Machine Vision
Coursework 2 (Regular)

Hugo Germain

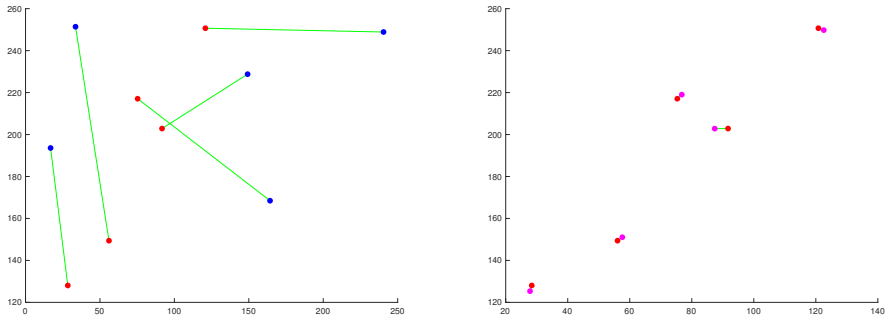
13 janvier 2017

In this Coursework, we will be revisiting the practicals from weeks 8 and 9, dealing with Homographies and Particle Filters. After presenting the work made in the lab, we will combine Tracking and Homographies into a new application.

Homographies - Part I

A) Practical1B

Refer to `practical1.m` for the full source code. After completion of the TODO's, here are the results we get :



(a) `pts1Cart` and their mapping to `pts2Cart` (b) Estimated `pts2Cart` using `HEst`

FIGURE 1 – Results from `practical1.m` using 5 points

The `calcBestHomography` function computes the best homography that maps `pts1Cart` to `pts2Cart`. We can see that the results are not perfect, we still get a slight offset on all points. The reason for this is that we are using 5 points, where 4 would be necessary to estimate a homography. Our system is *overconstrained* and cannot satisfy a perfect mapping (see *Fig.2*).

Moreover, our system is scale-ambiguous. Indeed, if we try multiplying the estimated homography by a constant, we come to the realization that we get the exact same results. This can be understood by explicitely writing the equations :

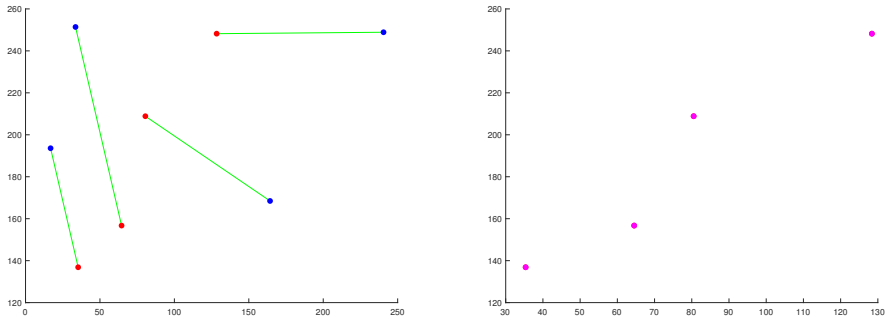
$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Hence, we can write that in Cartesian coordinates we get :

$$x = \frac{\phi_{11}u + \phi_{12}v + \phi_{13}}{\phi_{31}u + \phi_{32}v + \phi_{33}}$$

$$y = \frac{\phi_{21}u + \phi_{22}v + \phi_{23}}{\phi_{31}u + \phi_{32}v + \phi_{33}}$$

As a result, we can see that multiplying the Homography matrix by a constant will not change the result on x and y as the denominator and numerator will compensate.



(a) `pts1Cart` and their mapping to `pts2Cart` (b) Estimated `pts2Cart` using `HEst`

FIGURE 2 – Results from `practical1.m` using 4 points

As expected, when using 4 points our system is perfectly constrained and we can map any four points to any other four points exactly.

B) Practical1B

Refer to `practical1B.m` for the full source code. After completion of the TODO's, here are the results we get :

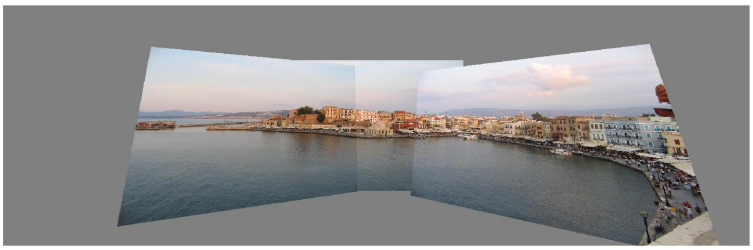


FIGURE 3 – Building a panorama by estimating homographies between images

Given the set of points `pts1`, `pts2` and `pts3`, that match feature points between images, our task is to compute the homographies transforming one set of point to the other. To do so, we end up reusing the code written in `practical1.m`. Once it is done, we then simply need to apply the homographies to the rest of the pixels, making sure they don't go outside the image borders.

Homographies - Part II

In this section, we will reuse the routine we wrote to estimate homographies, this time in order to predict where points on a plane will land on the camera.

A) Practical2

In `practical2.m`, we try estimate the pose of a plane relatively to the camera. To do so, we follow the instructions given in the slides, and compute the extrinsic transformation matrix T as :

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

The way we proceed is the following :

- First, we convert our cartesian image points to normalized homogeneous camera coordinates.
- Then, we estimate the optimal homography by solving a least-squared problem of the form $Ah = 0$.
- Lastly, we estimate the rotation matrix and translation vector from the homography using SVD and cross product.

Here are the results we get after running `practical2.m` :

```
T =
    0.9851   -0.0492    0.1619   46.0000
   -0.1623   -0.5520    0.8181   70.0000
    0.0490   -0.8324   -0.5518  500.8900
         0         0         0    1.0000
TEst =
    0.9846   -0.0511    0.1671   47.2990
   -0.1676   -0.5468    0.8203   70.9067
    0.0494   -0.8357   -0.5470  509.6346
         0         0         0    1.0000
```

We get some rather good initial estimates of the true transformation matrix.

B) Practical2b

In `practical2b.m`, we apply the previous routine of plane pose estimation to predict where points located relatively to that plane will land on the camera. Once we have estimated the pose of the plane, we perform the projection simply using the equation :

$$\lambda \tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{\Lambda} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\boldsymbol{w}}$$

Here are the results we get after running `practical2b.m` :

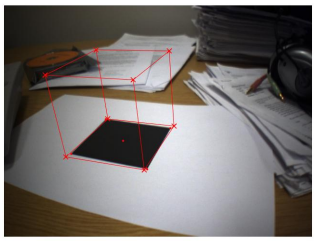


FIGURE 4 – Projected 3D cube after estimation of plane pose

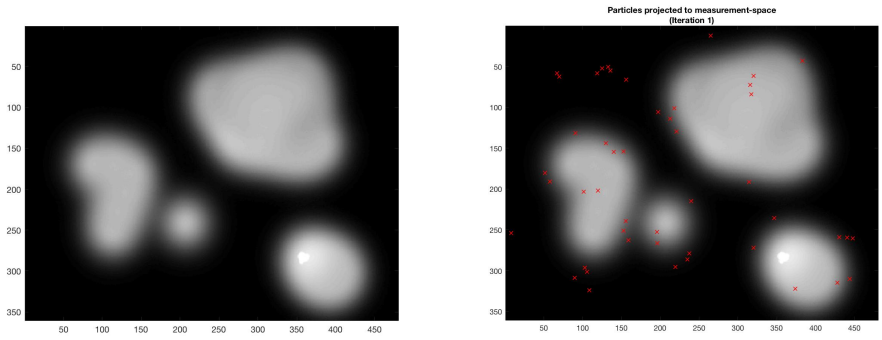
We can see that the results are not perfect. It is due to the fact that we are again using 5 points to estimate the homography, which overconstrains the system. If we were to use 4 points instead, we would most likely get more accurate results.

Condensation

In this section, we will take a look at the two practicals `practical9a.m` and `practical9b.m`.

A) Practical9a

In this program, we implement condensation. We randomly generate particles, and assign weights from a given likelihood to then resample from those particles and add some noise. In our case, the likelihood is a an image (`'abstract.png'`), and we start by giving the same weight to each particle. If we run the program only once, here are our results :

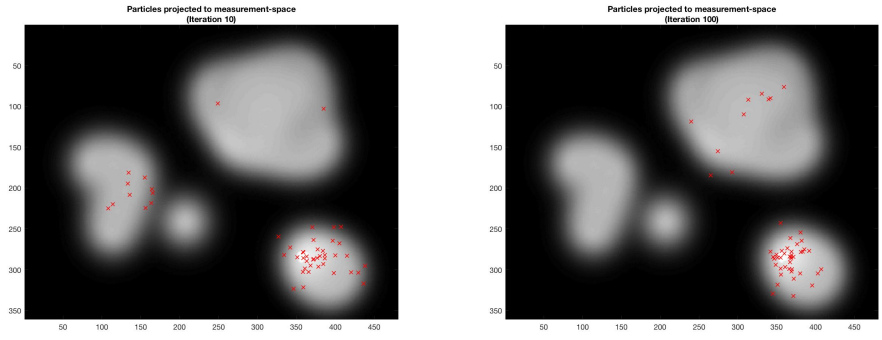


(a) Likelihood (b) Particles distribution after one iteration

FIGURE 5 – Results from `practical9a.m` after 1 iteration

We can see that the particles are distributed rather randomly in 2D space. This is normal, as at this stage we only compute the weights once using the likelihood, and don't reuse that information for further sampling.

Let us now increase the number of iterations to see the effects :



(a) Particles distribution after 10 iterations (b) Particles distribution after 100 iterations

FIGURE 6 – Results from `practical9a.m` after 10 and 100 iteration

At each iteration we take into account the likelihood to resample within the randomly generated particles. At each step we add random noise to make sure that two samples won't have the same coordinates and ensure a certain motion (Brownian noise).
Refer to `practical9a.m` for the full source code.

B) Practical9b

In this section, we apply condensation to track a given template. As previously, we start by sampling randomly and give weights to our estimates using a likelihood function before resampling and randomizing. In our case, the likelihood is given by similarity to a template patch.

Again, at each step we sample using weights and randomize the parameters of the obtained particles (here, the 2D position) to ensure uniqueness and movement. Here are the results we get :

(a) Particles distribution at time t=2 (b) Particles distribution at time t=22

FIGURE 7 – Results from `practical9b.m` at time t=2 and t=22

We can see that after a couple of image, the particles land on the car as expected, although a few are still wrong. To improve our system, we can add velocity. To do so, we simply add 2 more dimensions to our particles, and base our equations on the Brownian motion with constant velocity when randomizing the particles :

$$\begin{cases} w_{t+1} = w_t + v_t + \epsilon_p \\ v_{t+1} = v_t + \epsilon_v \end{cases}$$

Here are the results we get :

(a) Particles distribution at time t=3

(b) Particles distribution at time t=22

FIGURE 8 – Results from `practical9b.m` at time t=3 and t=22 using the velocity

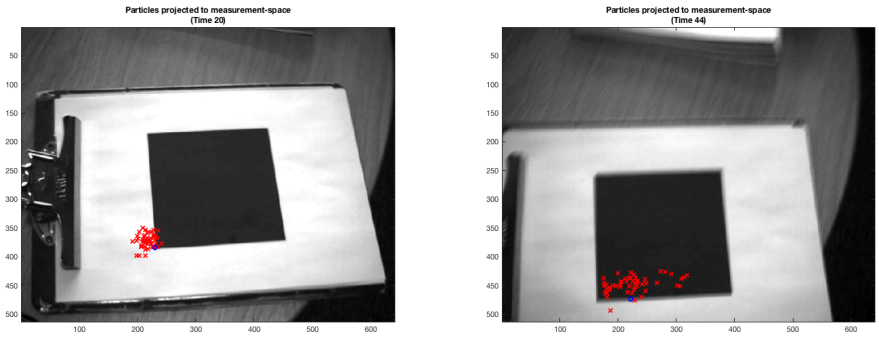
As expected, now that motion is taken into account, the static particles remaining on the building are gone. Indeed, their velocity don't match those whose likelihood is high, and get rejected from the particle filter after a few iterations.

Combining Tracking and Homographies

In this section, we make use of both previous practicals in order to perform the tracking of a distinct marker. First, we will take a look at a tracking program based on a particle filter, and then use the tracked position of the four corners to augment our video by projecting a 3D cube.

A) Practical9c

Let us first reuse our particle filter to track the corners of the black template. We will proceed in a similar fashion as previously. Using one template for each corner, we are able to assign weights by computing the similarity of the patches to the template. Here are the results we get :



(a) Optimal estimate at time 20

(b) Optimal estimate at time 44

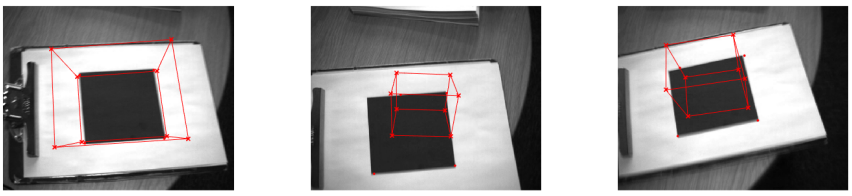
FIGURE 9 – Results from `HW2_practical9c.m` at time 20 and 44

We can see that for the first couple of frames, the tracking is performed nicely. However, when the footage gets shakier, the estimate slides along the bottom edge as the particles don't land on the corner anymore and the template cannot be found efficiently.
We will see that this induces some problems later on when projecting.

B) Tracking and Homographies

Here, we use the previously tracked corners to estimate the homography and project a 3D cube on the video. Again, we will be using the code from the practicals to perform this part. Having only four points, our problem is no longer overconstrained. However as we it will also lead to a lack of precision especially if our tracking is not done precisely.

Here are the results after running `HW2_TrackingandHomographies.m` :



(a) Projection at time 19 (b) Projection at time 57 (c) Projection at time 103

FIGURE 10 – Results from `HW2_TrackingandHomographies.m` at time 19, 57 and 103

As the tracking for each corner was not performed precisely, the resulting projected cube quickly slips off the given pattern.
There are a few ways we could improve the way tracking is performed, that would result in a more robust projection. The first one would be to increase the amount of particles generated and the range of the noise. By doing so, we would decrease the chance of "loosing" the corners when the cameras shakes heavily, as the search window would be bigger. Another way would be to add more dimensions to our particles. Besides the 2D position, we could incorporate velocity as we did in Practical 9, and have a 4-Dimensional particle system. Lastly, we could imagine adding more trackers. That way, even if one of them presents a certain error, other points will compensate. However, on this particular black rectangle, there is no other distinctive patch that could be easily tracked.

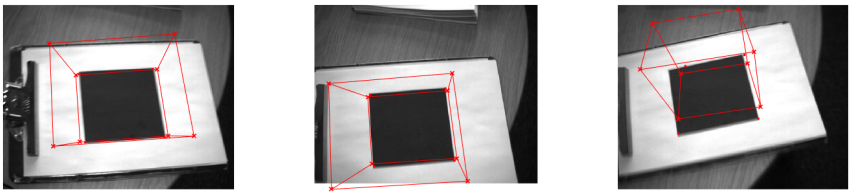
Extra Credit

Let us explore some additional improvements we can make on our particle filter.

Reducing the Search Space

A first idea, is to reduce the search space to the particles that land on the edges. To do so, we first run an edge detector on the image (ex : Canny Filter). Then, we assigning weights from the patch similarity, we add an extra weight for particles that have landed on an edge. That way, particles that will land on an edge will have priority over other particles and will be much more sampled from on the next iteration. We implement this improvement, along with more particle samples and a wider randomness distribution in `HW2_Practical9c.m`.

Here are the results we get :



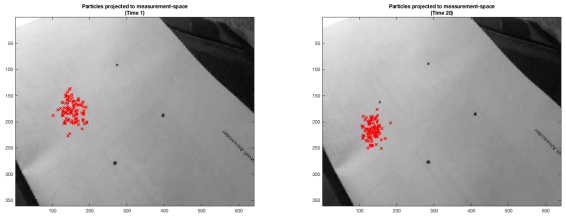
(a) Projection at time 19 (b) Projection at time 57 (c) Projection at time 103

FIGURE 11 – Results from `HW2_TrackingandHomographies.m` at time 19, 57 and 103, after reducing the Search Space

Overall, the results are slightly better. However we still get some errors still due to a bad tracking.

Using a custom video sequence

Here we will experiment with another sequence shot manually, this time with black dots used as trackers. The video images can be found under `/Pattern02`, and the `.mat` files have been modified to match the given trackers.



(a) Projection at time 1 (b) Projection at time 20

FIGURE 12 – Results from `Extra_HW2/HW2_Practical9c.m` at time 1 and 20 on a different video sequence.

Unfortunately, the tracking performs very poorly. Indeed, after the first couple of frames the particles diverge from the tracking point and rarely come back. This may be due to an error in the template parameters, or the way the trackers were designed. Due to a lack of time I was not able to perform further investigation on the matter and try to make another video sequence.

Conclusion

In this last Coursework of Machine Vision we covered the practicals from Week 8 and 9, and combined the two to build a tracking system that allows us to project a 3D Mesh on a custom plate. We were able to see the limitations of such a system, and experiment with various ways of improving it.