Last updated on Friday, August 18, 2017

2017 IFN680 - Assignment One (Particle filtering search)

Assessment information

- Code and report submission due on **Monday 25th September, 08.30am**
- Use **Blackboard** to submit your work
- Group size: three people per submission. Smaller group sizes allowed (1 or 2 people).

Overview

- You will implement a simple *particle filtering* search to estimate the pose of 2D patterns.
- The approach is a simplification of a method that is used for 3D pose estimation from monocular cameras. Restricting the scope to 2D avoids the challenges associated with projective geometry and does not require familiarity with camera calibration matrices and extrinsic projection matrices. However, the method you will implement generalizes well to the 3D case.
- You are provided with scaffolding code that you will need to complete.
- You will also perform experiments and report on your results.

Introduction

One of the most important problems robots face is to recognize objects and estimate their 6 degree of freedoms (6-DOF) pose parameters in less constrained environments than manufacturing plants. Most object detectors only provide a bounding box of the object of interest, but no prediction about the pose of the object (how the object is oriented with respect to the camera). However, in many applications the pose of certain objects in the environment is important. For example, the pose of containers, vehicles or objects that the robot manipulate.
Whenever a CAD model of the object of interest is available, an edge-based method can be employed to estimate the full pose using a monocular camera (6-DOF). There exist other pose estimatino methods based on *keypoints*. But most keypoints require relatively computationally expensive descriptors which maintain local texture or orientation information around stable points to be distinctive. Whereas the edge features are easy to compute and computationally cheap. Since edges are usually computed by image gradients, they are also moderately invariant to illumination and viewpoint.

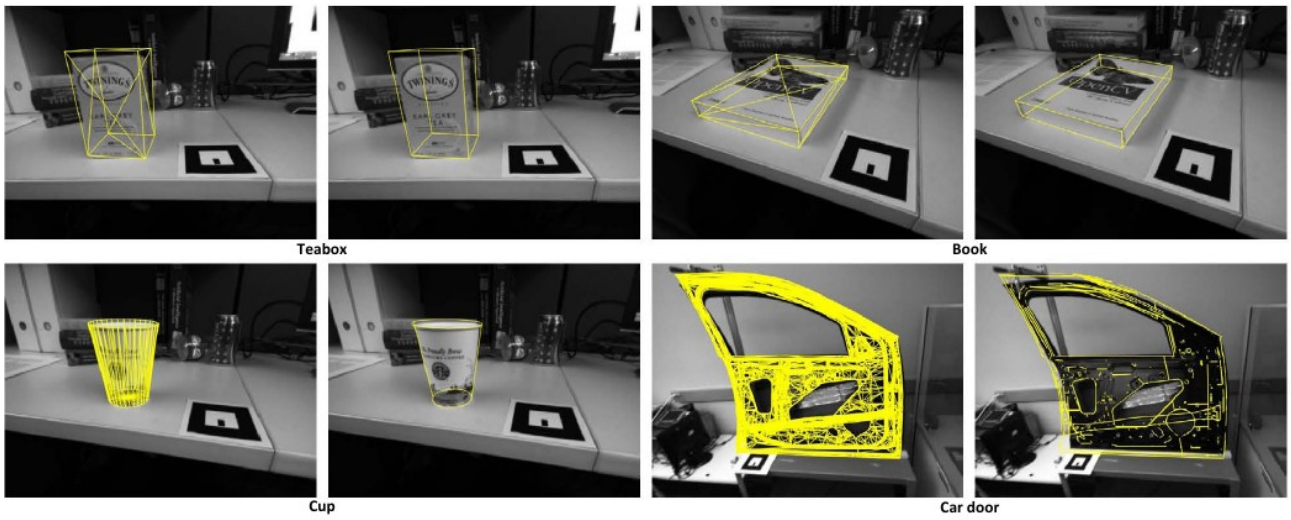


Figure 1: Original and simplified CAD models. The salient edges provide a good set of model edges to track.

The geometric information of a CAD model can be leveraged to score an hypothesis on the pose of the object by comparing the actual edges in the images with expected edges according to the pose hypothesis. Figure 1 illustrates how a 3D pose prediction system focusing on the salient edges can fit a CAD model.

Scoping the assignment to 2D

In order to keep moderate the mathematical complexity of the assignment, we restrict the the pose estimation problem to a 2D environment where objects of interest have 4-DOF.

Consider the image of red patterns over a blue background below. The code provided allows you to build any pattern that can be represented as a graph. To keep things simple, this example contains only triangles and squares. From this *edge image* we can compute the *distance image* that indicates for each pixel how far it is located from the neatest edge pixel.

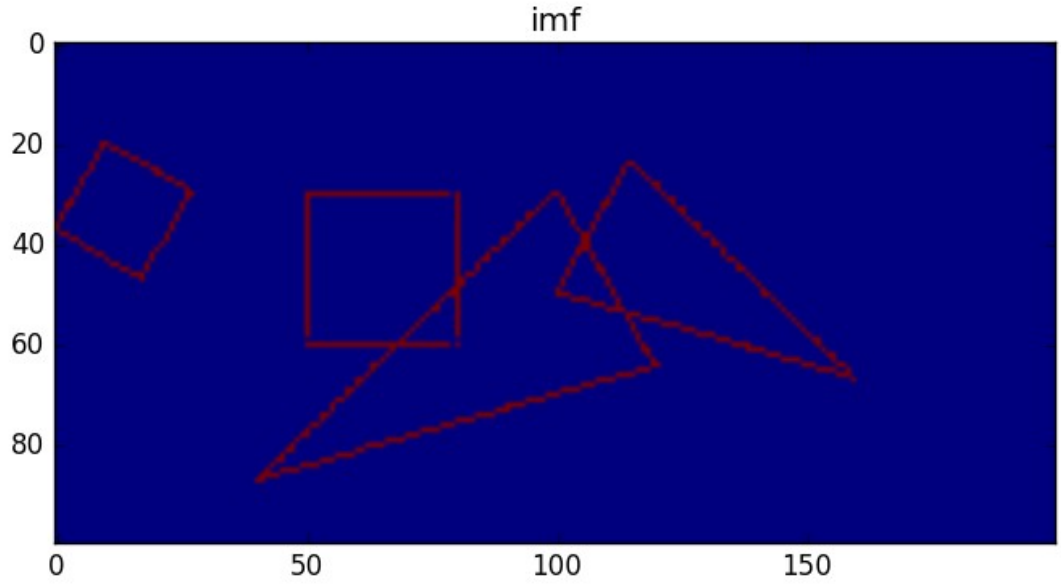


Illustration 2: Patterns

The image below is a float image where the pixel color indicates how far it is from the nearest edge pixel.

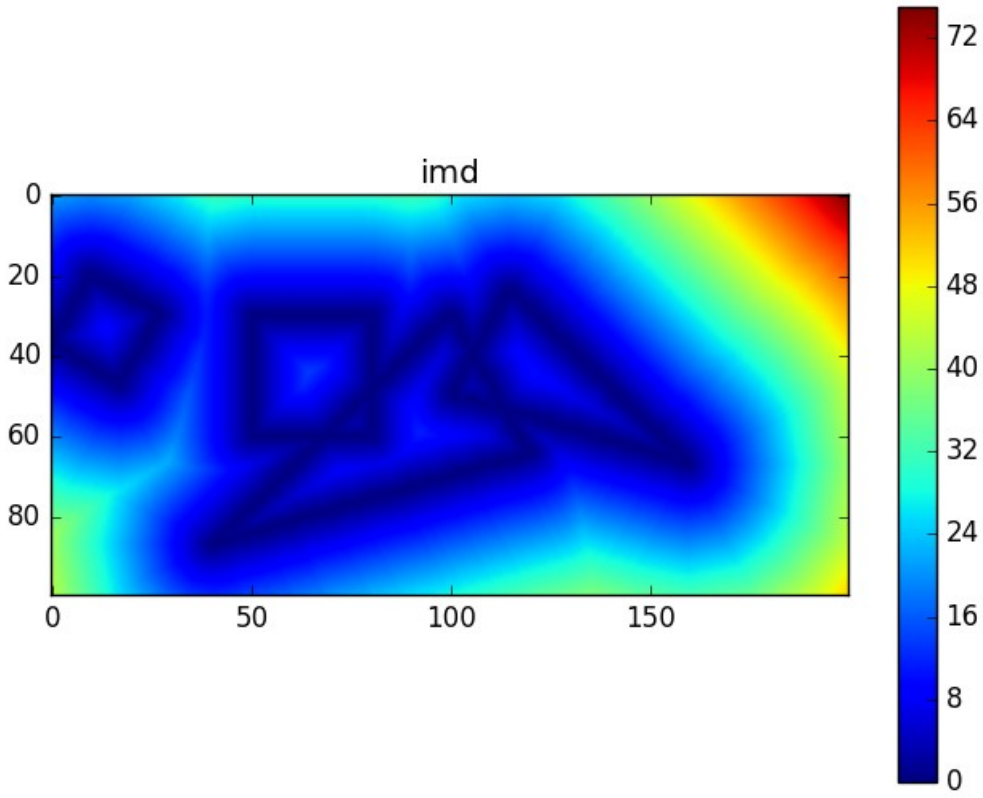


Illustration 3: In the distance image, the edge pixels are located at the bottom of the valleys

Assume that we wish to find the pose vector **(x,y,theta,scale)** of the large triangle. We start the search with a population of pose guesses illustrated in green in the figure below (Illustration 4). Note that the vertical y-axis is oriented differently in this figure! The search algorithm we use is a variation of the classical *particle-filter* search. The pseudo-code of this algorithm is similar to the the *Cross Entropy Search* that we saw in the lecture of Week 03.

 # Particle filter search
*Initialize population **W** with random guesses of pose vectors*
***Loop** until computational budget exhausted*
    evaluate** the cost **C[i]** of each **W[i,:]
    re-sample** the population according to **exp(-C[i])
    ***mutate** each new individual **W[i,:]** by adding some noise*

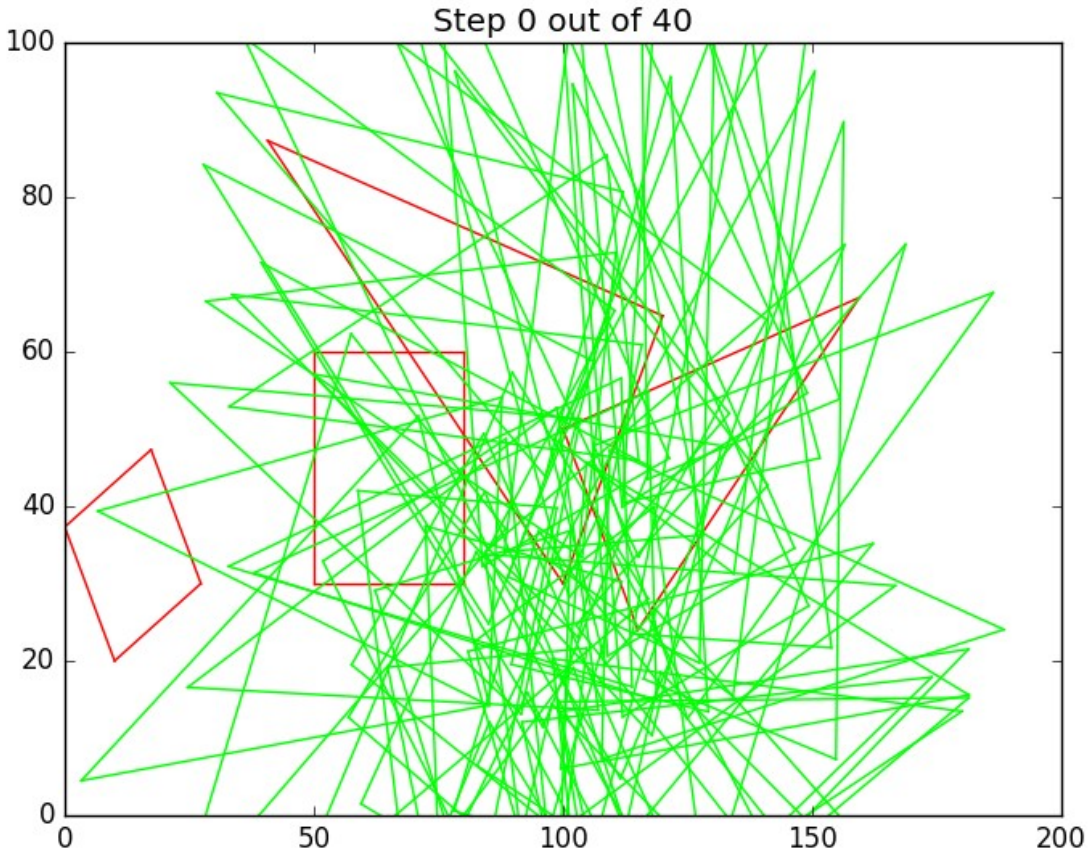
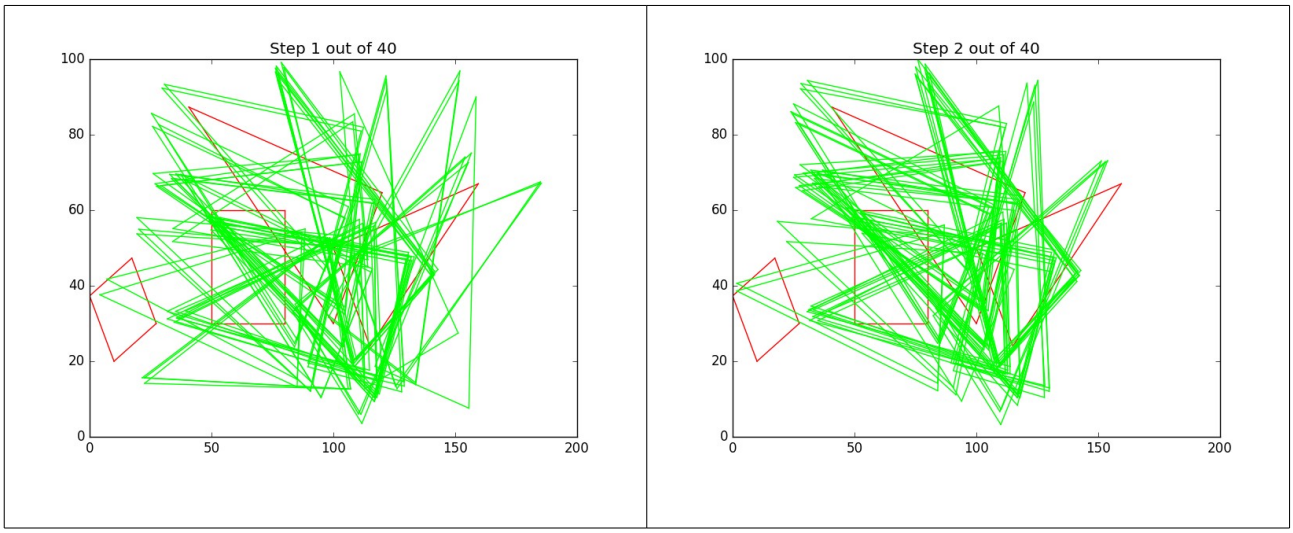
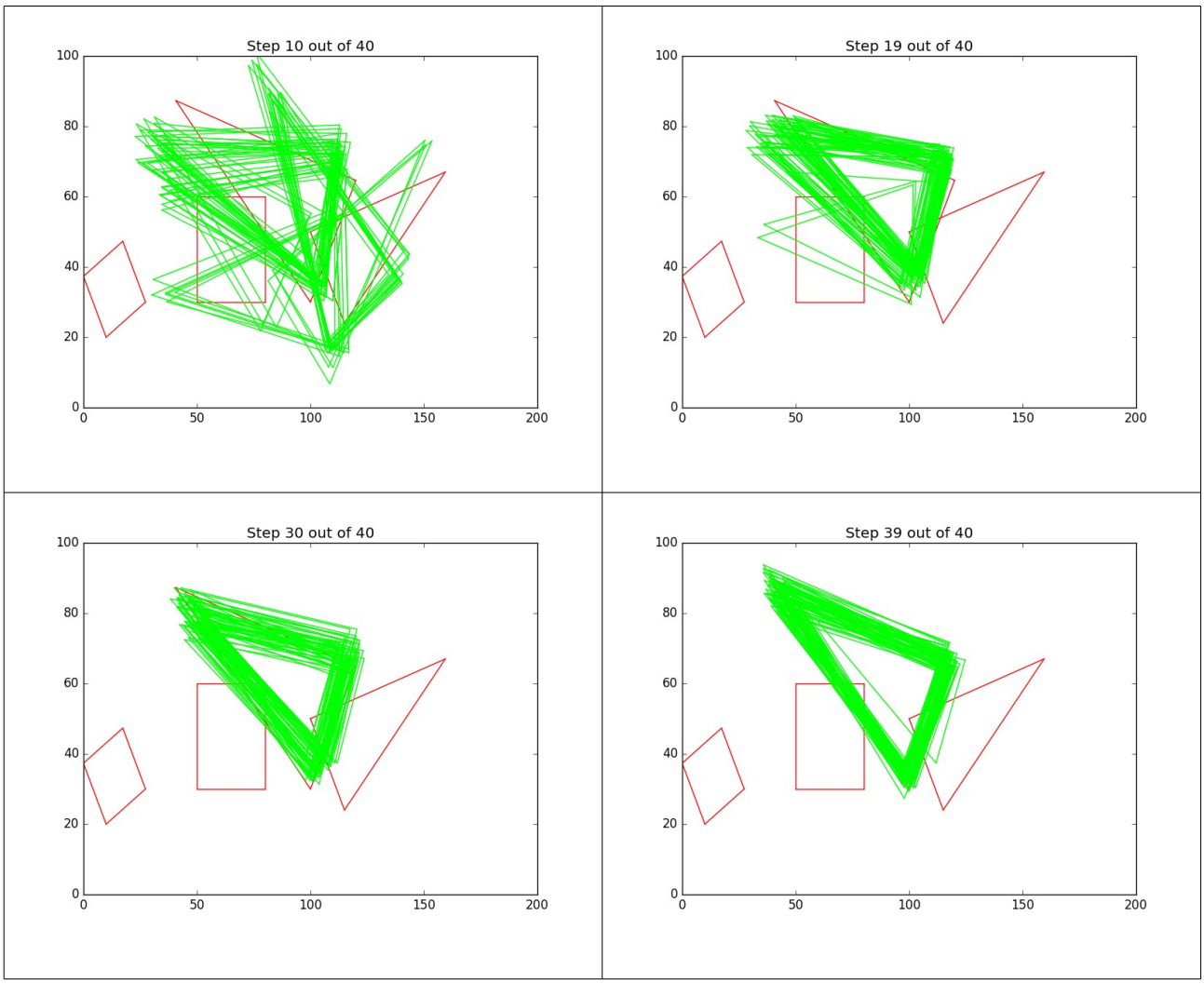


Illustration 4: Initial population at step 0. Guesses are in green.

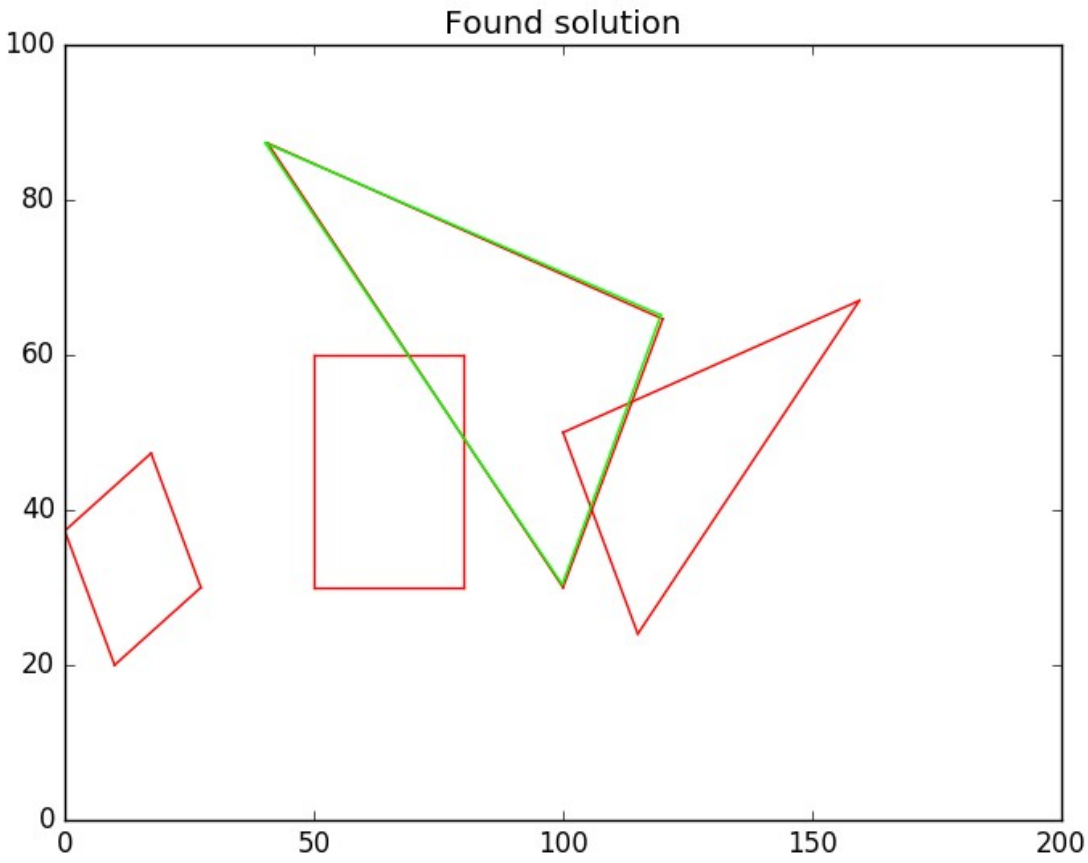
After evaluating the c*ost **C[i]** of each individual pose vector **W[i,:]*** where the cost is defined as the average distance of the pixels of an individual (green in Illustration 4) to an edge pixel (red in Illustration 4), the next generation of the population is created by re-sampling according to ***exp(-C[i]).*** That is, individuals with a low cost are more likely to be selected. To explore the pose space, some noise is added to each individual in the mutation phase. The loop is iterated until the maximum number of steps is reached.

The sequence of figures below illustrates how the population evolves in a typical run.





The best pose vector found in this run was the one shown in the figure below. The search was successful in this particular instance, but there is no guarantee of convergence to the global minimum as attraction to a local minimum can happen (unlucky initialization, small population can lead the search to a local minimum).



The best cost at each generation is shown in the plot below.

Your tasks

Code

- Complete the class **PatternPosePopulation** in the file **my_submission.py**
- Include in this file additional code you write to perform your experiments

Experiments

If we have time for the evaluation 1000 individuals, we could choose either of the following options for a particle filter search

- run 5 generations for a population of 200 individuals
- run 50 generations for a population of 20 individuals
- run 200 generations for a population of 5 individuals

It is not clear a priori what is the best choice. Using the two test images provided, first find a computational budget (that is, the number of individual evaluation done in a call to the function **particle_filter_search**) suitable for this problem. Then, investigate how to best balance the number of generations and the size of the population in this budget.

Describe your findings in the report. Use tables and figures to support your arguments. Make a recommendation.

Submission

You should submit via Blackboard a zip file containing
A **report in pdf format** strictly **limited to 6 pages in total.**

- explain clearly your methodology for your experiments
- present your experimental results using tables and figures

Your Python file **my_submission.py**

Marking Guide Focus

- **Report**:
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
- **Code quality**:
  - Readability, meaningful variable names.
  - Proper use of Python constructs like numpy arrays, dictionaries and list comprehension.
  - Header comments in classes and functions.
  - Function parameter documentation.

    - In-line comments.
- **Experiments**
    - Soundness of the methodology
    - Evidence based recommendations

Final Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions.
- Email questions to f.maire@qut.edu.au