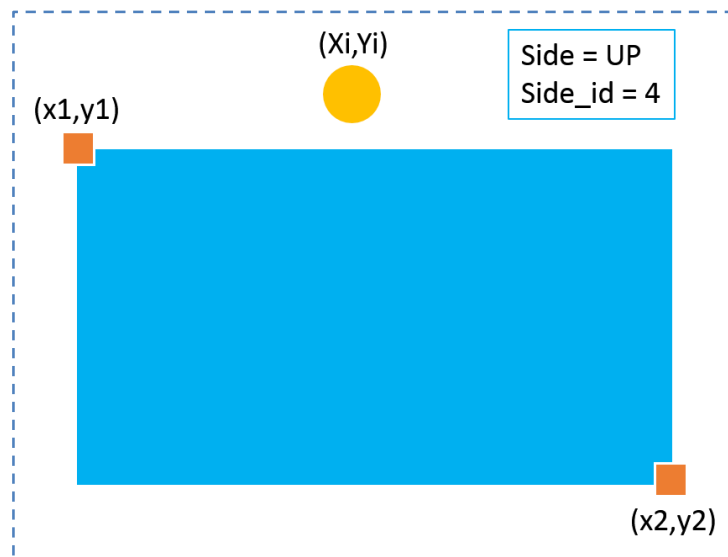
הסבר על הפונקציה:

מטרת הפונקציה היא למצוא ליד איזו צלע של השולחן נמצא גוף.

בכדי לעשות כן הפונקציה מקבלת שלושה ארגומנטים:

- מיקומו של הגוף (כמבנה Pos).
 - מיקומן של שתי פינות קיצוניות של השולחן (כמבני Pos).
- את מיקום הפינות אנו מחשבים למעשה בעת גזירת השולחן.
בכדי למצוא את הצלע שלידה נמצא השולחן, הפונקציה משווה את מיקומו של הגוף למיקומן של פינות שולחן, באמצעות מערכת תנאים פשוטה.

דוגמא לתרחיש המופנה לפונקציה:



לאחר שנמצאה הצלע, הפונקציה מחזירה את מספרה (ישנה מוסכמה לגבי המספור של הצלעות) ואת שמה (כמחרוזת).



תוכניות עזר

כאן נרחיב על תוכניות שכתבנו שבהן הפונקציה הראשית אינה משתמשת אך הן נחוצות.

cleanCameraConnection

מבנה הפונקציה:

```
function cleanCameraConnection()  
    temp = imaqfind();  
  
    for i=1:size(temp,2)  
        stop(temp(i));  
    end  
end
```

לקריאה עיונית:

<http://pastebin.com/rUedGMm8>

הסבר על הקוד:

מטרת הפונקציה הנ"ל היא לסגור את החיבורים הקודמים למצלמה.

הצורך בפונקציה:

כאשר התוכנית שלנו מבצעת חיבור למצלמה, היא למעשה מונעת מתוכנות אחרות לעשות שימוש בה. לכן, בסוף התוכנית אנו סוגרים את החיבור למצלמה, כך שהיא תהיה זמינה לתוכנות אחרות במידת הצורך, או כאשר נריץ שוב את התוכנית.

לעיתים קורה שהתוכנית נעצרת באמצע בגלל שגיאה, וכתוצאה מכך החיבור למצלמה אינו נסגר כדרוש, והמצלמה הופכת בלתי זמינה, ולא ניתן להשתמש בה שוב כאשר נריץ שוב את תוכניתנו.

לכן, כתבנו פונקציה שסוגרת את החיבורים הקיימים למצלמה, והופכת אותה לזמינה שוב.

פונקציה זו מהווה חלק מממשק המשתמש (GUI) שכתבנו עבור התוכנית, אשר מריץ פונקציה זו בכל מקרה של שגיאה בהרצת התוכנית. ככה אנו מוודאים כי המצלמה תמיד פנויה לשימוש לאחר הרצת התוכנית.

אופן פעולתה של הפונקציה:

הפונקציה מקבלת מסביבת העבודה (של Matlab) את כל החיבורים הפתוחים למצלמה בצורה של מערך. ואז, הפונקציה סוגרת כל חיבור, אחד אחד, בעזרת לולאה שרצה על כל איברי המערך.

Sample_h

מבנה הפונקציה:

```
function sample_h(color)
    conn = connectToDatabase();
    NUM_OF_SAMPLES = 1000;

    setupCamera();
    start(source_capture);
    trigger(source_capture);
    temp = getdata(source_capture);

    I = imshow(temp);
    [~,rect] = imcrop(I);

    h_max = 0; h_min = Inf;
    for i=1:NUM_OF_SAMPLES
        if (mod(i,75) == 0)
            disp(i);
        end
        trigger(source_capture);
        temp = getdata(source_capture);
        temp = imcrop(temp, rect);

        temp = rgb2hsv(temp);
        h = temp(:,:,1);

        if (strcmp(color, 'red'))
            for i=1:size(h,1)
                for j=1:size(h,2)
                    if (h(i,j) >= 0.5)
                        if (h(i,j) < h_min)
                            h_min = h(i,j);
                        end

                        else
                            if (h(i,j) > h_max)
                                h_max = h(i,j);
                            end
                        end
                    end
                end
            end
        else
            mini = min(min(h));
            maxi = max(max(h));

            if (mini < h_min)
                h_min = mini;
            end
            if (maxi > h_max)
                h_max = maxi;
            end
        end
    end
    sql = sprintf('ALTER TABLE `%s` CHANGE COLUMN `h_min` `h_min` FLOAT NOT NULL DEFAULT %f, CHANGE COLUMN `h_max` `h_max` FLOAT NOT NULL DEFAULT %f;', color, h_min, h_max);
    exec(conn, sql);
end
```

לקריאה עיונית:

<http://pastebin.com/Nq1qY3LY>

הסבר:

מטרתה של תוכנית זו היא לדגום לשמור את ערכי הגוון (H) של צבע נבחר, בטבלה שלו במסד הנתונים. בתוכנית זו נעשה שימוש אחד בלבד עבור כל צבע.

התוכנית מקבלת כארגומנט יחיד את שם הצבע כמחרוזת (string).

הצורך בתוכנית:

בתוכניתנו הראשית (Robilliard), נעשה שימוש במספר צבעים המופיעים בפריים הנקלט מהמצלמה.

לכל צבע יש טבלה במסד הנתונים שלנו (על שרת ה-MYSQL), ובטבלה זו מופיעים ערכי ה-HSV המינימאליים והמקסימאליים עבור צבע זה. כל שורה בטבלה מייצגת דגימה שלקחנו של הצבע לו שייכת הטבלה.

הערכים היחידים שאינם משתנים בטבלה עבור כל אחת מהדגימות הם ערכי הגוון (H) של הצבע, זאת מאחר ובפורמט ה-HSV, הגוון אינו משתנה בתאורות שונות, אלא רק העוצמה והבהירות.

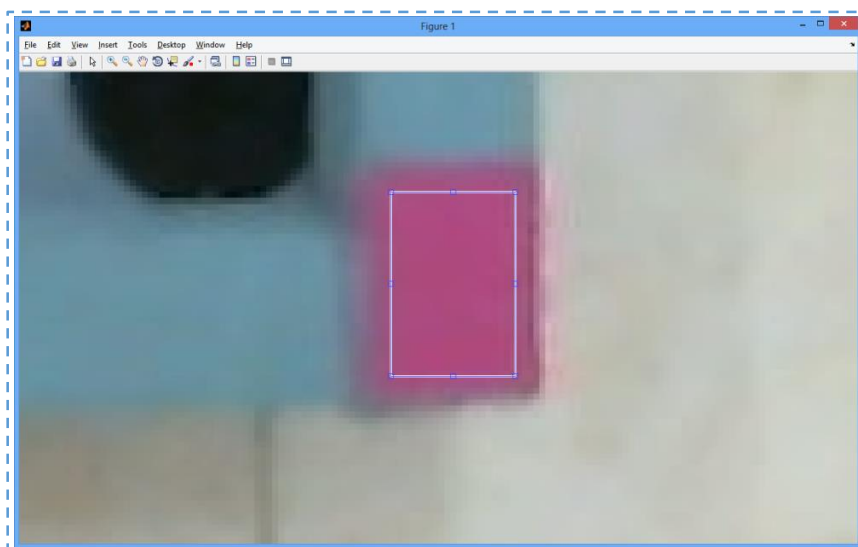
כאשר אנו רוצים להוסיף צבע חדש למסד הנתונים, עלינו ליצור לו טבלה, ולטבלה זו להכניס דגימות של אותו הצבע. כאמור, ערכי הגוון אינם משתנים בגלל שינוי תאורה, ולכן ניתן לדגום את הגוון של הצבע החדש רק פעם אחת.

לכן, אנו עושים שימוש בתוכנית זו שמגדירה את ערכי הגוון המינימאליים והמקסימאליים של הצבע הנבחר, על סמך מספר רב של דגימות שנלקחות בהרצתה.

אופן פעולתה של התוכנית:

בתחילת התוכנית מוגדר מספר הדגימות שיש לקחת עבור הצבע הנבחר, ומתבצעים חיבורים למצלמה ולשרת ה-MYSQL.

לאחר-מכן, התוכנית קולטת פריים יחיד, כך שנוכל לסמן את מיקום הצבע בפריים (מיקום הצבע חייב להישאר קבוע לכל אורך הרצת התוכנית), בעזרת ממשק פשוט שבו בוחרים את האזור הנ"ל.



כעת, התוכנית נכנסת ללולאה שמספר חזרותיה שווה למספר הדגימות שקבענו. בכל חזרה של הלולאה נקלט פריים חדש מהמצלמה, ומתבצעת דגימה נוספת של הצבע.

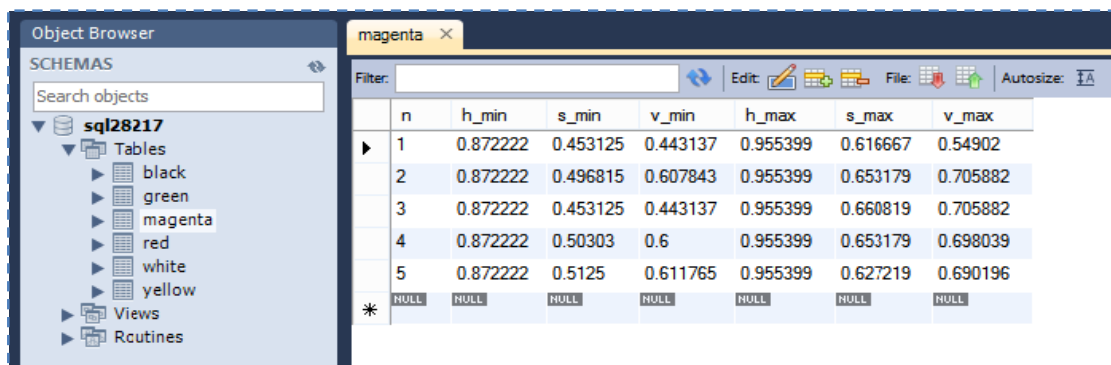
לצבע האדום יש כלל מיוחד עבור אופן הדגימה, אך לא נסביר את הסיבה לכך, מאחר והסברנו אותה כבר באחד הפרקים הקודמים.

מתוך כל הדגימות נבחרים ערכי הגוון הנמוך והגבוה ביותר, והתוכנית מגדירה את הערכים הללו כערכי הגוון הקבועים של הצבע הנדגם (ערכים אלו נקבעים כברירת מחדל במבנה הטבלה).

הפקודה ב-SQL שמעדכנת את הטבלה:

```
ALTER TABLE `color_name` CHANGE COLUMN `h_min` `h_min` FLOAT NOT NULL  
DEFAULT %f CHANGE COLUMN `h_max` `h_max` FLOAT NOT NULL  
DEFAULT %f;
```

להן תמונה של טבלת צבע מתוך מסד הנתונים:



	n	h_min	s_min	v_min	h_max	s_max	v_max
▶	1	0.872222	0.453125	0.443137	0.955399	0.616667	0.54902
	2	0.872222	0.496815	0.607843	0.955399	0.653179	0.705882
	3	0.872222	0.453125	0.443137	0.955399	0.660819	0.705882
	4	0.872222	0.50303	0.6	0.955399	0.653179	0.698039
	5	0.872222	0.5125	0.611765	0.955399	0.627219	0.690196
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Sample_sv

מבנה הפונקציה:

```
function sample_sv(color)
    conn = connectToDatabase();

    NUM_OF_SAMPLES = 1000;

    global source_capture;
    setupCamera();
    start(source_capture);

    trigger(source_capture);
    temp = getdata(source_capture);

    I = imshow(temp);
    [~,rect] = imcrop(I);

    max_val = [0,0]; min_val = [Inf, Inf];
    % max_val(1), min_val(1) -> s;
    % max_val(2), min_val(2) -> h;

    for i=1:NUM_OF_SAMPLES
        if (mod(i,50) == 0)
            disp(i)
        end
        trigger(source_capture);
        temp = getdata(source_capture);

        temp = imcrop(temp, rect);
        temp = rgb2hsv(temp);

        for j=2:3
            var = temp(:,:,j);

            mini = min(min(var));
            maxi = max(max(var));

            if (mini < min_val(j-1))
                min_val(j-1) = mini;
            end

            if (maxi > max_val(j-1))
                max_val(j-1) = maxi;
            end
        end
    end

    sql = sprintf('INSERT INTO %s (s_min, v_min, s_max, v_max)
VALUES (%f, %f, %f, %f);', color, min_val(1), min_val(2),
max_val(1), max_val(2));

    exec(conn, sql);

end
```

לקריאה עיונית:

<http://pastebin.com/zchzQNjj> .

הסבר על התוכנית:

מטרתה של תוכנית זו היא לדגום ולשמור את ערכי העוצמה (S) והבהירות (V) של צבע נבחר, בטבלה שלו במסד הנתונים. בתוכנית זו נעשה שימוש בכל פעם שהתאורה משתנה, ויש צורך לדגום מחדש ערכים אלו כדי שזיהוי הצבע יפעל כראוי.

הפונקציה מקבלת כארגומנט יחיד את שם הצבע כמחרוזת (string).

הצורך בתוכנית:

הצורך בתוכנית זו זהה לצורך בתוכנית לדגימת ערכי הגוון (sample_h), מלבד העובדה שערכי העוצמה והבהירות כן משתנים עקב שינויי תאורה, ולכן לתוכנית זו יש יותר משימוש אחד עבור כל צבע – בתוכנית נעשה שימוש בכל פעם שזיהוי הצבעים אינו תקין עקב שינויי תאורה.

אופן פעולתה של התוכנית:

אופן פעולתה של התוכנית זהה לאופן הפעולה של התוכנית לדגימת ערכי הגוון (sample_h).

ההבדלים היחידים בין התוכניות הם:

- שבתוכנית למדידת ערכי עוצמה ובהירות (sample_sv), הערכים הנמדדים הם העוצמה (S) והבהירות (V), ולא ערך הגוון (H).
- הפקודה ב-SQL שמעדכנת את הטבלה.

הפקודה ב-SQL שמעדכנת את הטבלה:

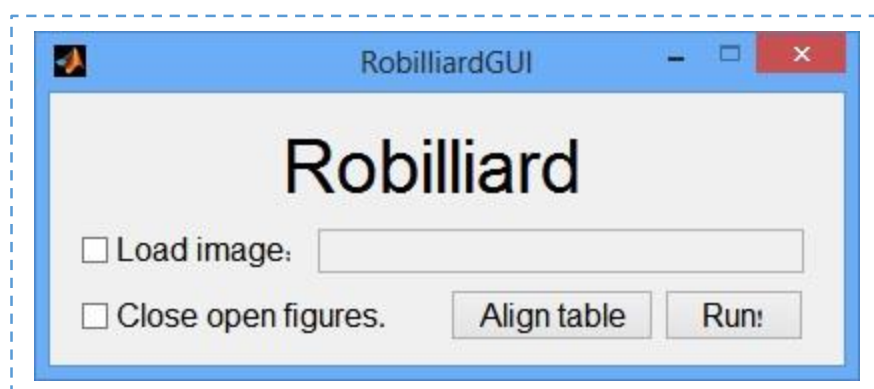
```
INSERT INTO color_name (s_min, v_min, s_max, v_max)
VALUES (%f, %f, %f, %f);
```

Robilliard GUI

הסבר על התוכנית:

לשם הקלה בניפוי באגים ובהרצת התוכנית מספר רב של פעמים, בנינו ממשק משתמש גרפי (Graphic User Interface) עבור התוכנית. מתוך הממשק ניתן להריץ את התוכנית במצב רגיל, במצב ניתור באגים (debug) ועוד.

לא נכניס כאן את הקוד של הממשק או את דרך פעולתו, מאחר וזו אינה רלוונטית להרצת התוכנית, והיא נועדה רק להקל עלינו בבנייתה.





לאן הגענו - סיכום

לאן הגענו?

לאחר שנה של עבודה, התוכנית מצליחה לזהות בהצלחה את כל האסטרטגיות הנתמכות האפשריות, כולל אסטרטגיות הכוללות שרשור, ויודעת להדריך את הרובוט כיצד לבצע את המהלך הנבחר.

על-אף ההישגים המרשימים שאליהם הגענו, ניתן עוד לעבוד על התוכנית ולשפרה: לא הספקנו לכתוב פונקציה שתחשב את כדאיות המהלכים ותבחר את המהלך הטוב ביותר, ולא הספקנו להוסיף סטטיסטיקה לפרויקט על אף שרצינו.

הקוד של התוכנית עצמה בנוי בצורה גמישה ומובנת, כך שניתן ללמוד ממנה ואף לשפרה – הפונקציה לבניית אסטרטגיה תומכת בהכנסת הרחבות הכוללות מהלכים מורכבים מסוגים נוספים.

כמו-כן, מרבית הפונקציה עליהן בנויה התוכנית הן פונקציה שאנו כתבנו בעצמנו, והן בנויות כפונקציות כלליות, מה שאומר שניתן להשתמש בהם גם בפרויקטים אחרים שאינם קשורים לביליארד.

בנוסף לספר הפרויקט כתבנו מדריך משתמש עבור כל אחת מהפונקציות שכתבנו, בשפה האנגלית, כך שנוכל לשתף את עבודתנו עם תלמידים מכל העולם.

את הפרויקט עצמו, ואת כל המסמכים הקשורים אליו שמרנו בשרת אחסון ברשת, כך שהם זמינים להורדה, צפייה ושימוש בכל עת – זוהי המתנה שלנו למגמה, ואנו מקווים כי התלמידים שילמדו בה בשנים הבאות יוכלו לעשות בקוד שלנו שימוש, בכדי להעצים את הפרויקטים שלהם ולהעלות את הרמה של המגמה אפילו יותר.

הפרויקט זכה להמון תשומת לב והערכה מצד מורים, אנשי צוות בבית-הספר, מצד הצבא ואפילו מצד משרד החינוך.

אנו גאים מאוד על השיגנו, נהנינו מבניית הפרויקט, ואנו שמחים כי בחרנו בפרויקט זה כפרויקט שלנו במגמה אלקטרוניקה. הבחירה ללמוד במגמה הוסיפה להנאה שלנו מהתיכון ומהלימודים בו, ולימדה אותנו המון.

הפרויקט ריתק אותנו ועבדנו עליו הרבה שעות מעבר לשעות המעבדה המוגדרות, ואנו זוכרים לילות שעבדנו בהם על הפרויקט למשך מספר רב של שעות, תוך כדי הנאה רבה. נושא עיבוד התמונה הוא נושא מרתק בתחום ההנדסה, ונשמח לעסוק בו גם בעתיד.

מעשיית הפרויקט זכורות המון חוויות טובות: העזרה והתמיכה ההדדית בכיתה, הצחוקים והשטויות, והזמן שטס. אפילו כתבנו תוכנה לשליטה על הרובוט בעזרת שלט ג'ויסטיק של XBOX, והסענו אותו ברחבי הכיתה ודרסנו אנשים.

אנו (ברק ועידן), נהנינו לעבוד בצוות, וכתיבת הספר העלתה בנו זכרונות מכל אורך השנה, ובעזרתה ראינו את כל מה שהצליח להשיג במהלך השנה.

לסיום, נרצה להודות למורנו במגמה, רפי אמסלם, על לימד אותנו שלוש שנים במגמה, ועל שסבל אותנו בשנה האחרונה ועזר לנו בכל שביקשנו בעת בניית הפרויקט. **תודה רפי!**