# פרויקט בראייה חישובית

# בנושא זיהוי אובייקטים

מגיש: דולב שפירא shdolev@post.bgu.ac.il

פרויקט זה הוכן בחלקו בשיתוף ליאור ספיר כחלק ממיני פרויקט "נושאים בבינה מלאכותית"

# תוכן עניינים


| | |
|---|---|
| שער | 1 |
| תוכן עניינים | 2 |
| מבוא | 3 |
| חבילות נחוצות | 4 |
| פירוט הקוד | 5-10 |
| Templates טובים | 11 |
| הוראות שימוש | 12 |
| בעיות פוטנציאליות | 13-14 |
| הצעות לשיפור | 15 |
| מקורות | 16 |

# מבוא

עבודתנו עוסקת בזיהוי אובייקט בתמונה (תיקרא בהמשך scene) בהינתן מספר תמונות המאפיינות את האובייקט (ייקראו בהמשך templates) בסביבת ROS.

למערכת מספר ערוצים אליה היא מאזינה:

- CONTROL_TOPIC – ערוץ בקרה המאפשר שליטה בפעולת המערכת.
- IMAGE_TOPIC – ערוץ ממנו נקראת ה scene.

למערכת ערוץ יחיד אליו היא מפרסמת:

- RESULT_TOPIC – ערוץ אליו היא מפרסמת את תוצאת הזיהוי.

תוצאת הזיהוי מתבטאת במערך של floats אשר בו שיעורי הנקודות המתארות את המרובע אשר תוחם את האובייקט ב scene.

במסמך זה נסביר על הנושאים הבאים:

- חבילות – באילו חבילות אנו משתמשים.
- הקבצים – מה מתואר בכל אחד ותפקידו.
- הקוד – תפקיד כל פונקציה והקשר ביניהן.
- Templates טובים – כיצד לבחור חלקים טובים לזיהוי וכיצד ליצור אותם.
- הוראות שימוש – כיצד להשתמש בקוד בצורה נכונה.
- בעיות פוטנציאליות – קשיים העלולים להתרחש בעת שימוש בקוד.
- הצעות לשיפור – הוספת תכונות שיהפכו את הקוד לידידותי לשימוש.
- מקורות – מקומות מהם לקחנו מידע וקוד.

# חבילות נחוצות

- ROS:

חבילה זו מספקת לנו יכולות כמו פרסום והאזנה ל – topics, בנוסף ליכולת לייצר Node.

- OpenCV:

חבילה זו מאפשרת להשתמש באובייקטים מסוג תמונה בצורה נוחה, לייצא אותם לקובץ, להציגם על המסך בקלות, ותכונות נוספות.

- Features2D:

חבילה זו, אשר נכללה בגרסאות מסוימות של OpenCV (נכון להיום 2.4.9-2.4.11), וייתכן שבגרסה אשר ברשותכם אינה נכללת, אחראית על סיפוק הממשק העיקרי בזיהוי האובייקטים הנמצא בשימוש בקוד. על מנת להקל את התהליך, בודדנו אותה עבורכם.

- Homography (nonfree):

חבילה אשר מאפשרת למצוא את ההטלה של template ב scene, בדומה ל Features2D, גם זו בעלת חשיבות קריטית לפעולת האלגוריתם, ונכללת בגרסאות מסוימות של OpenCV, על מנת להקל את התהליך, בודדנו אותה עבורכם.

- Image_Transport:

חבילה של ROS המספקת אבסטרקציה להאזנה ופרסום של תמונות. שימוש בחבילה זו מקנה גמישות בדרך העברת התמונות בין ה-nodes השונים.

- CV_Bridge:

חבילה אשר מספקת אבסטרקציה להמרה של הודעות תמונה של ROS (טיפוס sensor_msgs/Image) לתמונות של OpenCV (טיפוס Mat).

# פירוט הקוד

הקוד מחולק בצורה כזו שביצוע ה includes יתבצע מקבצי ה Header הרלוונטיים. אנו משתמשים במספר קבצים עיקריים:

1. object_recognition.cpp
2. object_recognition_node.cpp

יש לציין כי אנו מניחים את תקינות הקלט.

- globals.hpp

בקובץ זה אנו מגדירים משתנים גלובליים וקבועים שנרצה להשתמש בהם גם בשלב הזיהוי, וגם בשלבים אחרים:

1. NODE_NAME – שם ה Node בעת האתחול.
2. CONTROL_TOPIC– שם topic המשמש לבקרה. ה – node שניצור בהמשך יאזין ל topic זה, ויפעל בהתאם להוראות הבקרה שיתקבלו. אופי ההודעות העוברות ב – topic זה הן std_msgs/Int32.
3. IMAGE_TOPIC– שם topic אשר ממנו נרצה לקבל את התמונות בהן יתבצע הזיהוי, כלומר, ממנו ניקח את ה scene בכל פעולת זיהוי. אופי ההודעות שעובר ב – topic זה הוא sensor_msgs/Image.
4. RESULT_TOPIC – שם topic אליו נפרסם את תוצאות הזיהוי. תוצאת הזיהוי מורכבת ממערך של 8 floats, המכיל את שיעורי ה – x ושיעורי ה – y של פינות המרובע התוחם את האובייקט ב scene במידה והזיהוי הצליח. במידה והזיהוי נכשל, יישלח מערך המכיל 8 אפסים.
5. DELAY – תכונה המייצגת את מרווח הזמן בין הצגת תמונה הנקלטת מהמצלמה לתמונה הבאה המוצגת (תחת ההנחה כי המרווח אינו קטן ממרווח פרסום התמונות ל IMAGE_TOPIC).
6. WRITE_PATH – הכתובת אליו יש לכתוב את התמונה מ – IMAGE_TOPIC בהתאם לאות הבקרה.
7. TEMPLATE_QUANTITY – מתאר את מספר ה – templates מהם יתבצע זיהוי, שימו לב שעליכם לשנות את הגדרה זו בהתאם לכמות ה templates שיצרתם.
8. MAX_SAMPLE_COUNT – מספר הדגימות (scene שונים) המקסימלי שיש לבצע לפני סיום פעולת הזיהוי. פעולת הזיהוי צורכת משאבים רבים, ולכן יש צורך לבחור גבול על מספר הניסיונות לזיהוי שיש לבצע.
9. ACCURACY – פרמטר המתאר את יחס הנקודות המינימאלי אשר צריך להימצא בתוך המרובע הנפרש על ידי 4 הפינות, לבין אלו שמחוצה לו.
10. Scene_matches – מכיל אובייקטים מסוג Mat אשר בהם מוצגת התוצאה של הזיהוי במידה והתבצע זיהוי מוצלח.
11. Sample_count – משתנה העוזר לעקוב אחר מספר הדגימות שדגמנו בזיהוי הנוכחי.
12. Templates – מערך של אובייקטים מסוג Mat המייצגים את תמונות ה – templates שנבחרו.
13. Scene – אובייקט Mat המכיל את הדגימה הנוכחית.
14. Object_recognition_output – טיפוס המכיל את אחוז הנקודות שנמצאו בזיהוי אשר נמצאות בתוך המרובע הנפרש על ידי corners, ואת שיעורי נקודות המרובע אשר פורש את האובייקט ב scene הנוכחי.
15. Results – מערך של מבנים מסוג Object_recognition_output, המכיל את תוצאת זיהוי עבור כל template.
16. Result – תוצאת הזיהוי הטובה ביותר בהתאם למדד המוגדר. המדד אשר אנו הגדרנו הוא ה percentage הכי גבוה (אחוז הנקודות שנמצאות בתוך המרובע הנפרש על ידי הפינות).

- object_recognition_header.hpp

קובץ זה מכיל את כל אשר נחוץ לפעולת הזיהוי, פרט למשתנים הגלובליים וההגדרות אשר ב – globals.
בקובץ זה תמצאו הכרזות על הפונקציות בעזרתן פועל זיהוי האובייקטים (מתואר כאן הסבר על תפקידן ובהמשך תתואר פעולתן):

1. `bool point_is_inside(vector<Point2f> corners, Point2f pt)`

פונקציה זו אחראית על בדיקה האם נקודה נתונה pt נמצאת במרובע הנפרש על ידי corners. ישנה חשיבות לסדר corners אם נחשוב על פעולת הזיהוי כמרובע: 0 הוא הקדקוד השמאלי העליון, 1 הוא הקדקוד הימני העליון, 2 הוא הקדקוד הימני התחתון, 3 הוא הקדקוד השמאלי התחתון.

2. `float is_mostly_inside(vector<Point2f> sceneCorners, vector<Point2f> points)`

פונקציה זו מממשת את מדד הדיוק של הזיהוי הנוכחי – היא אחראית על חישוב אחוז הנקודות הנמצאות בתוך המרובע הנפרש על ידי sceneCorners מבין points. גם כאן ישנה חשיבות לסדר שליחת הקודקודים, אם כי כאן ישנה בדיקה אקטיבית האם הסדר תקין (יכול להיות זיהוי אשר הפינות ייצרו 8 לפי הסדר הקודם, זיהוי כזה הוא אינו זיהוי תקין).

3. `void object_recognizing(int template_num)`

פונקציה זו משמשת לחיפוש אובייקט המתואר על ידי ה – template המצוין ב – template_num במערך של templates בתוך scene. היא אחראית לאתחול results[template_num] עבור כל template, כלומר, במידה והזיהוי הצליח וחוקי (גדול ממדד הדיוק ופורש מרובע), יכיל את אחוז הדיוק לפי המדד שהגדרנו של הזיהוי, ופינות המרובע הפורש את ה template הנתון על ידי template_num בתוך scene.

4. `void object_recognition()`

פונקציה זו אחראית על ביצוע פעולת הזיהוי object_recognizing עבור כל template. במידה ונמצא זיהוי מוצלח, היא תאתחל את result להיות התוצאה הטובה ביותר לפי המדד שהגדרנו, בעוד אם הזיהוי נכשל, היא תאתחל את result להכיל אחוז 0, ובפינות את הנקודה (0,0) 4 פעמים.

- object_recognition_node_header.hpp

קובץ זה מכיל את כל אשר נחוץ להפעלת ה Node, פרט למשתנים הגלובליים וההגדרות אשר ב – globals.
בקובץ זה תמצאו הכרזות על הפונקציות בעזרתן פועל ה Node (מתואר כאן הסבר על תפקידן ובהמשך תתואר פעולתן):

1. `void init_templates()`

פונקציה האחראית על אתחול מערך ה – templates, אשר אותם יש לזהות בדגימות.

2. `void activate_control_callback(const std_msgs::Int32::ConstPtr& msg)`

פונקציה מסוג callback (הנקראת לאחר שהתפרסמה הודעה חדשה ב – CONTROL_TOPIC אליו מתבצעת האזנה) שתפקידה הוא הפעלת תכונותיה השונות של ה Node. בפונקציה זו נקבל הודעות מסוג std_msgs/Int32, ונפעל בהתאם לערכן:

0 – פעולת אתחול של משתני הבקרה (Reset).
1 – להציג את התמונה מ – IMAGE_TOPIC בהתאם ל – DELAY.
2 – כתיבה של תמונה בודדת מ – IMAGE_TOPIC ל – WRITE_PATH.
3 – ביצוע פעולת הזיהוי בהתאם ל – MAX_SAMPLE_COUNT.

3. `void read_image_callback(const sensor_msgs::ImageConstPtr& msg)`

פונקציה מסוג callback המקבלת כהודעה תמונה מ - IMAGE_TOPIC, ואחראית לביצוע ההוראה שנקלטה בידי topic ה – control (הצגת התמונה \ כתיבה \ ביצוע זיהוי).

4. `void analyze_object_recognition()`

פונקציה זו אחראית לקריאה לביצוע הזיהוי, והחלטה האם יש צורך בדגימה נוספת. אם הזיהוי נכשל, ואין התערבות מצד הבקרה (פקודת Reset), נמשיך לנסות לבצע זיהוי עד אשר יהיה זיהוי מוצלח, או שנגיע למספר הדגימות המקסימלי שהגדרנו ב – MAX_SAMPLE_COUNT.
במידה והזיהוי באיטרציה הנוכחית נכשל, נפרסם תוצאה הכוללת 8 אפסים. במידה והזיהוי הצליח, נפרסם תוצאה של קודקודי המרובע החוסם את האובייקט ל – RESULT_TOPIC.

- Object_recognition.cpp

קובץ זה למעשה אינו נמצא בשימוש, אך צורף על מנת שיהיה קל להפריד בין הקוד של זיהוי האובייקטים, ולזהות מאוחר יותר, בקובץ object_recognition_node.cpp את החלק השייך לזיהוי האובייקטים בקלות רבה יותר. כל החלקים המתוארים בקובץ זה נמצאים גם בקובץ object_recognition_node.cpp.

1. `bool point_is_inside(vector<Point2f> corners, Point2f pt)`

כפי שהזכרנו מקודם, פונקציה זו אחראית לבדיקה האם נקודה נתונה נמצאת בתוך המרובע הנפרש על ידי corners. דרך הבדיקה היא חישוב ערכי ה – x של הנקודות על צלעות המרובע תחת אותו ערך y, וחישוב ערכי ה – y של הנקודות על צלעות המרובע תחת אותו ערך x. כלומר, מדובר בלבדוק האם הנקודה נמצאת בין הצלע ה"עליונה" ל"תחתונה", והאם היא נמצאת בין הצלע ה"ימנית" ל"שמאלית" דרך משוואות הקווים הישרים העוברים בין ה corners המתאימים.

במידה ומתקבל כי הנקודה בתוך המרובע נחזיר true, אחרת נחזיר false.

2. `float is_mostly_inside(vector<Point2f> sceneCorners, vector<Point2f> points)`

כפי שהסברנו, פונקציה זו אחראית למציאת אחוז הנקודות הנמצאות בתוך המרובע החסום אשר נמצאה להן התאמה ב – scene הנתון.

ראשית מתבצעת בדיקה לוודא כי סדר הקדקודים חוקי, ולאחר מכן אנו מחשבים את האחוז.

3. `void object_recognizing(int template_num)`

פונקציה זו מאתחלת את results[template_num] עבור ה – template הנוכחי הנבדק בהתאם לתוצאת הזיהוי של ה – template אשר מצוין על ידי template_num ב – scene. לשם מימוש פונקציה זו אנו נעזרים בטכניקת זיהוי תבניות בתמונות.

על מנת לבצע את הזיהוי אנו נעזרים במספר כלים:

- SiftFeatureDetector - באמצעות כלי זה אנו מזהים נקודות מתפתח (key points) גם ב-template וגם ב – scene. יש לשים לב כי ישנם סוגים שונים של FeatureDetector וכי באמצעות ניסוי וטעייה מצאנו כי SIFT הוא הטוב ביותר לשימושים שבדקנו. ייתכן כי עבור משימות שונות לזיהוי אובייקטים יידרש Feature Detector אחר.
- SiftDescriptorExtractor - בהינתן ה – key points שמצאנו בשלב הקודם נשתמש בכלי זה על מנת לחשב את ה-Descriptors גם עבור ה-template וגם עבור ה-scene. יש לשים לב כי במידה ונעשה שינוי ב – FeatureDetector המוזכר בסעיף הקודם יש צורך לשנות גם את ה-Extractor בהתאם.
- FlannBasedMatcher - בהינתן descriptors עבור ה-template ועבור ה-scene כלי זה מבצע התאמה ביניהם. גם כאן, יש מספר סוגים של Matchers וייתכן כי עבור מטרות שונות יהיה צורך להשתמש ב – Matchers שונים.
- findHomography - פונקציה זו מוצאת את ה-perspective transform הנדרש לעשות בין שני וקטורים (הרכבנו וקטור עבור ה-template ועבור ה-scene), כלומר, את ההטלה של האובייקט המזוהה על ידי template לתוך scene.
- perspectiveTransform - פונקציה זו מבצעת בפועל את ההטלה שנמצאה באמצעות findHomography.

4. `void object_recognition()`

פונקציה זו פועלת בצורה ברורה למדיי, כפי שתואר כבר. להבנה עמוקה יותר ניתן להסתכל על הקוד

- Object_recognition_node.cpp

קובץ זה מייצג את מימוש ה – node אשר אחראי להפעלת זיהוי האובייקטים ומספק דרך נוחה להצגת תמונה מ – IMAGE_TOPIC, כתיבה ממנו ל – WRITE_PATH.

ה – Node נקרא בשם המצויין על ידי NODE_NAME, מאזין ל – IMAGE_TOPIC, CONTROL_TOPIC ומפרסם את תוצאות הזיהוי (במידה והזיהוי הופעל) ל – RESULT_TOPIC.

כל אחת מהפקודות הללו ניתן לבצע דרך שליחת std_msgs/Int32 מתאים דרך CONTROL_TOPIC.

לכל אחד מהפקודות הללו יש משתנה בוליאני המתאים לפקודה, פרט לפקודה 0, אשר מאפסת את כולם.

בקובץ זה נמצא את כל הפונקציות אשר תוארו ב – object_recognition.cpp ובנוסף להן, נמצא את:

1. `void init_templates()`

פונקציה האחראית על קריאת ה – templates.

2. `void activate_control_callback(const std_msgs::Int32::ConstPtr& msg)`

נוסיף על התיאור אשר כבר מוזכר כי בעת אתחול משתנה הבקרה לא נקראת פונקציה אשר מפעילה את אשר צוין, אלא כאשר תתבצע ה read_image_callback תבוצע הפונקציונליות המבוקשת.

3. `void read_image_callback(const sensor_msgs::ImageConstPtr& msg)`

התיאור אשר מוזכר קודם מפורט מספיק, להבנה עמוקה יותר, ניתן לפנות לקוד.

4. `void analyze_object_recognition()`

נוסיף על התיאור שפורט קודם כי הטיפוס אשר מפורסם ל RESULT_TOPIC הוא std_msgs/Gloat32MultiArray, אשר מכיל 8 תאים. תוכלו בקלות להרחיב את הפרסום כך שיישלח גם האחוז במידה ותחליטו כי מידע זה רלוונטי עבורכם.

5. `int main(int argc, char **argv)`

פונקציה זו אחראית נקראת בעת הפעלת ה – Node. היא מבצעת את אתחול ה – Node, ואת ההרשמה להאזנה ל – topics אשר הזכרנו, וממשיכה להתקיים עד אשר מפסיקים אותה.

# Templates טובים

במהלך העבודה גילינו כי לבחירת templates ישנה חשיבות מכרעת בתהליך הזיהוי. על מנת לזהות באפקטיביות את האובייקט יש צורך בבחירה מושכלת של templates. בחלק זה נדריך אתכם כיצד ליצור templates אשר "קל" יותר לזהות מאחרים.

ראשית, נגדיר שני מושגים: מאפיינים סטטיים ומאפיינים דינמיים.

מאפיין סטטי – מאפיין אשר תחת אותם תנאי תאורה (מיקום מקור האור, והספקטרום הנפלט ממנו), ותחת שינוי סביבה לא ישתנה משמעותית.

מאפיין דינמי – מאפיין אשר תחת אותם תנאי תאורה (מיקום מקור האור, והספקטרום הנפלט ממנו), ותחת שינוי סביבה ישתנה משמעותית.

לדוגמא: נחשוב על בקבוק קוקה קולה ריק. על הבקבוק ישנו הפקק, ה"סרט" עליו הלוגו ומידע על המשקה. אלו הם מאפיינים סטטיים. זאת אומרת, תחת אותם תנאי תאורה הם ייראו בקירוב אותו הדבר כל עוד הם באותו תווך (אוויר לעומת מים למשל). לעומת זאת, אם נסתכל על הבקבוק עצמו, נראה כי הוא שקוף. שקיפות היא תכונה "רעה" לבחירת template, שכן אם הסביבה משתנה, כך גם נצפה ל – template שונה, שכן המראה הנשקף דרך הבקבוק ישתנה.

ככלל, תמיד נחפש מאפיינים סטטיים ולא מאפיינים דינמיים.

כעת נפרט כיצד לבחור template טוב:

1. צלמו מהמצלמה ממנה מתבצע הזיהוי של האובייקט (ניתן להיעזר בתכונת ה – write).
2. צלמו מזוויות שונות אשר הרובוט יכול להימצא בהן.
3. צלמו ממרחקים שונים אשר הרובוט יכול להימצא בהם.
4. במידה והאובייקט יכול להימצא בתנאי תאורה שונים, צלמו template עבור כל תנאי תאורה אפשריים אשר הרובוט יכול להימצא בהם בעת זיהוי האובייקט.
5. אין להשתמש ב – template "רועש" מדי.
6. על ה – template להיות ברזולוציה נמוכה מרזולוציית המצלמה, אך ברזולוציה מספיק גבוהה כך שיתבצע זיהוי בקלות, זאת על ידי ניסוי וטעייה.
7. בכל template צריך להופיע לכל הפחות מאפיין סטטי אחד, וכמה שפחות מאפיינים דינמיים.
8. על המאפיין הסטטי להיות בגודל פרופורציונלי לגודל האובייקט (אם הוא קטן מדי, לא נצליח לזהות מרחוק).
9. השתדלו להכיל גבולות בולטים (הבדלי צבע) ב – templates אשר אתם יוצרים, גבולות הם סימנים קלים לזיהוי בידי האלגוריתם.
10. אל תשתמשו בחלקים עם חומרים מבריקים כ – templates, הם נחשבים בגדר מאפיינים דינמיים.
11. אל תשתמשו בחלקים עם חומרים שקופים כ – templates, הם נחשבים בגדר מאפיינים דינמיים.
12. השתמשו ב – templates עם רמות כלליות שונות, לדוגמא, בעין – אישון, אישון וקשתית, וכל העין (לא כולל עפעפיים), כמובן זאת תחת ההנחה שהעין לא משקפת ומבריקה (שהיא שגויה, אך לשם הדוגמא נניח כי אין זה כך).
13. אם יש ברשותכם האפשרות לשלוט בזמני חשיפה של המצלמה, העדיפו את זה אשר יגרור מעט רעש, גם בעת צילום ה – templates וגם בעת ביצוע הזיהוי.
14. ערכו את התמונה, הסירו מאפיינים דינמיים כמה שניתן והשאירו מאפיינים סטטיים, כלומר, בצעו חיתוך על התמונה (cropping).

במידה ותיצמדו לכללים אלו, סביר להניח שתצליחו לזהות את האובייקט בקלות למדיי.

# הוראות שימוש

1. העתיקו את התיקייה תחת /catkin_ws/src.
2. הגדירו את ה – topics, paths כראות עיניכם (ב globals.hpp).
3. צרו templates טובים וקלים לזיהוי בעזרת הדגשים המפורטים בפרק הקודם.
4. שנו את הקוד ב – ()init_templates, ובמידה ויש צורך שנו את TEMPLATE_QUANTITY.
5. בצעו catkin_make.
6. הפעילו את הסביבה (roscore, roslaunch…).
7. הפעילו את ה Node: roslaunch object_recognition object_recognition_launch.launch
8. על מנת להציג על המסך תמונות מ – IMAGE_TOPIC אשר קבעתם שלחו ל - CONTROL_TOPIC הודעה מסוג std_msgs/Int32 עם התוכן 1. ניתן לעשות זאת על ידי:
   rostopic pub CONTROL_TOPIC std_msgs/Int32 1 כאשר CONTROL_TOPIC הוא השם אשר קבעתם.
   על מנת לכתוב תמונה מ – IMAGE_TOPIC ל – WRITE_PATH אשר קבעתם שלחו ל - CONTROL_TOPIC הודעה מסוג std_msgs/Int32 עם התוכן 2. ניתן לעשות זאת על ידי:
   rostopic pub CONTROL_TOPIC std_msgs/Int32 2 כאשר CONTROL_TOPIC הוא השם אשר קבעתם.
   על מנת לבצע זיהוי בתמונה מ – IMAGE_TOPIC אשר קבעתם שלחו ל - CONTROL_TOPIC הודעה מסוג std_msgs/Int32 עם התוכן 3. ניתן לעשות זאת על ידי:
   rostopic pub CONTROL_TOPIC std_msgs/Int32 3 כאשר CONTROL_TOPIC הוא השם אשר קבעתם. שימו לב, על מנת לכתוב את תוצאת הזיהוי, יש צורך בעריכת הקוד (מסומן לכם בהערות היכן יש לעשות uncomment).
   על מנת לאתחל את משתני הבקרה (להפסיק הצגת התמונה מהמצלמה או פעולה קיימת של פעולה אחרת) שלחו ל - CONTROL_TOPIC הודעה מסוג std_msgs/Int32 עם התוכן 0. ניתן לעשות זאת על ידי:
   rostopic pub CONTROL_TOPIC std_msgs/Int32 0 כאשר CONTROL_TOPIC הוא השם אשר קבעתם.

# בעיות פוטנציאליות

בעיית הזיהוי היא בעיה לא קלה. מעבר למגבלה חישובית (זמן ריצה) ישנן מגבלות נוספות המקשות על זיהוי האובייקטים:

1. קצב העברת התמונות איטי למדיי

בשל מגבלות הנובעות מהתקשורת, קצב הקליטה, וקצב השליחה של הנתונים, קצב העברת התמונות לא גבוה במיוחד, מה שמאריך עוד יותר את תהליך הזיהוי.

2. תנאי תאורה משתנים

הרובוט עלול להימצא בסביבות שונות, הנמצאות תחת תנאי תאורה שונים, כמו למשל אור שמש ותאורת פלורסנט. תחת תאורות שונות עצמים נראים שונה.

3. אובייקטים מבריקים \ שקופים שיש לזהות

סיפקנו הדרכה כיצד לבחור templates אשר יעבדו, אך הסברנו כי עצמים מבריקים \ שקופים אינם עוברים זיהוי בצורה טובה, שכן תחת סביבות שונות, הצורה בה הם נתפסים משתנה.

4. קצב עיבוד התמונות אינו מהיר.

תהליך הזיהוי אינו מהיר. ככל שישנם יותר templates, תהליך הזיהוי איטי עוד יותר.

5. לא ניתן לטפל בכמה callbacks במקביל במידה ואין תלות ביניהם (באמצעים טריוויאליים)

במהלך עבודתנו שמנו לב כי המערכת המרכיבה את ה – Node מבצעת callbacks בדומה לביצוע interrupts. בעת ביצוע ה – callback לא ייקרא callback אחר בשל masking שנעשה מאחורי הקלעים.

6. רזולוציית תמונה מצולמת (scene) נמוכה מדי

המצלמה מכוונת (כאשר בדקנו לאחרונה את הנתונים) לרזולוציה נמוכה, וגם הרזולוציה המקסימלית שלה אינה גבוהה במיוחד. ככל שהרזולוציה גבוהה יותר, כך ניתן לבצע זיהוי למרחק גדול יותר (תחת הנחה כי ישנם תנאי תאורה מספקים והרעש בתמונה מזערי).

7. ערך ISO גבוה מדי הנועד לפצות על זמן חשיפה קצר מדי

ישנו פרמטר במצלמה השולט על רגישות החיישן לאור. פרמטר זה נקרא ISO. כאשר מגבירים את רגישות החיישן לאור, אנו מקבלים אפקט המקביל לחשיפת החיישן לפרק זמן רב יותר, אך בניגוד לחשיפה ארוכה, מתווסף רעש לתמונה. רעש זה עלול להפריע לתהליך הזיהוי.

8. בחירת templates לא טובה

הכללים שהבאנו נועדו לשפר משמעותית את סיכויי תהליך הזיהוי, אך לא מצאנו האם ניתן לחזות מראש את הצלחתו של template מסויים.

9. צילום במצלמה של הרובוט לא תמיד נוחה

כפי שהמלצנו, יש לצלם את התמונות במצלמה שממנה יתבצע הזיהוי. לעיתים לא נוח לעשות זאת, שכן קשה להרים את הרובוט לקבלת templates מפורטים.

10. אין התחשבות בגודל המרובע המזוהה

המדד להצלחת הזיהוי אשר הגדרנו לא מתחשב בגודל המרובע הנפרש, אלא רק באחוז הנקודות המושלכות לתוכו בהצלחה. לא נתקלנו בין הניסויים שלנו במקרים רבים שהמרובע היה גדול משמעותית (1 מתוך 125) מאשר האובייקט שיש לזהות.

11. בעיות סינכרוניזציה וכשלים טכניים

בעת העבודה נתקלנו בבעיות אשר לא הצלחנו לזהות את מקורן:

- כשל בפונקציה IMWRITE מחבילה OPENCV.

- כשל בשליחת הודעות ב – ROS: בעת ביצוע פרסום, גילינו כי פעמים רבות (רוב הפעמים למעשה) הפרסום אינו נקלט, ועל כאן יש צורך לחזור על הפרסום מספר פעמים רב מאוד, שגוזל זמן ריצה יקר.
  כמו כן, גם אם הנתון יפורסם, אין הבטחה ברורה כי ייקלט בצד השני.

# הצעות לשיפור

הצעותינו לשיפור יתמקדו בחלק מהקשיים אשר תוארו בפרק הקודם:

1. שיפור הזיהוי תחת תנאי תאורה שונים

הדרך הקלה לפתרון בעיה זו היא צילום templates תחת כל תנאי תאורה אפשריים. כאשר ניסינו לבצע זיהוי בעזרת מפת גבולות, נכשלנו בזיהוי. נדרש מחקר על שיטות זיהוי נוספות אשר בעזרת מפת גבולות יתאפשר זיהוי אובייקטים, שכן אלו נשארים קבועים תחת רוב תנאי התאורה (פרט לאובייקטים מבריקים / שקופים).

2. אובייקטים שקופים / מבריקים שיש לזהות

נסו לחפש מאפיינים סטטיים של אובייקטים אלה, או אם אתם מצפים למצוא אותם במקום קבוע, מאפיינים של הסביבה אשר יסמנו נקודות ציון בדרך לאובייקט.

3. קצב עיבוד התמונות איטי

ניתן לנסות לשנות את הקוד ולבצע חישוב מקבילי (תחת ההנחה שאין מגבלת חומרה המונעת זו) – הפעלת thread שיחשב את תוצאת החישוב לכל template בשילוב עם barrier כדי לוודא ש – results אותחל כראוי.

4. רזולוציית תמונה נמוכה

בקבצי ההפעלה ניתן לערוך את רזולוציית המצולמת מהמצלמה. כמובן ישנו גבול עד כמה ניתן להגדילה, אך שיפור הרזולוציה תגרור שיפור בזיהוי (תחת ההנחה כי אין רעש רב ותנאי התאורה הינם זהים).

5. ערך ISO גבוה מדי הנועד לפצות על זמן חשיפה קצר מדי

בררו עם יצרן המצלמה כיצד ניתן להתאים את זמן החשיפה בצורה דינמית. חשיפה ארוכה "תמרח" אובייקטים הנעים בתמונה. ניתן לפתור בעיה זו גם בעזרת שיטות שונות המאחדות מידע מכמה תמונות (דגימות) לתמונה בודדת עם רעש מופחת, כמו Super-Resolution.

6. בחירת templates לא טובה

נסו את ה – templates אשר צילמתם לפני שתמשיכו הלאה. לבחירת ה – templates חשיבות מכרעת בתהליך הזיהוי.

7. צילום במצלמה של הרובוט לא תמיד נוחה

אם פועלים בשיטה דומה לזו אשר צוינה קודם (Super-Resolution) ייתכן וניתן יהיה להשתמש לזיהוי גם בתמונות ממצלמות אחרות, וכך לאפשר גמישות רבה יותר. שימו לב, במידה ותעשו כן, ייתכן ותצטרכו לשנות את ה – templates כך שיעברו את התהליך גם כן.

8. אין התחשבות בגודל המרובע המזוהה

ניתן לשנות את פונקציית המדד כך שתיתן משקל לשטח המלבן ולאחוז הנקודות המושלכות, כך לשפר את דיוק הזיהוי בפועל.

# מקורות


- Feature Matching
  http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
- Feature Detectors
  http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html
- Common Interfaces of Descriptor Extractor
  http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_extractors.html
- Common Interfaces of Descriptor Matchers
  http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html
- SIFT
  http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- Distinctive Image Features from Scale-Invariant Keypoints (David G. Lowe 2004)
  http://www.cs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf
- Feature matching with FLANN
  http://docs.opencv.org/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html#feature-matching-with-flann
- Features2d + Homography Tutorial
  http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html