Segmentation for Robotic Arm

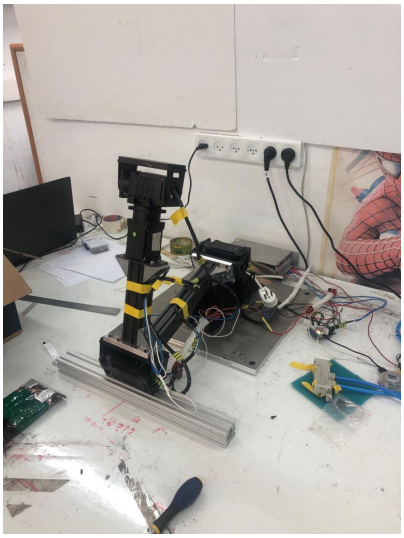
Ethan Baron and Snir Green

Abstract

The Robotics Excellence Program in mechanical engineering department are trying to build a robotic arm that picks tin cans (as the picture on the right) from a clutter in a box and organize it for shipment.

The robotic arm will use:

1. Vacuum end point to pick up cans from above (from the "head" of the tin can).
2. Linear end point (clamp like) to pick cans that are laying or in an angle.

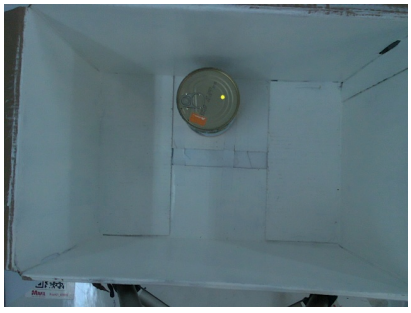


Above the box there is a static RGBD LIDAR camera.

Our project was creating the "vision" part of the robot, meaning each time the robot wants to take the next can from the clutter – he sends a request to our program that is connected to the static camera.

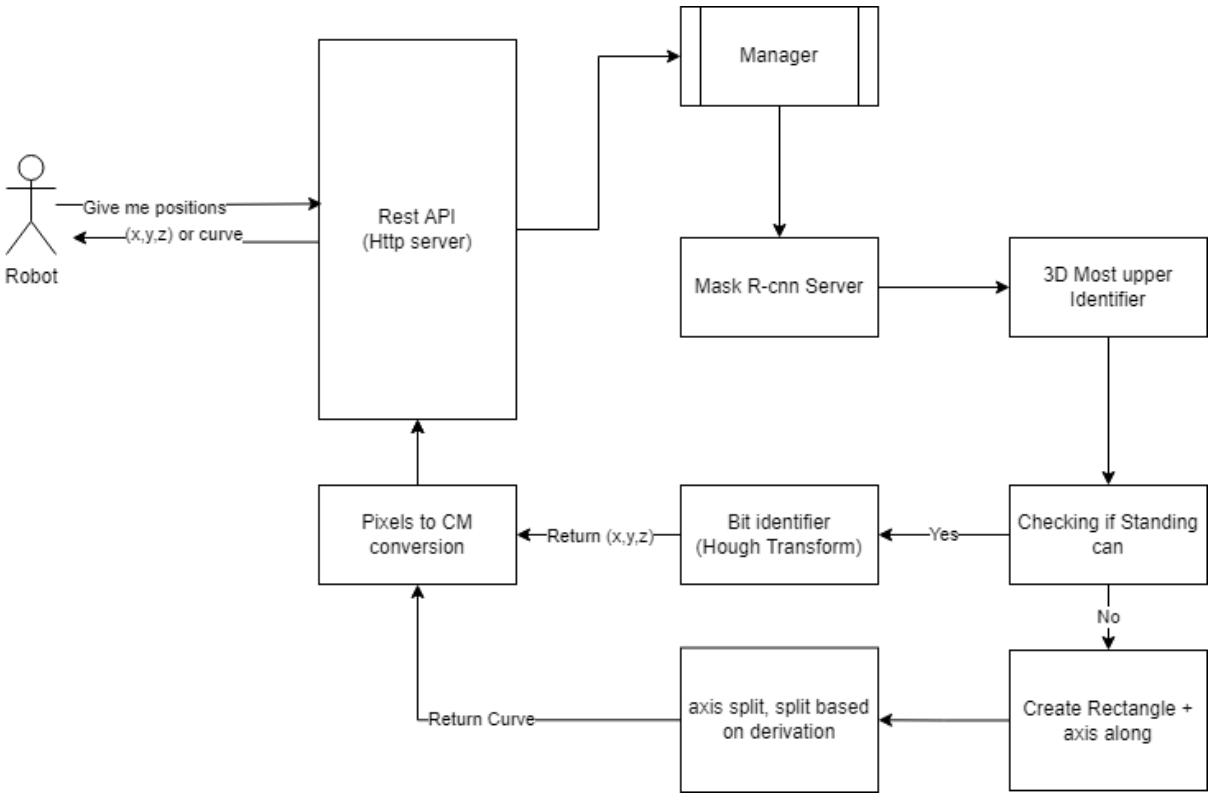
Our program thus accepts a request from the robot for the next tin can, takes a picture using the RBGD camera, process it with algorithm that will be explained and:

1. Returns the location of 1 point on the top of the cans if it is standing.
2. Returns 2 points which constitutes a straight line on the middle most upper part on the can (Examples below).



Algorithm

We'll walk through the whole algorithm, shown in the diagram below:



In each part we'll make the part we are talking about in blue.

Manager

Firstly, the robot sends an HTTP request to the REST API of the algorithm. We choose to separate the robot and the vision algorithm itself as part of micro services view in mind, while the HTTP could be replaced in later versions for ZeroMQ for example.

When accepting an HTTP request for getting the next can position, the program takes an RBGD picture using the camera, and sends it to the Mask RNN neural network for segmentation (will be explained further below).

Mask RCNN

Mask R-CNN is a CNN based neural network that is the state of the art in image and instance segmentation today. It is based on Fast R-CNN, which is region-based CNN. It gives bounding boxes as well.

We took the TensorFlow version which is quite deprecated and hard to use (only works with old versions of TensorFlow and other libraries), and the checkpoint which is trained on the COCO dataset.

The COCO dataset contains 88 classes. Cans are not one of them, but we hoped it might recognize the cans as bottles or other objects and we'll ignore the class itself.



(Some examples of how it performed)

As you can see, it identified some of the cans quite well, and others poorly. Important to remind here – we only need to recognize one most upper can correctly – therefore even if the algorithm did not recognize all the cans correctly, one is enough.

Important to note that in simpler scenarios it worked quite well.

Because of the poor results, we had to train the Mask R-CNN with more data.

After a survey in Fiverr, we choose some company in Kenya to help us. We took 1200 pictures asked them to create annotations of the pictures:



And then trained the COCO checkpoint with the new dataset.

Some results:



In an environment containing only cans, the network identifies cans correctly in >90% of the cases. It does make a lot of false positives with other objects, but as our environment contains cans only, it was enough for us.


It does make errors (for example – here we think it might make error because we didn't train the NN on very large cans or that the configuration of mask r-cnn defines a maximum size for segmentation in the picture).

3-D Most Upper Identifier

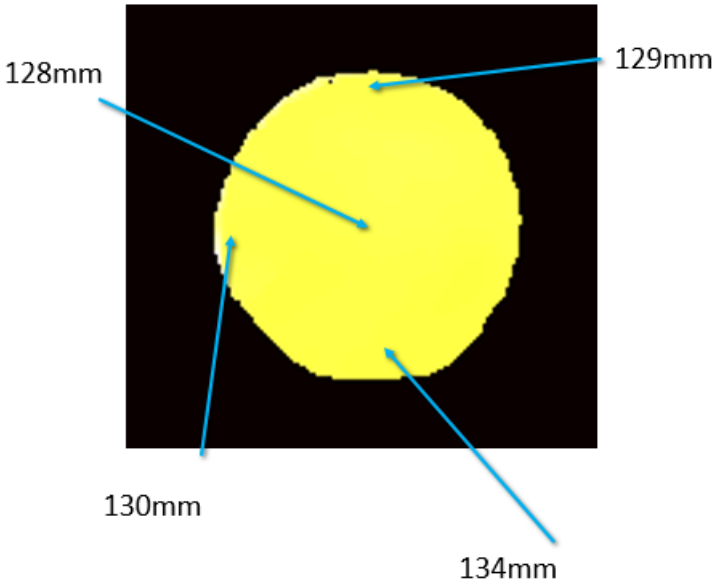
Remember – we are after segmenting the picture and getting the masks for each can.

Now, we firstly smooth each of the mask with Gaussian filter $\sigma = 1$. Afterwards,

we want to get the most upper one. For each mask, we find the most upper pixel inside the mask. Then, we take the mask with the highest pixel of all those and mark it as the most upper one. It does give sometimes the second most upper due to camera noise (we'll be explained in the next part), but mostly works quite well and can be improved easily with using percentiles in the Z dimension.

Depth Camera Noise

We found out quite late in the project that the depth camera is quite noisy. As can be seen in the diagram on the right, while looking at a standing can from above, there should be a difference in heights of less than 1mm. In the actual data received from the camera, for a can standing in the middle there might be a difference of 5mm, and for a can standing in the far end of the camera, there can a difference of up to 15mm.



This is after adjusting the camera and validating it is perpendicular to the floor.

We needed to find somehow a way to tell whether a can is standing or laying (because of the different ways to handle it). Thus we tried a couple of methods to differentiate between laying and standing can:

1. Smoothing the Z matrix with Gaussian Filter, Uniform Filter, and Median Filter with different kernel sizes.
2. Afterwards, using different algebraic measures as percentiles in the matrix – check whether the difference between different percentiles is always lower in standing vs laying cans

3. Using the 1st and 2nd derivatives of the Z matrix. The 1st derivative of standing can should be 0, as the 2nd one. They both in a laying can should be very different than 0.
4. Smoothing the 1st and 2nd derivatives and using different algebraic measures as comparing averages, summing and difference in percentiles with or without removing outliers.

Unfortunately, we failed in finding a golden rule that works for all cans and scenarios.

Checking If can is Standing

By empirical means of trial and error, we found that for $\epsilon = 6 * 10^{-3} m = mm$, the difference between the 50 percentile and the 10 percentiles is smaller than that ϵ.

Thus, if the condition:

$$d_{50} - d_{10} \leq 6mm$$

Holds, it is classified as standing can, otherwise not. Important to note that in some edge cases where the cans are on the far right/left the condition does not hold.

Bits Identifier

Motivation

When a can is standing, we want to return a point (x, y, z) where the robotic arm can pick it up using a vacuum gripper.

If the can has a smooth surface, we can grip it from the center of the can:

But, if there is a can openner, which we will call “bit”, the vacuum gripper cannot lift the can from the center, since the bit is interferes and letting the air to get out from the side.

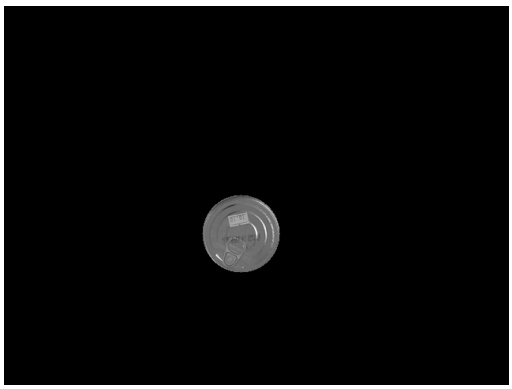
Therefore, we need to return a farther point. For example:

For that, we need to first identify the bit and then find the mid point between the center and the point on the circle (The geometric calculation will be detailed later)

Finding the bit

First, we take the given masked image.

Then, we isolate and process the image to grayscale image.



Then, we find the bits.

Naïve solution

We have tried to use circle Hough Transform (CHT).

CHT is a basic feature technique used in digital image processing for circles detection in a 2D image.

We can see the problem in the following image:

We get a lot of false positives from the image.

The Next Step – Hyperparameters Tweaking

Hough Transform algorithm uses edge detection (with canny). When we use the CHT, we can tweak the hyperparameters of the canny algorithm.

We found the best hyperparameters.

But, even then, we got some problematic cases that recognize few circles on the top of the can.

Final Step – Choosing Correct Circle

Since after tweaking CHT hyperparameters there are still some false positives, we decided to run CHT multiple times with different hyperparameters and use heuristics to choose the correct bit circle from the given ones.

The heuristics includes the most reoccurring circle, the circle closest to the mean circle, distance from center (which should be $\sim\frac{3}{4}r$, when r is the radius of the can) and number of close circles.

Calculating the "Sweet Spot"

After we found the circle of the bit, we can calculate the "sweet spot" – the point where the vacuum gripper can lift the can from.

To find the "sweet spot", we used simple geometric calculations:

Some definitions:

$$bit\_center := (x_b, y_b)$$
$$can\_center := (x_c, y_c)$$
$$can\_radius := r_c$$

We will find the line that goes through the of the can and the bit:

$$m = \frac{y_b - y_c}{x_b - x_c}$$
$$y = m \cdot x + m \cdot x_b + y_b$$
$$c := mx_b + y_b \Rightarrow y = mx + c$$

Now we will find the intersections with the can circle:

The circle equation of the can is:

$$(x - x_c)^2 + (y - y_c)^2 = r_c^2$$
$$y = m \cdot x + c$$
$$\Rightarrow (1 + m^2) \cdot x^2 + (2my_c + 2mc - 2x_c) \cdot x + x_c^2 + y_c^2 + c^2 - 2y_c c - r^2 = 0$$

And we can find x:

$$x := x_1, x_2$$
$$\Rightarrow y = y_1, y_2$$

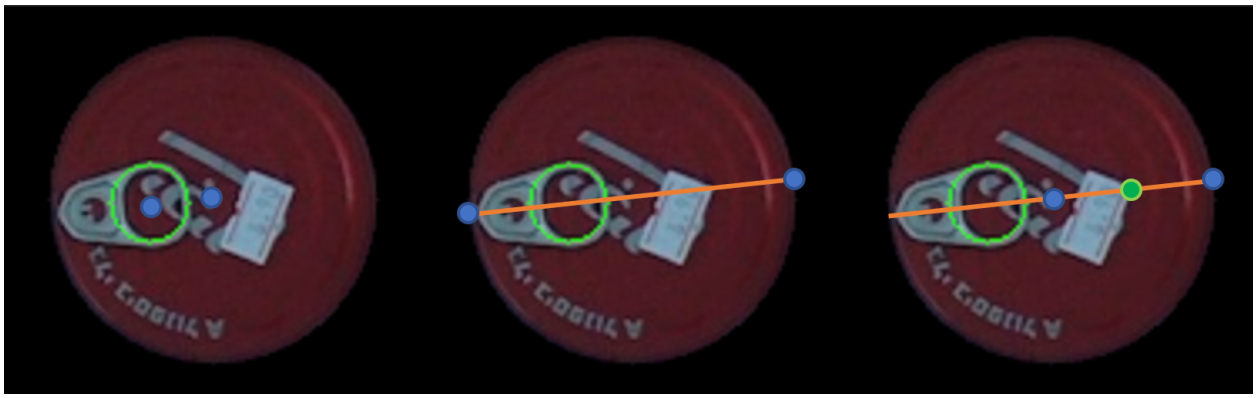
Now, we find the farther point from the bit:

$$farther\_point = \max_{|(x_i, y_i)|_2} \{(x_1, y_1), (x_2, y_2)\}$$
$$farther\_point := (x_f, y_f)$$

Then, finally, we can find the "sweet spot" which is the midpoint between the farther point and the center of the can:

$$sweet\_spot = \left(\frac{x_f + x_c}{2}, \frac{y_f + y_c}{2}\right)$$

Here's an example:



After we found the "sweet spot" we can return the (x, y, z) values back to the robotic arm.

Finding Surrounding Rectangle

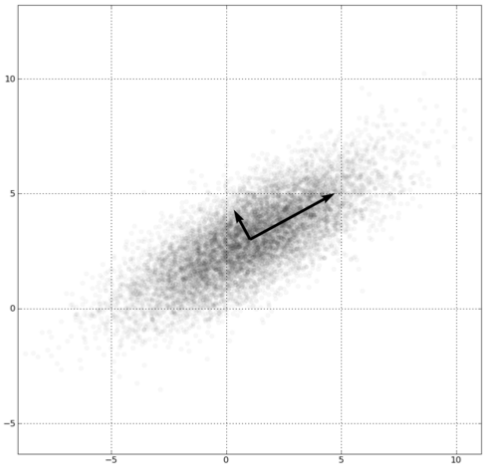
Motivation

When the can is not standing, we want to find the rectangle surrounding the can.

After finding this rectangle, we can cancel some noise of the mask, and more importantly, we can take the midpoints of the rectangle and this will be our curve.

Solution – PCA

PCA is a method to transform data to orthogonal vectors that describes the variance of the data. For example, in the following scatter, the variance of the data can be descried as the vector presented as arrows:



And more specifically, for our needs, we can describe this mask:

As these vectors, which are orthogonal:

After these vectors are found, we can use the new coordinates system to find the final rectangle:

Finding the Curve

Motivation

After Finding that the can is not standing, and finding the surrounding rectangle, we want to find the curve that is returned to the robotic arm.

To do that, we should find two points that describe the curve.

Finding Curve 2D Coordinates

The first step after finding the surrounding rectangle, is to use the mid points of the rectangle.

If the can is laying fully down, theses points describe the curve itself, and they are the points that will be returned. But, for tilted cans, we need to first find the most upper point on the curve between the two midpoints:

After finding the most upper point on the “mid curve”, the final curve should be the curve between the upper point and one of the midpoints on the rectangle.

Choosing curve

To choose the correct curve, let’s consider the next scenario:

Given the two blue curves, we want our curve to be the right curve in this scenario.

If we take the 2nd derivative of the depth camera, the left curve's derivative should be ~ 0, but the right curve's derivative should not.

This is how we can distinct between the curves.

Finally, we return the final curve in a form of two points (x, y, z).

Pix to CM

Motivation

The coordinates for the curve or the "sweet spot", are given in $([pixels, pixels, cm)]$.

But, in the physical world the robot needs the points given in cm.

Therefore, we need to convert $pixels$ to cm.

The Conversion

We found that the ratio between $pixels$ and cm is linear in the distance from "ground" (the bottom of the box).

Empirically, we found that this ratio is:

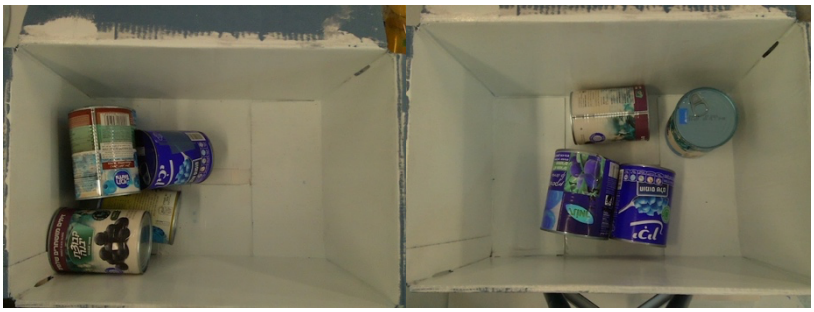
$$ratio = -0.22 \cdot dist\_from\_floor + 23.6$$

Using this equation, we can find the ratio and therefore find the coordinates in cm instead of pixels.

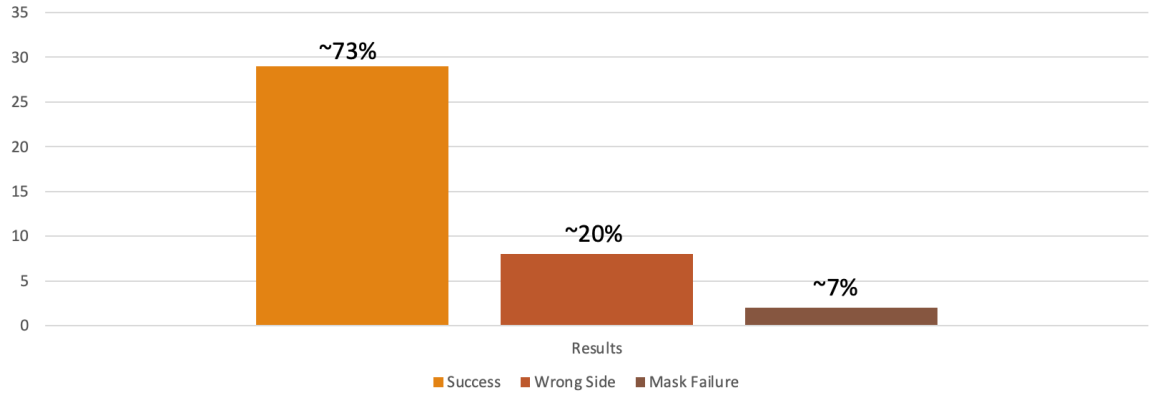
Results

Experiment setup

In the experiment we had a cardboard painted white. The depth camera is located about 60 cm above the ground and there were few different scenarios with: standing cans, laying cans, tilted cans and clutters.



Results



Success

About 73 percent of the scenarios were recognized successfully and returned the correct point/curve.

Wrong Side

About 20 percent of the scenarios found the mask successfully, but return the wrong side in the calculation of the curve

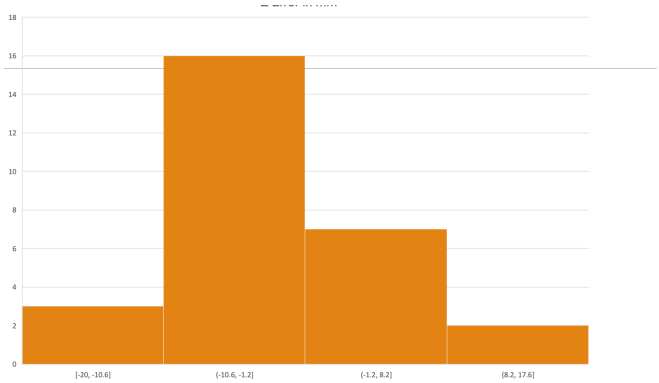
Mask failure

About 7 percent of the scenarios found the wrong mask. This could happen because a can was too big or too close to the camera.

Z Error

We found that because of the inaccuracies of the camera, The z result was inaccurate.

According to our measurements, we can see the inaccuracies in the next plot:



Suggested Improvements

- Less noisy camera / not based on LIDAR
- Working on reducing camera noise with algorithmic approach
- Training the Mask R-CNN on more data and more various data
- Converting the whole program into custom NN with tailor made loss function