Project Efraim

Robotic arm handling fruit using RealSense, Yolact & Arduino

Students : Yonatan Gershon, Annael Abehssera

Supervisors: Yaron Honen, Roman Rabinovich, Andrey Zhitnikov

Efraim, the agricultor.

Project plan

- **Goal**: Build a system that can locate a fruit in 3D using an Intel RealSense D435 and handle it with the help of a robotic arm.
- **Steps**:

1) Image processing
   a) Articles about the domain
   b) Defining image processing model
   c) Test the accuracy of the image processing model
2) Guide the robotic arm to the fruit using the coordinates extracted from the Intel RealSense camera
   a) Locate the fruit with the camera
   b) Simulation of the robotic arm
3) RealSense coordinates to Robotic arm movement
   a) Convert color pixel to coordinates
   b) Calculate arm angles with coordinates
4) Robotic arm calibration to the RealSense Camera
   a) Calibrate arm to servos angles
   b) Configurate Limitations

1) Image processing
a) Articles about the domain

- We read articles regarding YOLO and Yolact.
- Understood generally how deep learning works and how a model is trained.
- Decided to use COCO model for our image processing

⇒ Goal: recognize fruit with YOLO image segmentation

⇒ Conclusion : integrate Yolact model & COCO trained model with the RealSense camera in order to locate the fruit in 3D coordinates

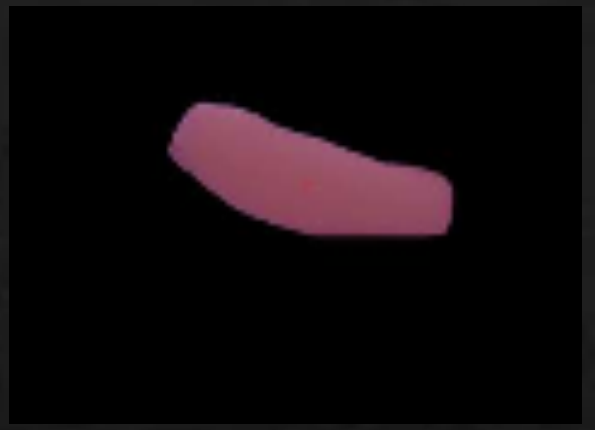
1) Image processing
b) Defining image processing model

- Used Yolact to segment specific items (bananas in our case)
- Integrated Yolact github code with the RealSense camera to receive 3D coordinates for segmented image
- Isolate the fruit from its surroundings.
- Calculate the best location to grip the fruit.

→

1) Image processing

c) Test the accuracy of the image processing model



Main challenge found using Yolact & RealSense

- 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 numbers we found large inaccuracies which we believed were caused by the segmentation crudeness.

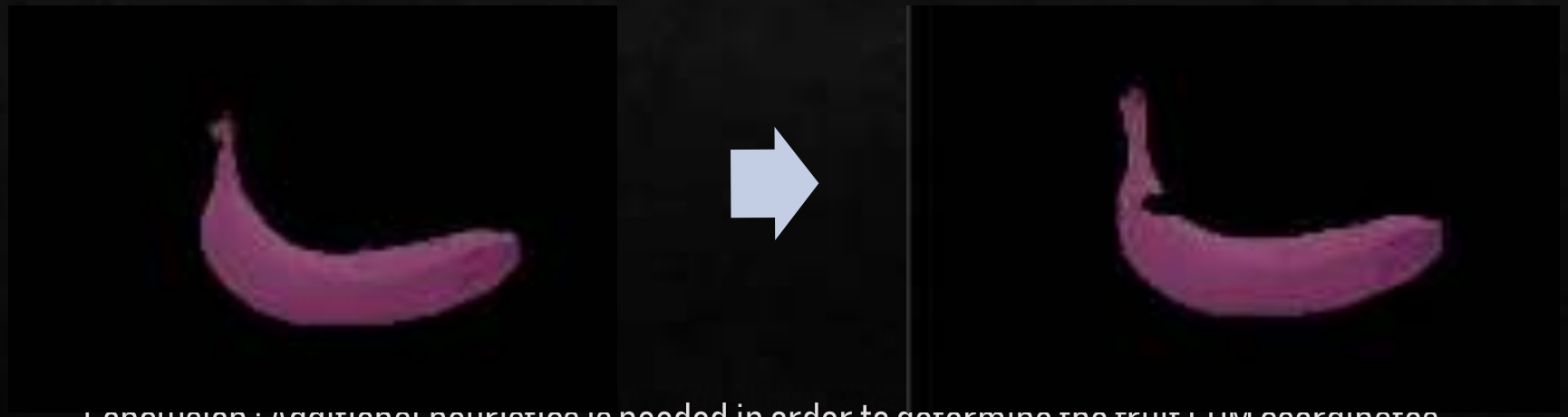
⇒ Conclusion :
A solution must be found to locate the fruit with high certainty and decide when the location calculated is valid.

2) Robotic arm movement

a) Locate the fruit with the camera

Improving segmentation:

- We used heuristics to remove from the segmentation any pixels that were 20cm further than the closest point of the banana.
- After the heuristics the COM became more stable but wasn't yet accurate enough. We viewed the mask matrix and noticed that many pixels would receive a '0' in the depth dimension due to synchronization between the RGB camera and the 'cloud points'. Removing all the pixels with no depth dimension resulted with an accurate distance & stable COM.

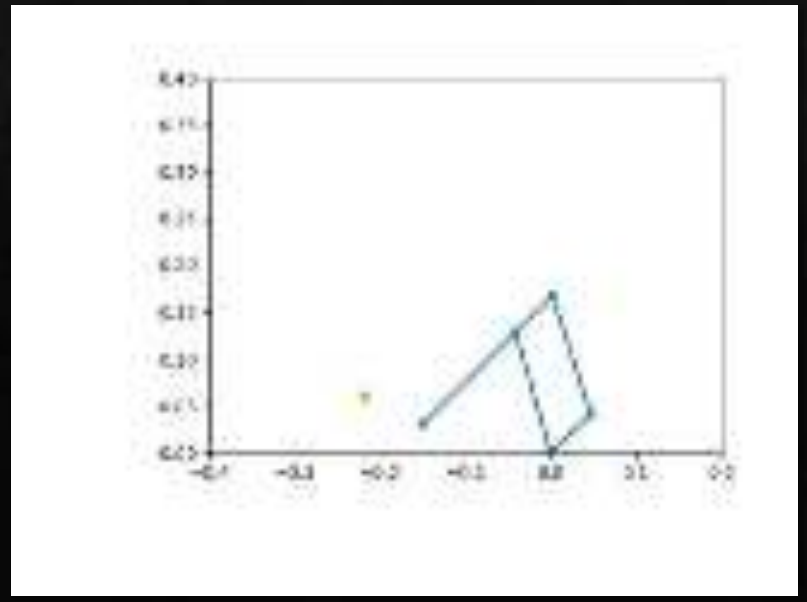


⇒ Conclusion : Additional heuristics is needed in order to determine the fruit COM coordinates.

2) Robotic arm movement
b) Simulation of the robotic arm

Robotic arm autonomous movement:

- The robot decides he knows the correct COM of the banana if he counts 5 consecutive frames with the COM within the margin of 2cm.
- Robotic arm movement is calculated using inverse kinematics from the default arm location to the COM of the banana.



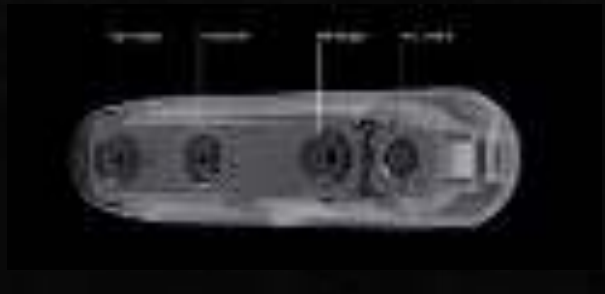
3) RealSense coordinates to Robotic arm movement

a) Convert color pixel to coordinates

Goal: simulate the fruit COM by clicking with the cursor on a pixel

of the RealSense RGB camera and retrieving the 3D coordinates .

- Mouse pointer selects pixel in color frame. The reason we pick the pixel from the color frame is because the Image Processing is done on this frame.
- Convert pixel from color frame to depth frame using RealSense library function: *rs2_project_color_pixel_to_depth_pixel*
- Convert depth pixel to real life coordinates using RealSense library function: *rs2_deproject_pixel_to_point*

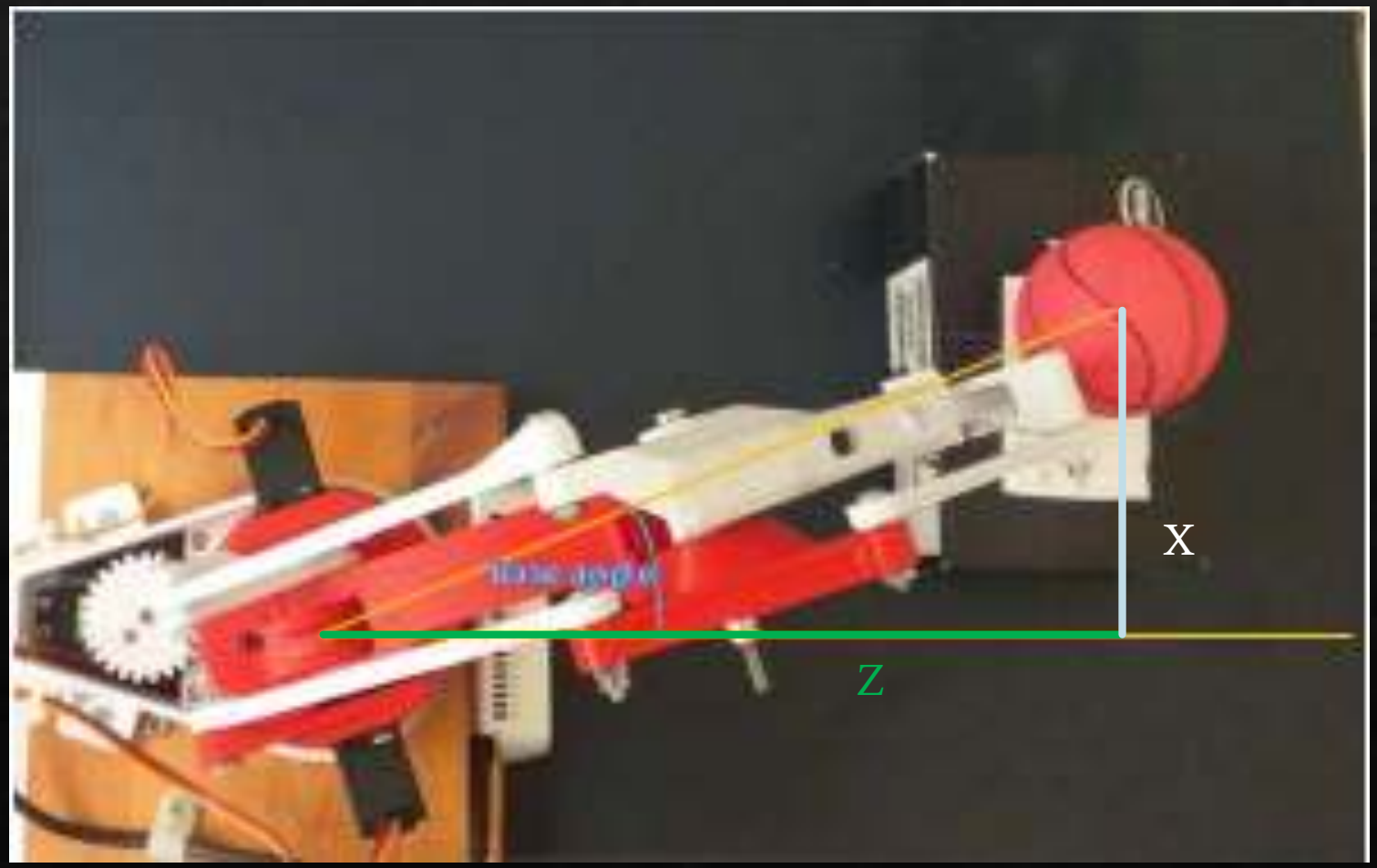


3) RealSense coordinates to Robotic arm movement

b) Calculate arm angles with coordinates

The base angle calculation:

$$base\_angle = \frac{180}{\pi} * \arctan(z / x)$$



3) RealSense coordinates to Robotic arm movement

b) Calculate arm angles with coordinates

The arm angle calculation:

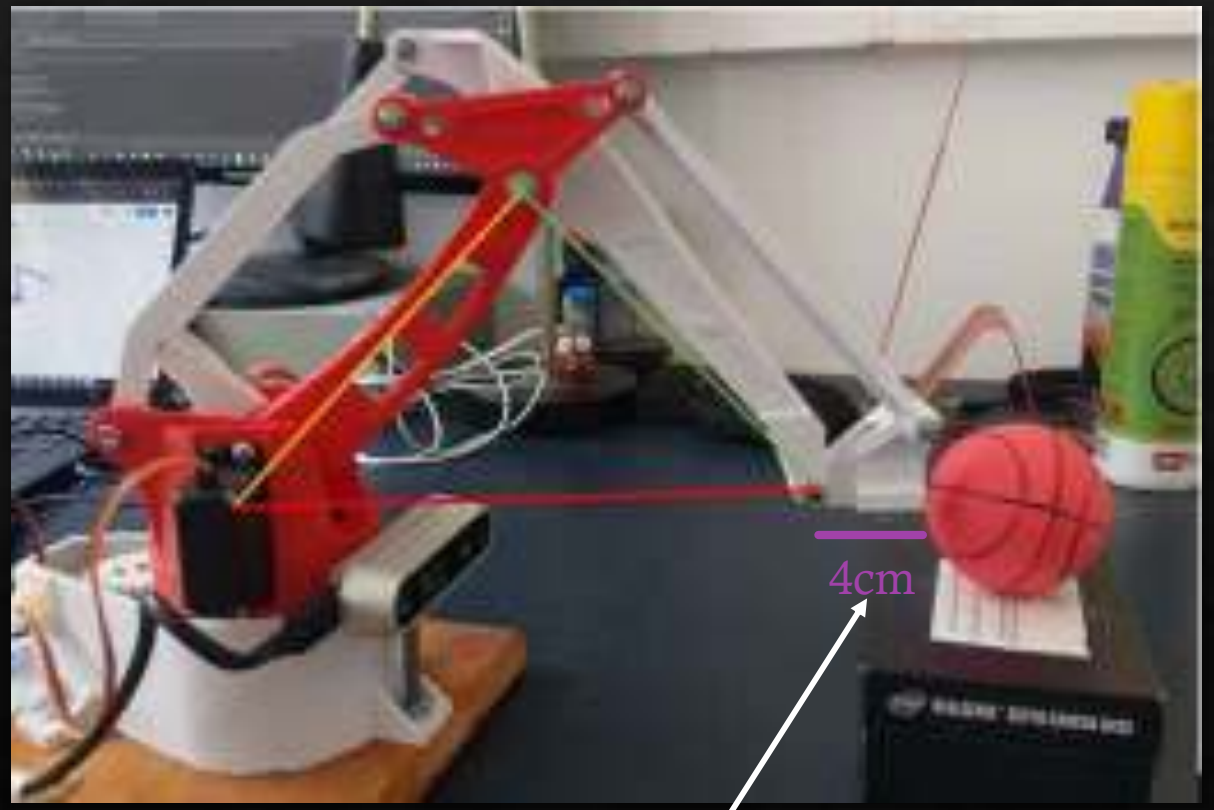
- Servo A angle calculation:

$$\text{distance\_to\_goal} = \sqrt{x^2 + y^2}$$

$$\text{angle\_a} = \arccos\left(\frac{148^2 - \text{distance\_to\_goal}^2 - 135^2}{-2 * 135 * \text{distance\_to\_goal}}\right)$$

- Servo B angle calculation:

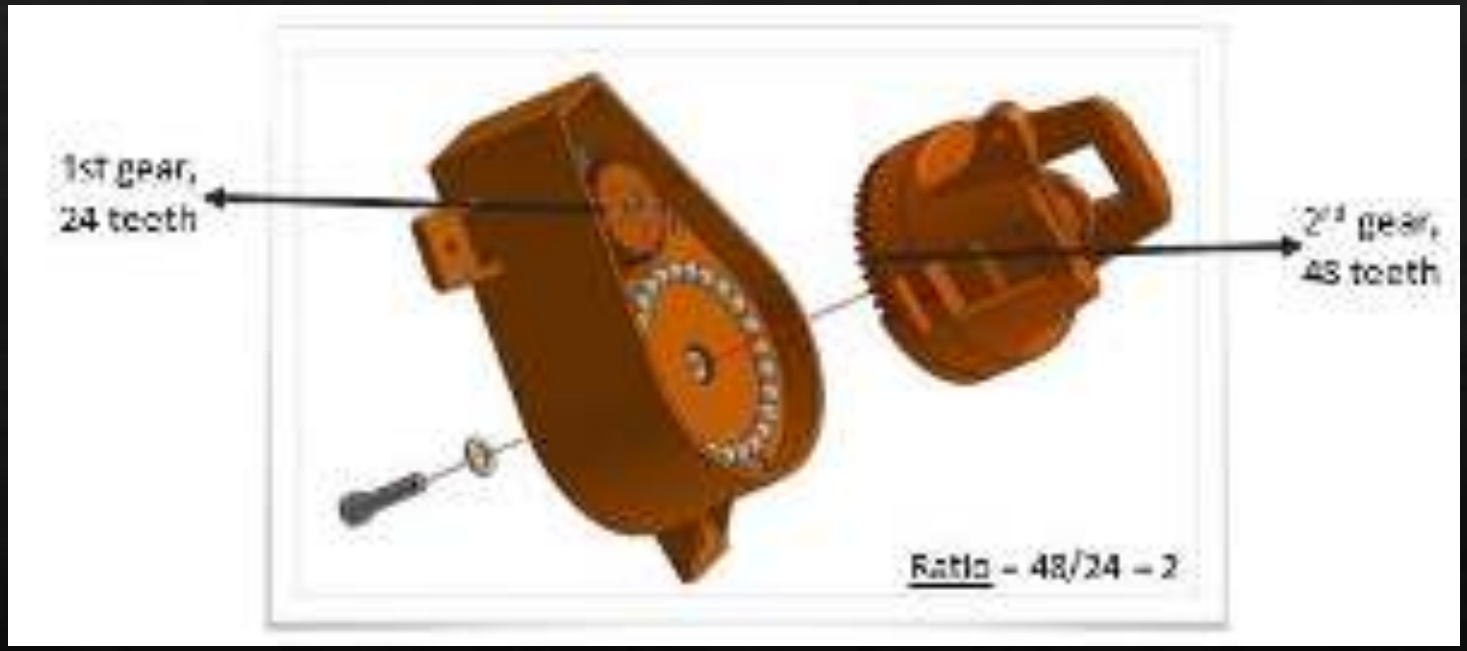
$$\text{angle\_b} = \arccos\left(\frac{\text{distance\_to\_goal}^2 - 148^2 - 135^2}{-2 * 135 * 148}\right)$$



Removing 4cm from z since gripper is always parallel to x,z plane

4) Robotic arm calibration to the RealSense Camera

a) Calibrate arm to servos angles



<u>The base angle calibration:</u>

- Adjust the (0,0,0) coordinates from the RealSense camera to the robotic arm base: x -= 0.02, y += 0.04 , z += 0.06
- Calculate the Base angle rotation using the ratio between the number of teeth of the 2 gears in Efraim's base: Angle factor = 2

4) Robotic arm calibration to the RealSense Camera

a) Calibrate arm to servos angles

The servo angle calibration was done at all 90 degrees

4) Robotic arm calibration to the RealSense Camera

b) Limitations

Arm movement limitation

base angle = 25 – 165 | red arm = 55 – 140 | white arm = 55 - 145

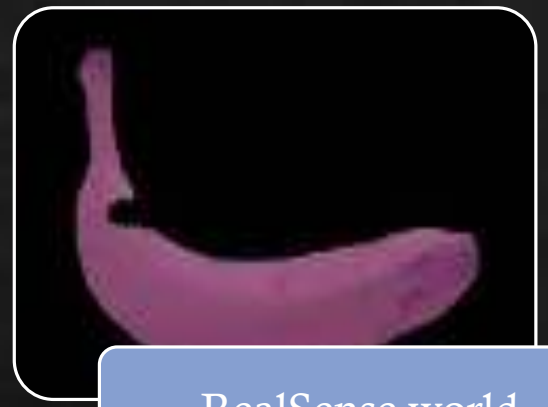
RealSense camera limitation

The RealSense camera can only calculate the depth of objects from 15cm away and the arm can only reach as far as 19 cm ➔ **Efraim has only 4 cm of operational area**

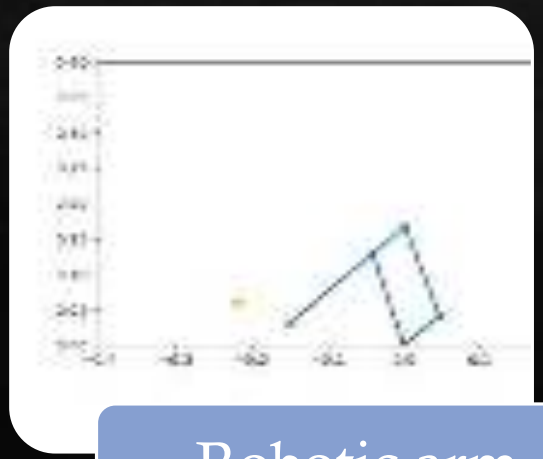
Efraim1 Overview

RealSense RGB Image

Yolact



RealSense world coordinates + heuristics



Robotic arm angles calculation

Arduino communication

Servo calibration

References

- 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