בס"ד

המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Efraim, the Autonomous Farmer

Yonatan Gershon and Annael Abehssera
Supervised by Roman Rabinovich

המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

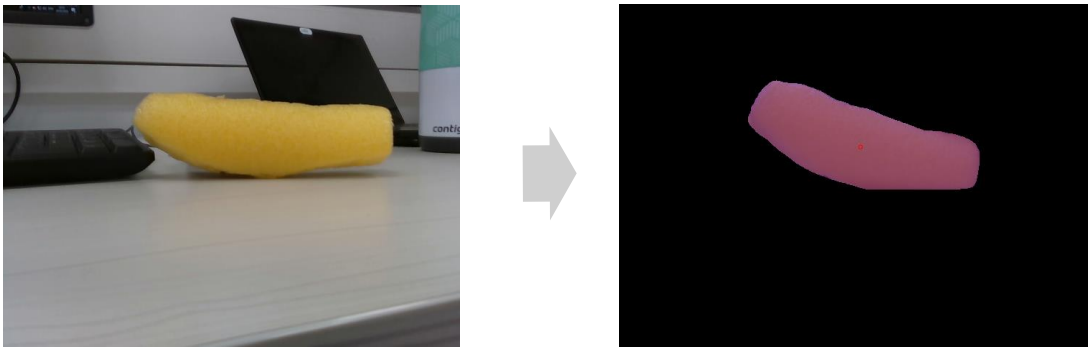
Abstract:

Efraim is a 3D printed robotic arm that uses an image processing real-time segmentation algorithm named Yolact[1] , an Intel RealSense 3D camera and an Arduino board to autonomously identify & locate an object and pick it using its robotic arm.

1) IMAGE PROCESSING

a) ISOLATE THE OBJECT IN THE IMAGE - YOLACT

We started by reading articles about YOLO and Yolact to understand better the domain of image processing and chose to use Yolact together with COCO model for our project. We decided to focus on the class of bananas because they can't be located using bounding boxes but rather require a segmentation image processing algorithm. Then, we integrated Yolact with the RealSense camera to receive 3D coordinates for the segmented image and isolated the fruit from its surroundings.



b) LOCATE THE OBJECT IN THE SPACE ACCURATELY

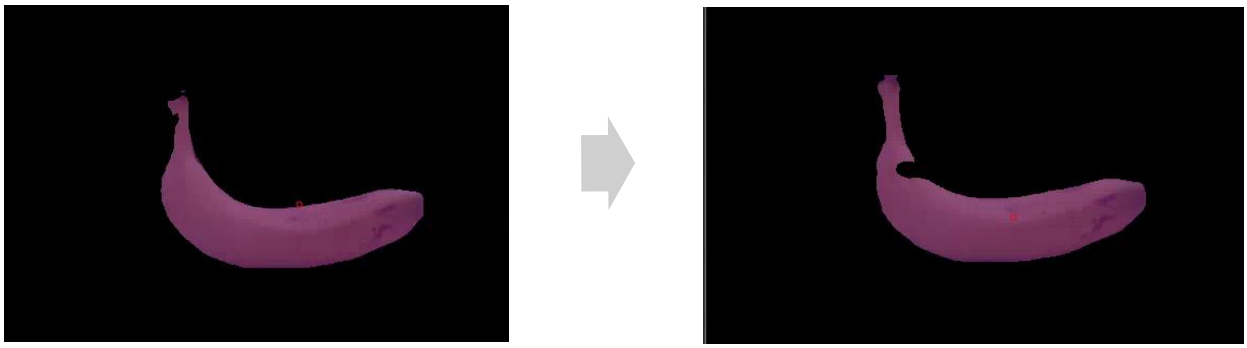
To pick the banana we need to find the best point that would allow us to grip it. We chose its center of mass and calculated it using the 3D coordinates from the RealSense camera point cloud of each pixel in the segmentation mask. That is when we were faced with our first challenges:

- Segmentation wasn't accurate or stable. The segmentation would sometimes focus on the fruit and abruptly include the surroundings of the fruit. This would cause the COM (center of mass) to be extremely unstable and in practice wasn't able to locate the fruit with certainty.
- Measuring the distance from the fruit and comparing with Efraim's distance in practice we found large inaccuracies which we believed were caused by the segmentation crudeness.

[1] Developed in Cornell University by Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee
https://arxiv.org/abs/1904.02689

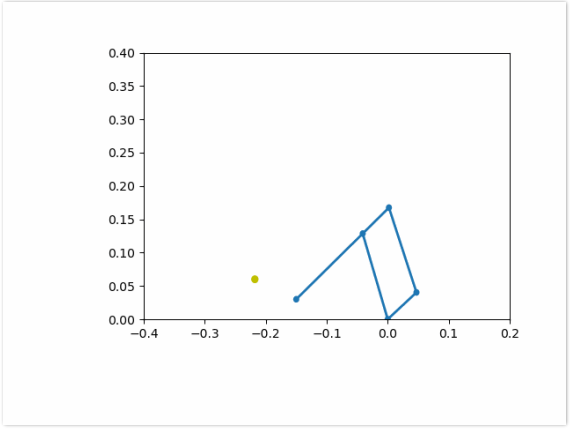
To solve these problems, we proceeded in 2 steps:

1. We used heuristics to remove from the segmentation any pixels that were 20cm further than the closest point of the banana. This removed any pixels of the far background from unstabling the COM calculation.
2. After implementing the heuristics, the COM became more stable but wasn't yet accurate enough. We viewed the mask matrix and noticed that many pixels received '0' in the depth dimension due to ongoing synchronization between the RGB camera and the 'cloud points' detectors of the RealSense camera. Removing all the pixels with no depth dimension resulted with an accurate distance & stable COM.



2) AUTONOMOUS MOVEMENT & SIMULATION OF THE ROBOTIC ARM

After we achieved stable COM calculations we decided to turn Efraim into an autonomous robot. Efraim decides he knows the correct COM of the banana if he counts 5 consecutive frames with the COM within a margin of 1cm. The robotic arm movement is calculated using inverse kinematics from the arm location to the COM of the banana.



המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

3) REALSENSE COORDINATES TO ROBOTIC ARM MOVEMENT

a) CONVERT COLOR PIXEL TO REAL LIFE COORDINATES

At this point the coronavirus kicked in and the lab was closed. We simulated the COM with double-clicking on the desired point on the camera stream. We needed to get real-life coordinates of the pixel from the Real Sense camera and move the robotic arm to that location.

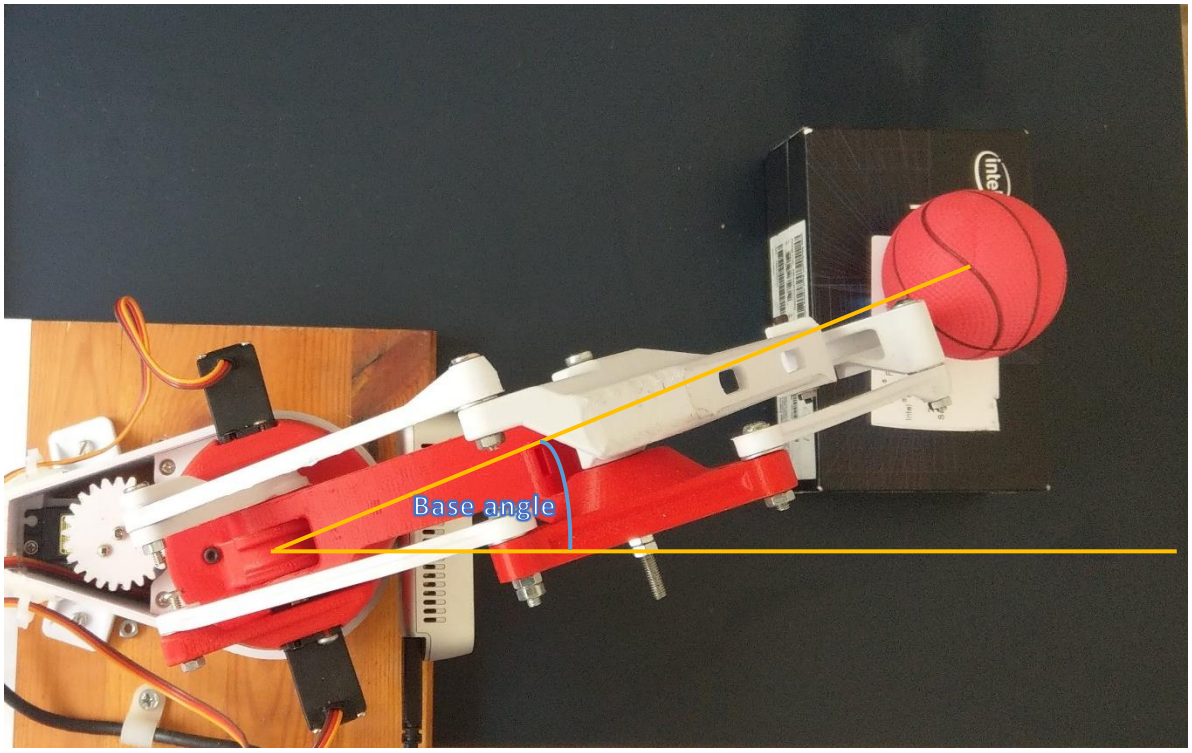
1. Mouse pointer selects pixel in colour frame. The reason we pick the pixel from the colour frame is because the Image Processing is done on this frame.
2. Convert pixel from colour frame to depth frame using function: rs2_project_color_pixel_to_depth_pixel
3. Convert depth pixel to real life coordinates using function: rs2_deproject_pixel_to_point

b) CALCULATE ARM ANGLES WITH COORDINATES

The base angle calculation:

alpha = (180 / math.pi) * math.atan2(z, x)

base_angle = 180 – alpha



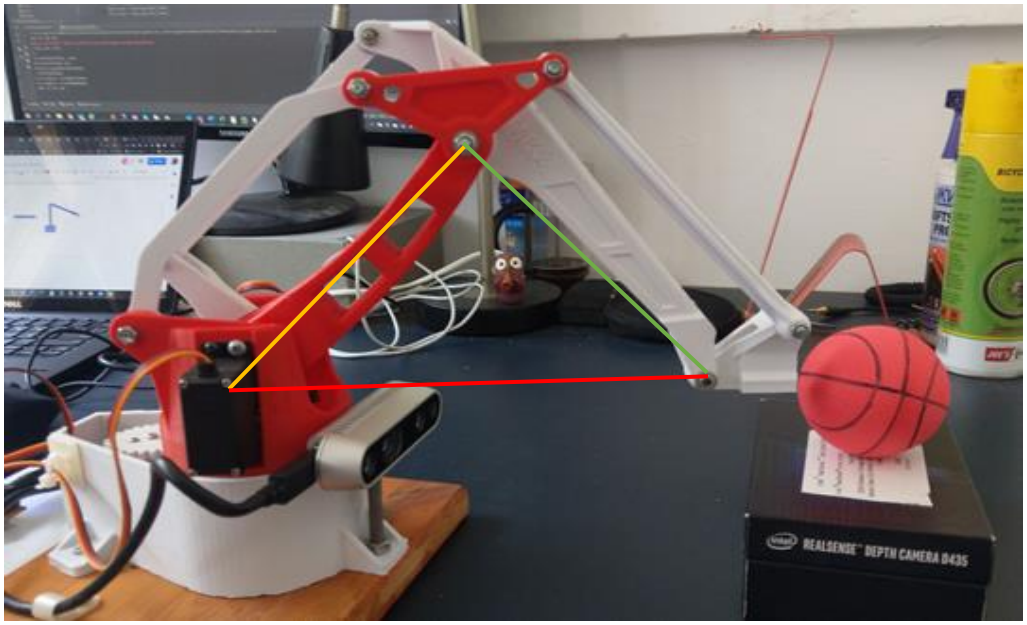
המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

We know that part A measures 13.5cm and part B 14.8cm. The vertex C of the triangle can be easily calculated thanks to the real-life coordinates of the yellow point. Thus, we have a triangle which all vertices lengths are known. With the law of cosines we calculated the servo angles needed to be rotated in order to reach the desired point.

The arm angle calculation:

1. Servo A angle calculation (using Pythagoras and the law of cosines):
   distance_to_goal = sqrt(x * x + y * y)
   angle_a = arccos((148 ** 2 - distance_to_goal** 2 - 135 ** 2) / (-2.0 * 135 * distance_to_goal)))

2. Servo B angle calculation (using Pythagoras and the law of cosines):
   angle_b = arccos((distance_to_goal ** 2 - 148 ** 2 - 135 ** 2) / (-2.0 * 135 * 148)))



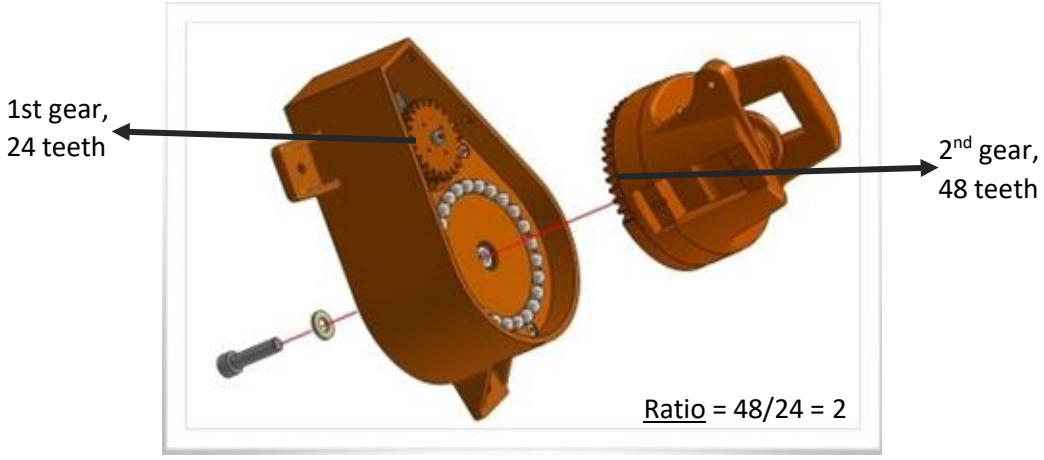
4) ROBOTIC ARM CALIBRATION TO REALSENSE CAMERA

a) CALIBRATE ARM ANGLES TO SERVO ANGLES

The base angle calibration:

- Adjust the coordinates from the camera point of view to the base': x -= 0.02, y += 0.04 , z += 0.06
- The ratio between the number of teeth of the 2 gears in Efraim's base is used as a factor for the angle: factor = 2.
- Then, according to the side it has to rotate to, we calculate the angle:

```
if (base_angle < 90):
        angle_fix = factor * (90 - base_angle)
        base_angle = 90 - angle_fix
 else:
        angle_fix = factor * (base_angle - 90)
        base_angle = 90 + angle_fix
```



The arm angle calibration:

1. **Servo A angle calibration:**

angle_from_horizontal = arcsin(y / distance_to_goal))

servo_angle_a = 90 - (angle_a + angle_from_horizontal)

servo_a_angle = servo_angle_a + 100

2. **Servo B angle calibration:**

servo_angle_b = angle_b - servo_angle_a + angle_from_horizontal

servo_b_angle = servo_angle_b + 42

But, again, the two parts of the arm were not moving as desired. We made some calibration in order to get the desired results. We placed the 2 parts of the arm to form a 90degrees angle and from this position we succeeded to understand how to control the servos movements.

b) LIMITATIONS

Arm limitation

we checked the range of angle each servo can move from physical aspect:

```
# angle range( 25 - 165, 55 - 140, 55 - 145)
```

RealSense Camera limitation

The RealSense camera can only calculate the depth of objects from 15cm away and the arm can only reach as far as 19 cm which leaves Efraim with only 4 cm of operational area it can reach

5) INTEGRATION FROM YOLACT TO EFRAIM

When the lab opened, we merged the code of the robotic arm movement to the first part that uses Yolact to calculate the COM of the banana. And to perfect Efraim, we added a gripper at the end of the arm and made a few adjustments in the code to get the right calibration. With the gripper Efraim's arm movement stays the same. Its gripper moves the following way: starts with an open position and after the gripper surrounds the fruit, it waits 1 second and then closes on the banana.

6) PERFORMANCE AND PRECISION

- Efraim is very fast: It takes just one second from the moment that the banana is recognized to its gripping.
- Locating the object: the COM (x, y, z) parameters given from the RealSense camera are highly accurate in our measurements
- Picking the object: the movement of the servo engines is a little crude and can result in slight inaccuracies. So in practice, Efraim gets to the desired point with a precision of $\pm 1cm$.

7) MODULABILITY

The project consists of two main parts:

I. Location the COM of an abject using Yolact & RealSense resulting with an x,y,z coordinates.
II. Moving the arm to the x,y,z coordinates using inverse kinematics.

This make the project very mouldable:

1. The system can be adapted to any robot/robotic arm by changing the inverse kinematics function. Or in the case of a two part arm, just adjust the lengths of the robotic arm in the code.
2. The object that Efraim identifies can easily be modified by choosing the class(es) we want Efraim to focus on and recognize. This is done by choosing the index of the desired class in the file *layers/functions/detections.py*, in the function *detect*. In fact, the choice of the class is done by multiplying the scores of the classes we want to supress by 0. In other words, we put a value 0 in the array of scores in all the indices that correspond to a class we don't want Efraim to recognize. We can know the indices of each class thanks to the details in the file *data/config.py* ➔ Efraim can be adapted to all the fruits that COCO recognizes.
3. If we want Efraim to recognize an object which class isn't included in COCO, we can train the model on this new class according to the instructions of Daniel Bolya in the github of Yolact[2].

8) FUTURE WORK

- Move the image processing from a computer GPU to a board so Efraim can move independently in the room and not be connected to a desktop computer.
- Efraim has trouble picking several time in a row due to the frames buffer filling up. This can be solved in various ways.

[2] https://github.com/dbolya/yolact

9) ACKNOWLEDGEMENT

We want to thank our supervisor Roman Rabinovich and Yaron Honen for all their help and advice along the way.

9) REFERENCES

https://arxiv.org/abs/1912.06218

https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

https://github.com/dbolya/yolact

https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0#point-cloud

http://www.eezyrobots.it/eba_mk2.html

https://github.com/justbuchanan/eezybotarm-mk2-software