Cocktail Maker

Be your own bartender

Final project by Aviram Adiri

aviramad@post.bgu.ac.il

Introduction

Being a cocktail bartender is not as simple as it looks. You need to acquire some special skills, such as understanding what drinks can be combined together, to know the influence of each type of drink, and to know how to fit the best cocktail to the customer. Getting those skills is the result of hard work and years of experience.

In addition, if an amateur wants to feel "like a bartender" and make a cocktail by himself, he needs to do several things:

1) Understand the drink's bottles he has.
2) Check the type of each bottle.
3) Check several combinations of the drinks he has by the internet.
4) Get the recipe of the selected cocktail.

Even though it looks like the "technical" part, every bartender will say that the hard part of this process is to recognize the bottles you have.

The purpose of this project is to make this process easier by recognizing the bottles you have using some Computer Vision skills.

While working on this project I tried two methods for solving this problem:

- Text recognition
- Object recognition

I tried to understand the advantages of each method, and to find the best way to solve this problem.

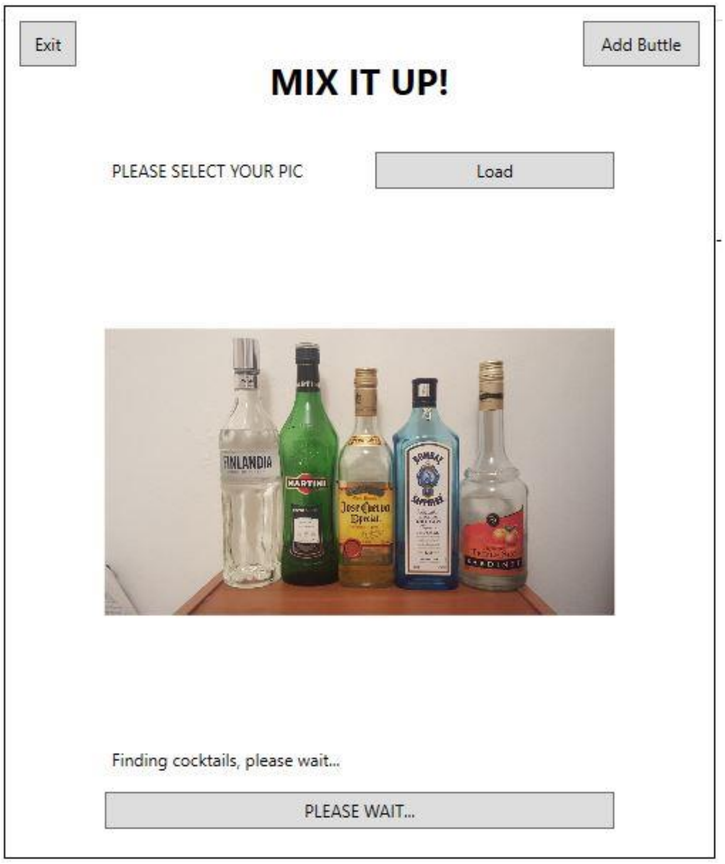
The main goal of this project is to create an application that receives an image of bottles, analyze the objects (bottles) in the image, and show several cocktails that can be made by some combination of those bottles.

Approach and Method

<u>The steps before the object recognition:</u>

In the main window of the application, the user can choose between uploading a new bottle to the bottles database, or upload a bottles image for finding an appropriate cocktail.

After uploading the image and pressing on "GET A COCKTAIL", the application will activate the Matlab program with the image and the DB information.



<u>The object recognition:</u>

In this part I will focus on the image processing issue.

The approach taken in this project was to compare between the bottle's image and the bottles in the database (that can be changed dynamically). For each bottle in the database determine whether the bottle exists or not using SURF – an Invariant Descriptor (will be explained).

The main steps:

1) Detect feature points in both images (the bottle and the user image), and select only the strongest points – we will call those points the interest points

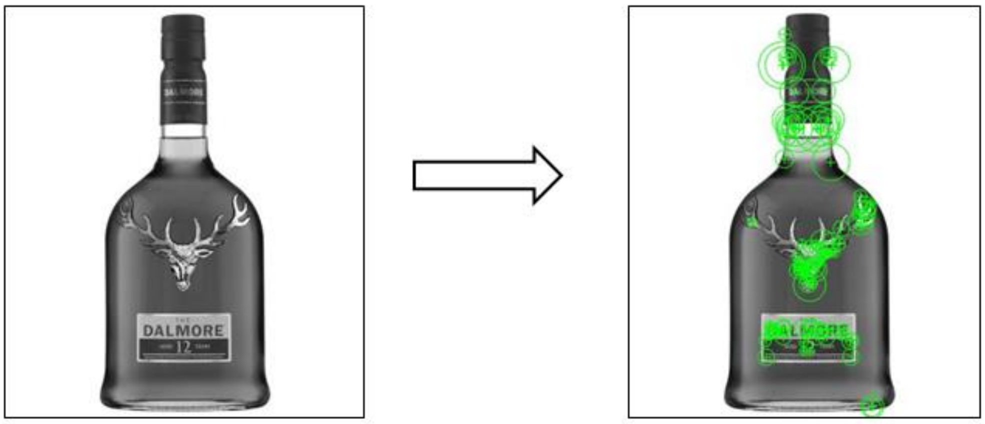
2) Extract feature descriptors at the interest points in both images.
3) Match the features using their descriptors.
4) Remove the outlier points.
5) Check the threshold.

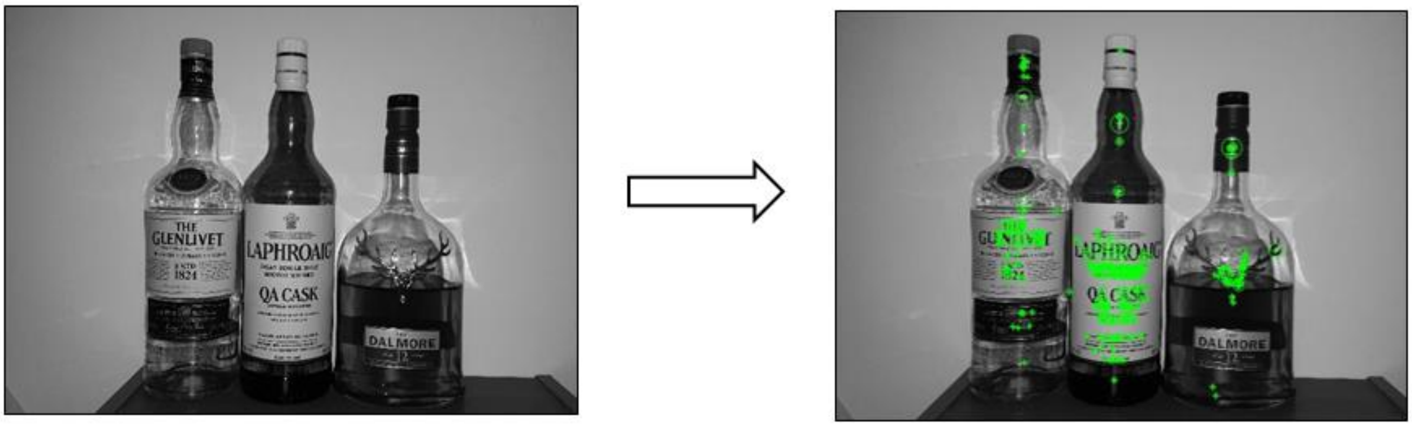
Now we will discuss the main steps:

<u>Step 1 - Detect the strongest feature points in both images:</u>

For each image, I detect the strongest feature points using the Speeded-Up Robust Features (SURF) algorithm. The SURF algorithm, which was first presented by Herbert Bay in 2006, is a detector and a descriptor for points of interest in images, and can be used for tasks such as object recognition. The advantage of this algorithm is its low complexity. This algorithm, based on LoG (Laplacian of Gaussian), looks for points of interests by finding blobs in the image, and compare them to their neighbors. You can read more about SURF here, and in the other links in the Reference section.

The result of this step (By using the Matlab functionality):





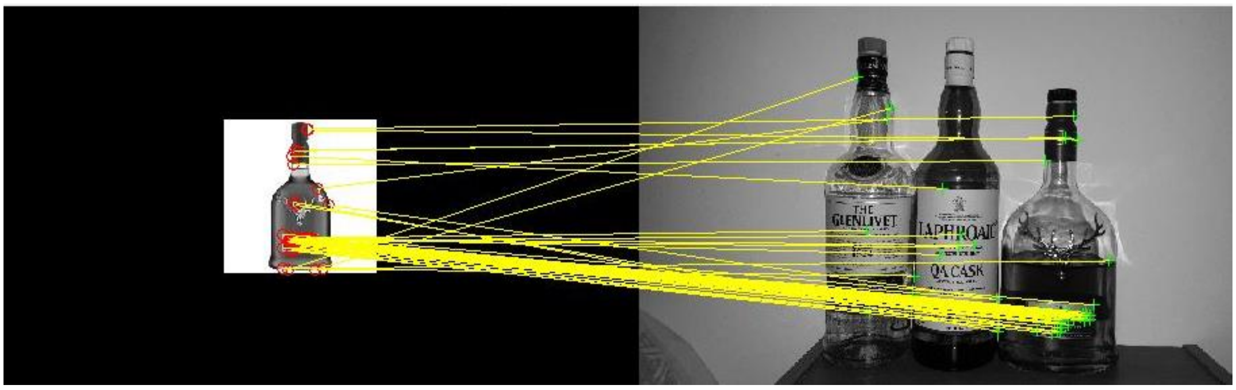
Step 2 - Extract feature descriptors at the interest points:

Extract feature vectors, also known as descriptors, and their corresponding locations, by using the interest points. The descriptors has been derived from pixels surrounding the interest points. In Matlab, the function "extractFeatures" get us those descriptors.

Step 3 - Match the features using their descriptors:

I compared between the features I found on each image, and looked for similarity between them. By comparing the features of each image, we can determine if the object is in the image, and if so- where it is located.

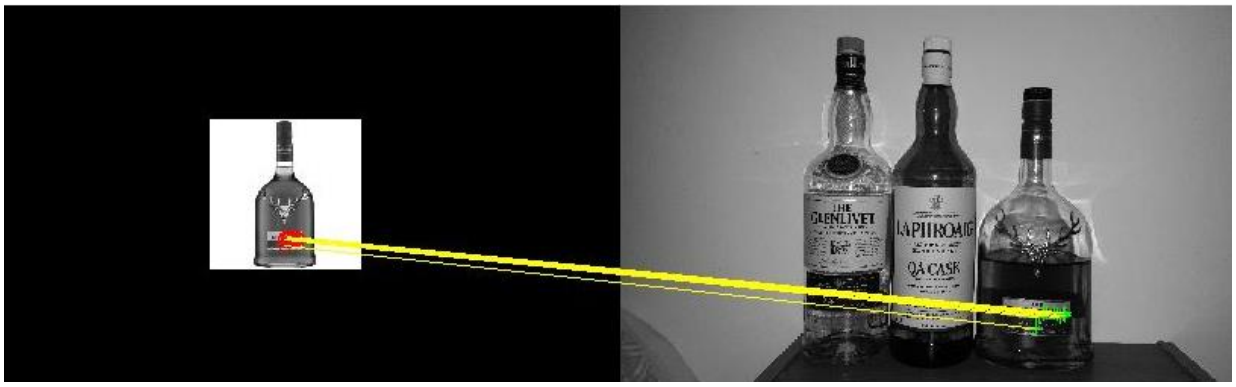
The result of this step:



Step 4 - Remove the outlier points:

Some of the matched points are the result of noise in the picture. As we can see, some of the points that have been found are incorrect. Like in edge detection, if we found an interest point, we can assume that there are more points in its area. So if we found an isolated point- we should probably remove it. For this mission, I used the "estimateGeometricTransform" function of Matlab. This function calculates the transformation related to the matched points, while eliminating outliers. With this transformation we can also locate the object in the image.

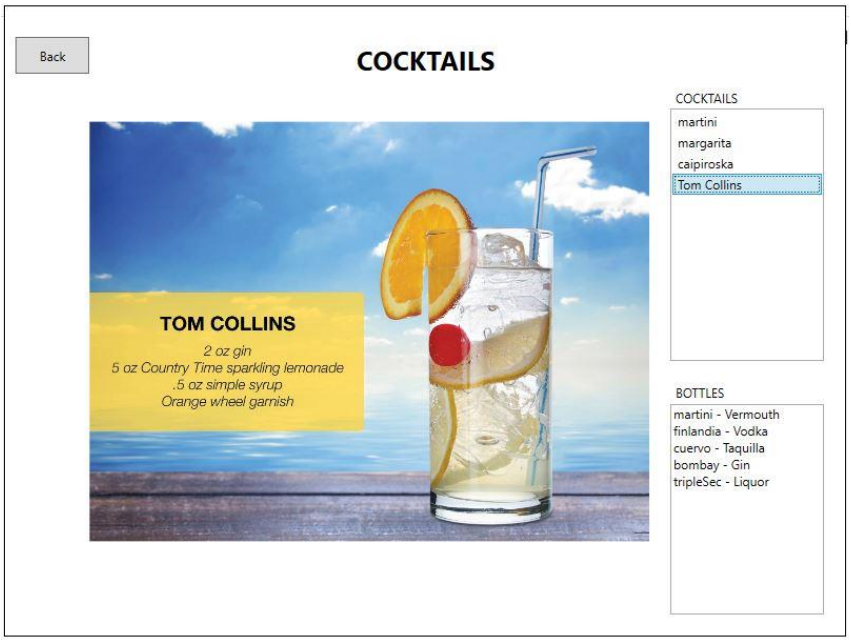
The result of this step:



The steps after the object recognition:

After getting a list of bottles, and returning it to the C# application, the application will look in the database for every cocktail that could be made by some combination of those bottles. Running first on the cocktail list assures us low complexity.

When finding the appropriate cocktails, the application will present it to the user:



Back

COCKTAILS

COCKTAILS
martini
margarita
caipiroska
Tom Collins

TOM COLLINS
2 oz gin
5 oz Country Time sparkling lemonade
.5 oz simple syrup
Orange wheel garnish

BOTTLES
martini - Vermouth
finlandia - Vodka
cuervo - Taquilla
bombay - Gin
tripleSec - Liquor

Implementation

User Interface – implemented in c#, using WPF.

Data Base and logic of cocktail selection – implemented in c#. The DB is based on dynamic data, and implemented as a separate unit so it could be replaced easily with connection to server.

Image Processing – implemented in Matlab, and based on SURF algorithm.

Results

Although the approach to the problem was clear to me, I discovered many issues that needed to be overcome while working on the system.

My first version of the application was based on text recognition. Even though the complexity was pretty good, I found out that most of the text recognition algorithms do not handle some special font and letters. In addition, it was hard to handle some rotations of the texts.

My second version was based on Object Recognition algorithm- SURF. I had some issues with this technique too.

The most critical issue was handling with similar bottles. Bottles with the same shape, same color or in general bottles that look the same while they are actually different. The biggest challenge was to find the right threshold, so that it will handle several issues:

- Success in distinguishing between bottles.
- Not missing existing bottles
- Handling different lights.

The best threshold I found creates a success with 18 images out of 20 I checked (90%). In the 2 failure tests it found a bottle that did not exist in the image.

Conclusions

My conclusions from this program are that it is a huge challenge to create an object recognition that fits lots of different situations, especially if you use different cameras and different lights. Also, the complexity is very unstable, and every step you make can change dramatically the complexity of the algorithm.

The final result is very stable, the complexity is pretty good (10-12 seconds for a result). In addition, the success rate is acceptable, and the failures occur because of a very strong similarity between two bottles.

This application can be improved by several aspects:

- Complexity- it can be improved by using parallel processing, and by rating the bottles in the DB by commonness.

- Recognition ability- it can be improved by separating the image by bottles, and checking every bottle separately. By doing this, we can know that there is only one bottle in each part of the image, and also improve the complexity (by parallel processing).

Additional information

- For watching the Demo: https://www.youtube.com/watch?v=kW6UkI_BVwo&feature=youtu.be
- For more information: aviramad@post.bgu.ac.il

References

https://en.wikipedia.org/wiki/Speeded_up_robust_features

https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

http://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf

http://www.mathworks.com/help/vision/examples/object-detection-in-a-cluttered-scene-using-point-feature-matching.html#btt5qyu