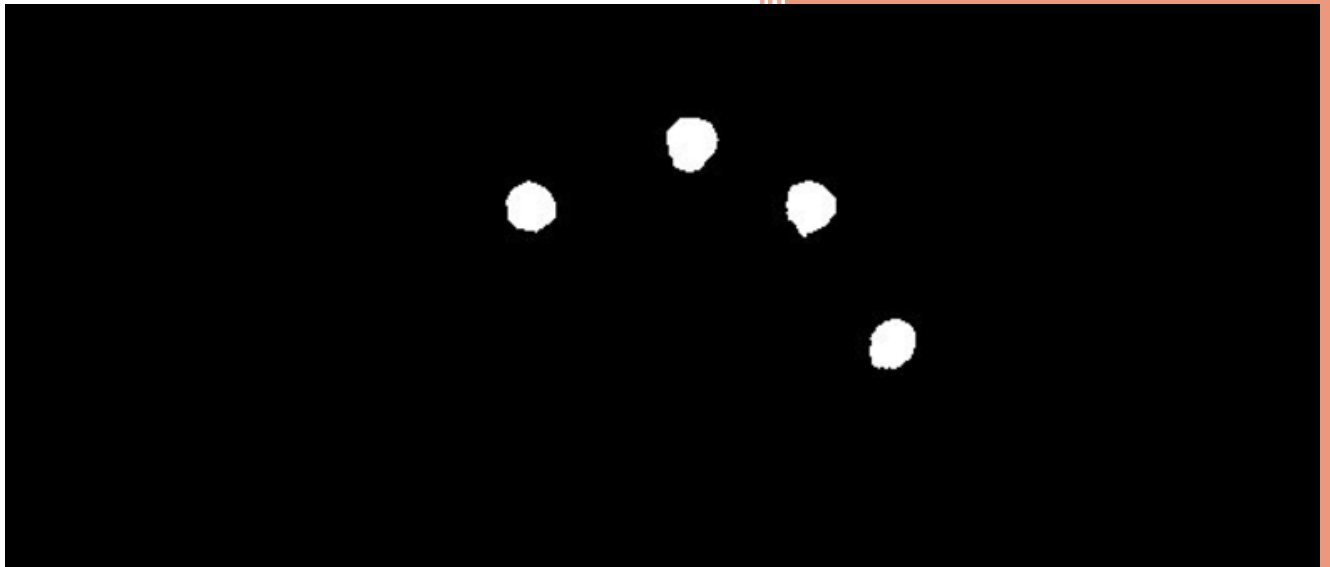


פרוייקט בעיבוד תמונה  
234329

## זיהוי עמיתים



נר דזייטש וספיר כהן  
המעבדה לעיבוד תמונה, הטכניון  
פרוייקט בעיבוד תמונה 234329

## תוכן עניינים

<a href="#">2</a>	הקדמה	❖
<a href="#">3</a>	המצלמה – PI NOIR CAMERA V2	❖
<a href="#">4</a>	עיבוד תמונה	❖
<a href="#">4</a>	OpenCV	○
<a href="#">4</a>	תהליך העיבוד	○
<a href="#">6</a>	אופטימיזציות	❖
<a href="#">6</a>	כדור פינג פונג ירוק	○
<a href="#">7</a>	ניסיון 1	○
<a href="#">8</a>	ניסיון 2	○
<a href="#">9</a>	ניסיון 3	○
<a href="#">10</a>	מדריך למשתמש	❖
<a href="#">11</a>	סיכום	❖

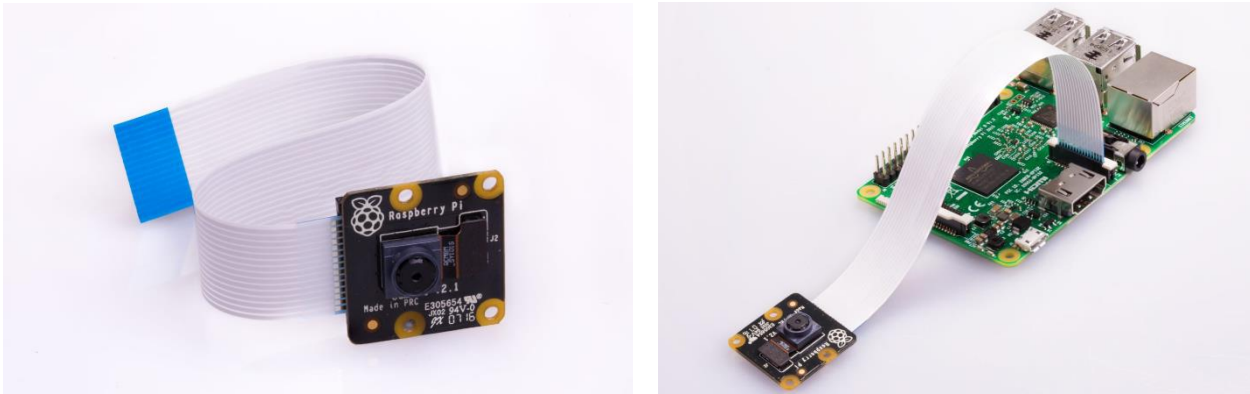
## הקדמה

פרוייקט זה עוסק בזיהוי עצם בנחיל של עצמים דומים. מטרת הפרוייקט הינה לשרת נחיל של רחפנים, כך שיוכלו לרחף כשיירה מבלי להיתקל אחד באחר.

כל רחפן נושא מעליו מחשב רסברי פאי ולו מותקנת מצלמת PI NOIR CAMERA V2 המצלמת ללא הפסקה בתדירות גבוהה יחסית. התמונה עוברת עיבוד שבסיומו ניתן לדעת כמה עמיתים, אם בכלל, נמצאים בקרבה.

מכיוון שצורתו של רחפן אינה זהה מכל זווית צילום, החלטנו שכל רחפן ישא כדור, אשר יראה אותו הדבר מכל זווית צילום שהיא. בכך פישטנו את העיבוד, שיתעסק במציאת עיגולים בכל תמונה.

## המצלמה – PI NOIR CAMERA V2



מודול ה PI NOIR CAMERA V2 הוא תוסף שתוכנן במיוחד עבור Raspberry Pi v2 ולו יתרונות רבים עבור הפרוייקט שלנו.

- **מצלמת 8 מגה פיקסל המסוגלת לצלם תמונות באיכות גבוהה של 3280x2464 פיקסלים.**  
על מנת שנוכל לזהות ביעילות את העצמים, נראה שהרזולוציה תהיה באיכות גבוהה ככל שניתן, בתנאי שקצב הצילום נשאר מהיר יחסית.
- **כרטיס צילום בגודל 25X23 מ"מ ובמשקל 3 גר' בלבד.**  
מכיוון שהרחפן יכול לשאת על עצמו משקל מינימלי, חשוב שהרכיבים עליו יהיו קלים במיוחד.
- **מכילה מסנן אינפרא-אדום.**  
האינפרא האדום עובד על איתור גופים הפולטים חום. המצלמה ממירה הפרשי חום שונים לתמונה ועל ידי כך ניתן לראות תמונה בחושך מוחלט.

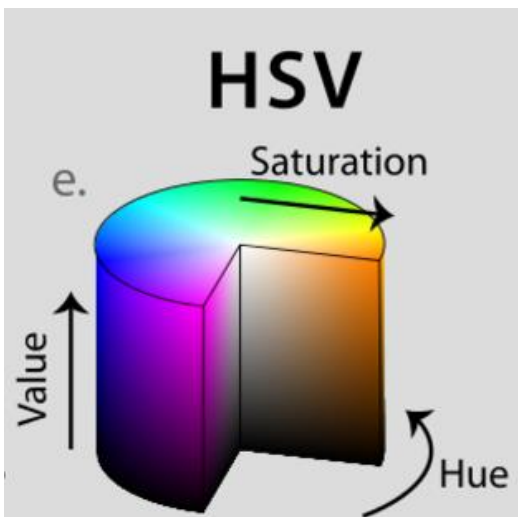
## עיבוד תמונה

### OpenCV

OpenCV הינה סיפרייה שנועדה לפתח יישומים של ראייה ממוחשבת ומוכוונת בעיקר עבור ניתוח ועיבוד תמונה בזמן אמת. בפרוייקט השתמשנו בסיפרייה זו על מנת להקל על תהליך העיבוד.

### תהליך העיבוד

1. המרת התמונה מפורמט BGR ל HSV  
אחת הדרכים הנפוצות לעבד תמונה באמצעות OpenCV היא להמיר תחילה את התמונה לפורמט HSV, בכך ייצוג הצבעים הינו קל יותר.



### 2. Threshold

נגדיר טווח של צבעים ירוקים (נסביר מדוע בחרנו דווקא בצבע הירוק תחת אופטימיזציות) ונבצע Threshold על התמונה.

את טווח הצבעים ניתן לחשב באמצעות color picker.



### 3. חיפוש של עיגולים בגדלים שונים בתמונה

```
std::vector<cv::Vec3f> circles;
cv:HoughCircles(greenHueImage, circles, CV_HOUGH_GRADIENT, 1, greenHueImage.rows/8, 100, 20, 0, 0); //detect circles

if (circles.size() == 0) {
    cout<<"Green blobs didn't found."<<endl;
    return 0;
} else {
    cout<< circles.size() << " blobs were detected."<<endl;
}

for (size_t currCircle = 0; currCircle < circles.size(); ++currCircle) { //Loop all over detected circles and outline then on the original image
    cv::Point center(std::round(circles[currCircle][0]), std::round(circles[currCircle][1]));
    int radius = std::round(circles[currCircle][2]);
    cv::circle(originalImage, center, radius, cv::Scalar(0, 255, 5));
}
```

תחילה השתמשנו בפונקציה HoughCircles השייכת לסיפריית OpenCV על מנת לזהות עיגולים בתמונה.

לאחר כמה בדיקות, גילינו כי הזיהוי לא עבד בצורה מדוייקת לטווחים של כמה מטרים, העיגולים נתפסו בתמונות כ"ירחים" ולכן עברנו להשתמש ב `erode`.

פונקציה זו גם כן של סיפריית OpenCV, מקבלת באחד הפרמטים שלה לפי איזו צורה לבצע "אכילה" (עיגול במקרה שלנו), עוברת על התמונה ו"אוכלת" את הקצוות של העצמים שזוהו.

אם בסופו של דבר נשארו בתמונה נשארו כתמים לבנים, נסיק כי נמצאו עיגולים, שכן כל קצוותיהם "נאכלו" בצורה סימטרית ולכן לא נעלמו מהתמונה לחלוטין.

```
//convert input image to HSV
cv::cvtColor(image, hsvImage, cv::COLOR_BGR2HSV);

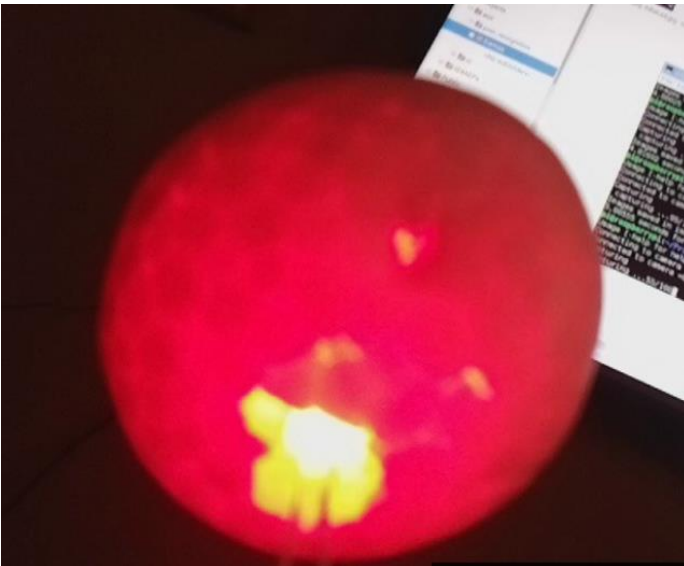
//Threshold the HSV image, keep only the green pixels.
cv::inRange(hsvImage, cv::Scalar(45, 50, 102), cv::Scalar(75, 255, 255), greenHueMask);

cv::erode(greenHueMask, greenHueMask, cv::Mat::ones(6,6,CV_8U));
```

### כדור פינג פונג ירוק

החלטנו כי העצם שעליו לזהות הינו כדור, דבר שהיה דיי ברור משום שצורתו נראית  
זוה מכל זווית שהיא. לאחר שלמדנו להכיר את המצלמה, בחנו את הכדורים השונים.

חשבנו על כדורים אדומים, זוהרים, כדורי קלקר שהכנסנו לתוכם לד אינפרא-אדום,  
לד צהוב, לד אדום.



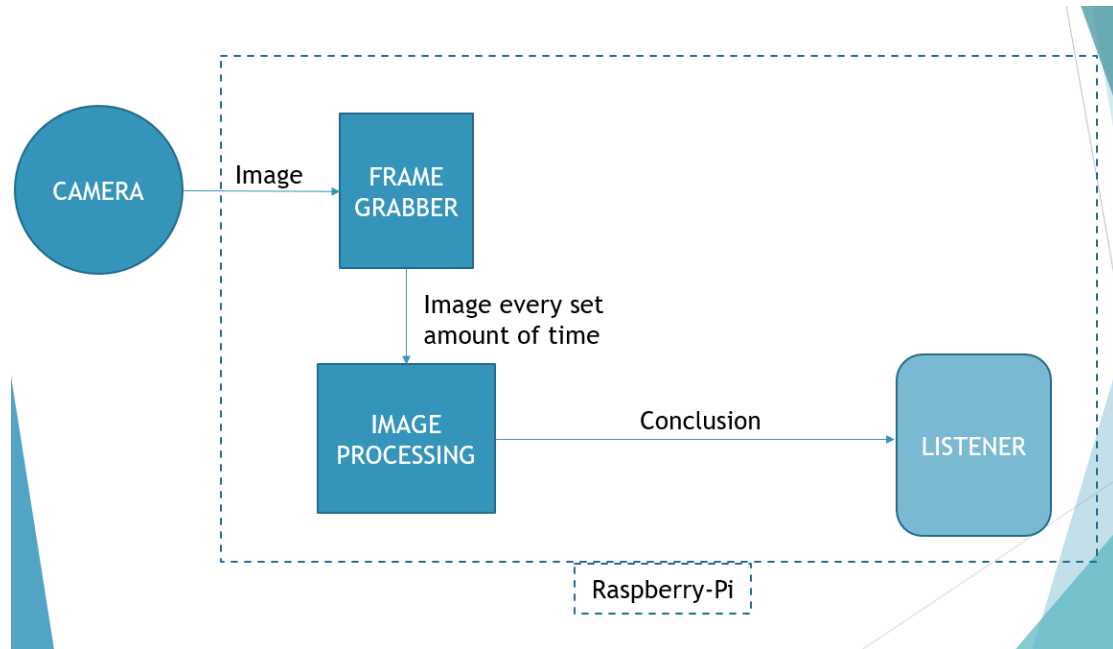
גילינו שעם נורת הליד בתוך הכדור, התאורה לא מתפשטת בצורה אחידה והכדור  
נראה דונה מזוויות שונות.

גילינו שבתאורות יום ולילה, צבעי התמונה משתנים.

את ההחלטה הסופית קיבלנו כאשר הסעלנו את המצלמה באור השמש, והבחנו כי  
הצמחייה בחוץ נראית למצלמה בגוונים אדומים. אז הבנו את ההשפעה של מצלמת  
הנואר המצלמת באינפרא-אדום, והחלטנו שהכדור שנוזהה יהיה בצבע ירוק, שכן  
באור יום הוא יהיה בולט מאוד בשטח הפתוח שבו ירחף הרחפן.

## ניסיון 1

תחילה, הרעיון העיקרי היה למקבל את הצילום עם עיבוד התמונה. כלומר, יפעלו שני מודולים שונים במקביל, כאשר האחד הינו מודול המצלמה והאחר מודול עיבוד התמונה.



מודול המצלמה אחראי על הדלקת המצלמה, וצילום ללא הפסקה של תמונות סטילס ברזולוצית 1080x1920. בכך הגענו ל26 פריימים לשניה.

מודול העיבוד אחראי על קבלת תמונה, ניתוח ופרסום של מספר העצמים שזוהו בתמונה.

הבעיה העיקרית היתה, העברת התמונה ממודול המצלמה למודול העיבוד. מכיוון שתמונה הינה בעצם מטריצה של פיקסלים שכל פיקסל הינו מערך של תוים, ניסינו במודול המצלמה לעבור על המטריצה ולשרשר את התוים למחרוזת. את המחרוזת ניתן לשלוח באמצעות ספריית IS4AEPS למודול עיבוד התמונה ושם להמיר את המחרוזת חזרה למטריצה המייצגת תמונה.

מפאת גודלה הרב של המטריצה, מעבד הרסברי הגיע ל 100% והתוכנית נתקעה.



## ניסיון 2

בניסיון השני ניסינו להשתמש ב boost serialization. הרעיון העיקרי של Boost serialization הוא archive.

Archive זהו רצף של בתים שמייצגים אובייקטים C++. אובייקט יכול לעבור סריאליזציה ולהתווסף ל archive, ולאחר מכן להיטען מה archive במטרה לשחזר את מה שנשמר קודם לכן.

הרעיון שלנו היה שמודול המצלמה ישמור את התמונה בבתים ב archive ומודול העיבוד יטען את הבתים ויקבל את התמונה. ניסיון זה אמנם עבד, אך באיטיות רבה.

```
class SerializableMat: public cv::Mat
{
public:

//----- Methods -----

    SerializableMat() {}
    SerializableMat(const cv::Mat &mat):cv::Mat(mat){}

    virtual ~SerializableMat(){}

private:

    friend class boost::serialization::access;

    template<class Archive>
    inline void serialize(Archive & ar, const unsigned int version)
    {
        //      std::cout << "Check my Smat out!: " << cols << ", " << rows << ", " << elemSize() << ", " << type() <<std::endl;

        int arCols = cols;
        int arRows = rows;
        size_t arElemSize = elemSize();
        size_t elemType = type();

        ar & arCols;
        ar & arRows;
        ar & arElemSize;
        ar & elemType;

        create(arRows, arCols, elemType);

        size_t dataSize = cols * rows * elemSize();

        //      std::cout << "about to memcpy: " << cols << ", " << rows << ", " << elemSize() << ", " << elemType <<std::endl;

        ar & boost::serialization::make_array(ptr(), dataSize);
    }
};
```

תהליך ההעברת התמונה בין המודולים לקח כ 200ms, תהליך הסריאליזציה לקח כ 2sec, כך גם הכיווץ והחילוץ. סך כל זמן התהליך לקח כ 8.5sec עבור תמונה.

### ניסיון 3

לבסוף החלטנו לאחד את המודולים. כלומר, בין כל תמונה שהמצלמה מצלמת, מתבצע עיבוד לתמונה האחרונה ופרסום של מספר העצמים שזוהו בתמונה.

```
while(true)
{
    //get camera image
    m_camera.grab();
    m_camera.retrieve(image);

    //cv::imshow("0", image);
    //cv::waitKey(500);

    //convert input image to HSV
    cv::cvtColor(image, hsvImage, cv::COLOR_BGR2HSV);

    //Threshold the HSV image, keep only the green pixels.
    cv::inRange(hsvImage, cv::Scalar(45, 50, 102), cv::Scalar(75, 255, 255), greenHueMask);

    //cv::imshow("1", greenHueMask);
    //cv::waitKey(500);

    cv::erode(greenHueMask, greenHueMask, cv::Mat::ones(6,6,CV_8U));

    //cv::imshow("2", greenHueMask);
    //cv::waitKey(500);

    if(0 != cv::countNonZero(greenHueMask))
    {
        eFoundAgents = PeerMessage::PeerRecognized;
    }

    // Publish
    PeerMessage msg(eFoundAgents);
    Publish(msg);

    // Reload
    eFoundAgents = PeerMessage::PeerNotRecognized;

    // Let go of the CPU for a short while
    boost::this_thread::sleep_for(boost::chrono::milliseconds(1));
}
```

בדרך זו אנו מצלמים ומעבדים מעל 20 תמונות בשניה.

## מדריך למשתמש

תחילה יש לפתוח חשבון ב [gitlab](#).

לאחר מכן, להתקין על הרסברי את הבאים:

- Boost
- OpenCV – ניתן להתקין מ[כאן](#).
- IS4AEPs – לבעלי חשבון גיטלאב, ניתן להתקין מ[כאן](#).
- CMAKE – למשתמשים שמעולם לא עברו עם cmake, מומלץ לעבור [תרגול](#).
- RaspiCam – ניתן להתקין מ[כאן](#).
- קוד הפרוייקט נמצאן בגיטלאב בקישור [הזה](#).

לבסוף, יש לקמפל ולהריץ באמצעות CMAKE:

```
mkdir build
cd build
cmake ..
make
./pr
```

## סיכום

במסגרת הפרוייקט הכרנו לראשונה את הרסברי פאי והמצלמה. נכנסנו למערכת ההפעלה LINUX, הכרנו סיפריות חדשות כגון OpenCV, Boost, IS4AEPs, Raspicam. למדנו איך לנהל זמנים, לחלק משימות, לראות את המערכת כולה. למדנו לעומק CMAKE ועבדנו איתו לאורך כל הפרוייקט. הכרנו את הבסיס של עיבוד תמונה.

הגענו להישג מלא של המטרה העיקרית – צילום ועיבוד תמונה באיכות גבוהה ובקצב מירבי.