Homographies and Particle Filters

Theo Turner, MEng Mathematical Computation

January 15, 2018

1 Introduction

This project concerns *homographies* and *particle filters*. *Homography* is an isomorphism of projective spaces and is used in this context for calculating the pose of three-dimensional objects for image processing. *Particle filters* are sequential Monte Carlo methods, used here for tracking. The TODOs for each part of the assignment are documented in this report. For further explanation, please see the source code.

2 Homographies Part 1

2.1 A.

This part of the practical calculates the optimal homography between two sets of points. This is done by applying a direct linear transform to compute the best map between **pts1Cart** and **pts2Cart**.

First, the points are made homogenous by appending them with a row of ones. A matrix A is constructed, forming a system of equations representing exterior orientation (two equations for each point). Next, the direction problem $Ah = 0$ is solved by computing the singular-value decomposition $A = ULV^T$ where $\hat{h}$ is then set to the last column of **V**. This is finally reshaped into the 3x3 matrix **H**, giving the homography.

Note that the homography used here is ambiguous in regard to scale. This can be seen by linearly scaling it and observing that functionally it does not change–to do this, simply pre-multiply the function by a scalar.

In addition, the routine may act as an exact map between any two sets of four points. This can be empirically verified by adjusting the number of points in **pts1Cart**, changing the scale of the homogeneous transformation, and adjusting the matrix **A** accordingly.

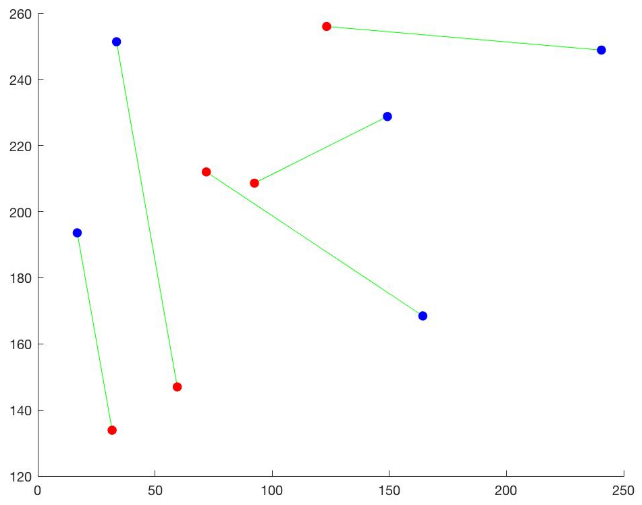

Figure 1: A generated set of five point pairs.

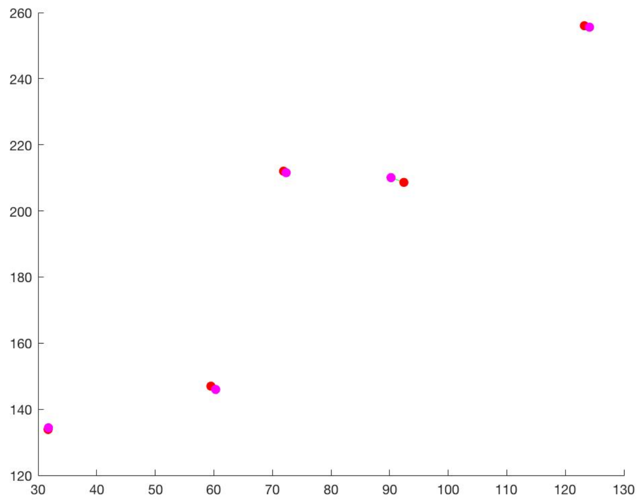

Figure 2: The best homography. Note the reduced distance between the points.

2.2 B.

This part of the practical uses the homography routine from part A. to stitch images, forming a panorama. The poses of the images are associated by a set of points. Again, the optimal homography between these is computed using the same function as before. The central image in the panorama is selected as the 'base' and surrounded by sufficient empty space to superimpose the other two images on it. Each other picture is mapped on to the base. First, the pixel position is transformed to the appropriate point for the final panorama. Subsequently, the transform is normalised and the panorama displayed.

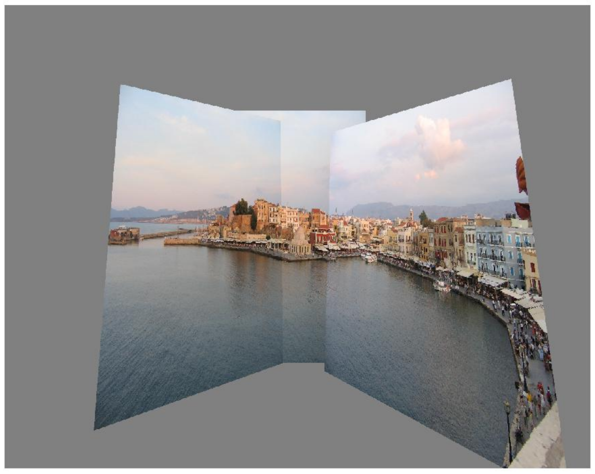

Figure 3: The final panorama. Observe that the two non-'base' images have been rotated in 3-space.

3 Homographies Part 2

3.1 C.

This part of the practical concerns the geometry of a single camera. Observed points on a plane are taken and their position in the final camera image is predicted. This is done by means of an estimation of the Euclidean transform between the plane and camera.

The estimate of the position of points on the plane is computed using a pinhole camera model. The points in **XCart** are passed through a *projective* camera defined by two matrices, one extrinsic and one intrinsic. First, the cartesian points are converted to homogeneous coordinates–again, this is done by appending ones (though in this case in three dimensions). The camera's frame of reference is then removed by applying the extrinsic matrix to the homogeneous coordinates (pre-multiplication). This leaves a homogeneous camera coordinate matrix. This matrix is projected into normalised camera coordinates by removing the fourth row. At this point, applying the intrinsic matrix (pre-multiplication) leaves the image coordinates. These are then be converted back to Cartesian coordinates **xImCart**.

Having computed Cartesian image coordinates, noise is added to simulate searching for the points in a noisy image. To estimate the pose of the image 'plane' relative to the camera, the Cartesian image coordinates **xImCart** are once again converted to homogenous in the same way as before. These are then converted to normalised camera coordinates **xCamHom** by application of the *inverse* of the intrinsic matrix (pre-multiplication). Finally, the best homography between **xCamHom** and the real-world positions (Cartesian coordinates) is computed using the same function as that used in **Homographies Part 1**.

In order to rotate the images in space, a rotation matrix **R** is computed. The first two columns of this matrix are taken from the first two columns of the homography using singular value decomposition. The third column is estimated by the cross-product of the first two columns. The third column is also scaled appropriately, and if the determinant of the matrix is negative, the third column is negated to make it positive. This leaves an estimated translation. To ensure that the translation is in the correct direction, if the z-component is found to be negative, the translation is negated and the rotation matrix is adjusted accordingly. Finally, the transformation matrix **T** is constructed using the rotation and translation components.

3.2 D.

This part of the practical uses the routines in part C. to draw a wireframe cube on an augmented reality marker, in this case a planar (flat) black square. The transformation between the camera and marker is calculated and a wireframe cube is projected on top of the square.

First the extrinsic matrix relating the plane position to the camera position is computed. This is done using the same function as is in part C. The wireframe cube is drawn by projecting its vertices through the projective camera and subsequently drawing edges between these. In this case, the results do appear fairly realistic, though the base of the projection does not match the detected corners of the augmented reality marker perfectly–this highlights the difficulty of using of a projective camera.

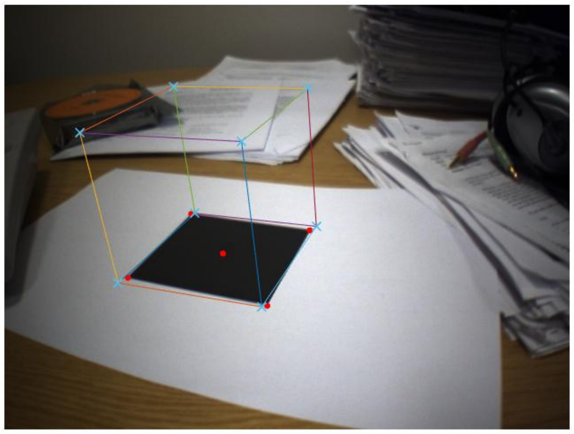

Figure 4: Wireframe cube projected over the augmented reality marker. Note that the base of the cube doesn't align perfectly with the detected corners of the square, highlighting the weakness of the use of a projective camera.

4 Condensation

4.1 E.

This part of the practical involves the Conditional Density Propagation (Condensation) Algorithm. The algorithm is performed through iterative factored sampling on simulated observations with the addition of an emulated model for motion. First, the data is normalised by dividing each sample by the sum of all samples and particles are initiated at random positions.

Subsequently, the iterative loop starts. The cumulative sum of weights is computed, the distribution at time **t-1** is resampled favouring higher posterior probability and samples to propagate (based on random thresholds) are chosen. The new position of the particles is predicted based on their distribution at time **t-1** by applying the motion model to each sample. The motion model is Brownian, meaning Gaussian noise is added. The new state of the particles is measured–each predicted particle position in state space is converted to measurement-space. The likelihood of each particle is then computed. Once again, the weights are normalised (in the same way as before) and the new set of particles is set to be the old set, ready for the next iteration. The iterative loop then ends.

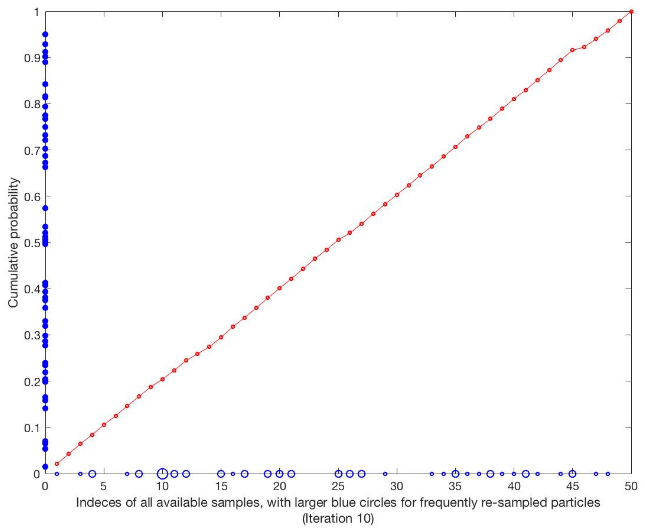

Figure 5: Cumulative probability over particle samples.

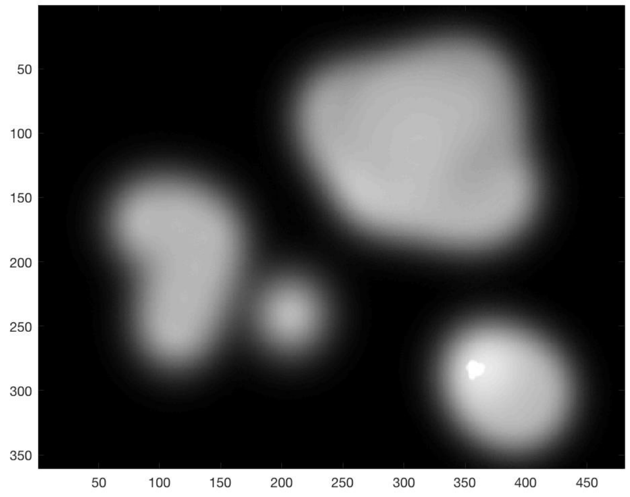

Figure 6: Space prior to projection.

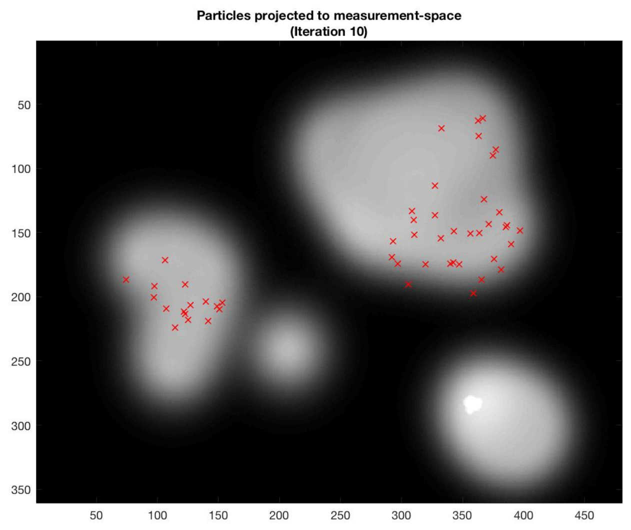

Figure 7: Final projection of particles over measurement space.

4.2 F.

This part of the practical involves tracking a moving shape in a sequence of frames. For all intents and purposes, this is done in the same way as part E. but with two key differences. Firstly, the likelihood is manually computed using the **MeasurePatchSimilarityHere()** function. Secondly, to improve accuracy and reduce false positives, noise is added to both position and velocity. It is particularly important to set velocity noise such that a good balance is found between accuracy and coverage.

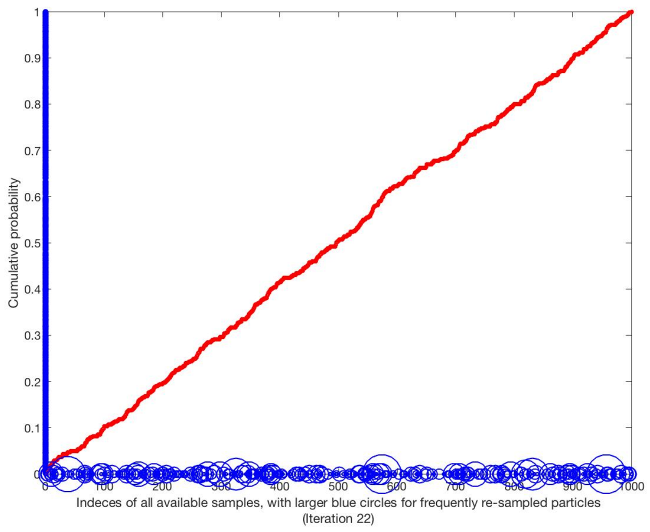

Figure 8: Cumulative probability over particle samples.

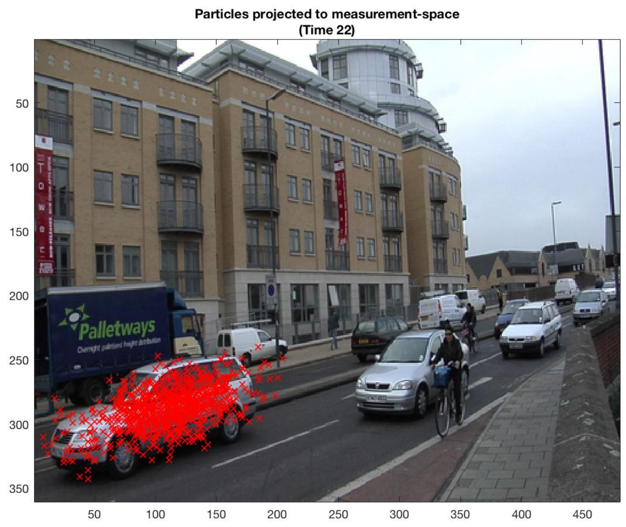

Figure 9: Final projection. Note that the car has good coverage and there are no distinct misclassified areas. Reduction of velocity noise leads to false positives, particularly around other vehicles and the tops of the buildings.

5 Combining Tracking and Homographies

5.1 G.

This part of the practical tracks the corners of an augmented reality marker over a sequence of frames. A particle filter tracks the entire posterior distribution with the maximum a posteriori estimate saved at each frame. Normalisation and the iterative loop (over all frames) is the same as in parts E. and F. but with three key changes. Firstly, the level of added noise is increased. Greater noise increases the 'search coverage' but reduces the density of this coverage. Again, a good balance must be found between the two–in this case, a slightly higher level of noise is appropriate. Secondly, the maximum a posteriori particle location is found using the weighted average of all particles. Thirdly, the weighted center of each patch of particles is computed for tracking each corner. It is also worth noting that the image sequence is stored in greyscale and as such **MeasurePatchSimilarityHere()** works in two dimensions rather than three.

5.2 H.

This part of the practical uses part G. to track each corner in turn over all frames. A wireframe cube is subsequently projected over the black square marker across the sequence of frames, effectively achieving *video see-through augmented reality.* Plane pose estimation and drawing of the wireframe cube is is done is done in the same way as in parts C. and D., replicated each frame.

The results look fairly realistic, however the projection of the wireframe cube is not particularly accurate when the camera moves quickly between frames. This is most likely due to a combination of two factors. Firstly, when the camera moves quickly, the corner of the black marker moves away from the center of the area searched at time **t-1** in fewer frames, thereby making each required adjustment more drastic. Secondly, the picture becomes blurred when the camera moves quickly, making the corners of the black marker less clear.

Two potential changes that could be made to improve the results would be further refining the motion model by experimenting with degrees of freedom as in part F. and improving the resolution/frame-rate of the footage to combat tracking difficulties encountered when the camera moves quickly.

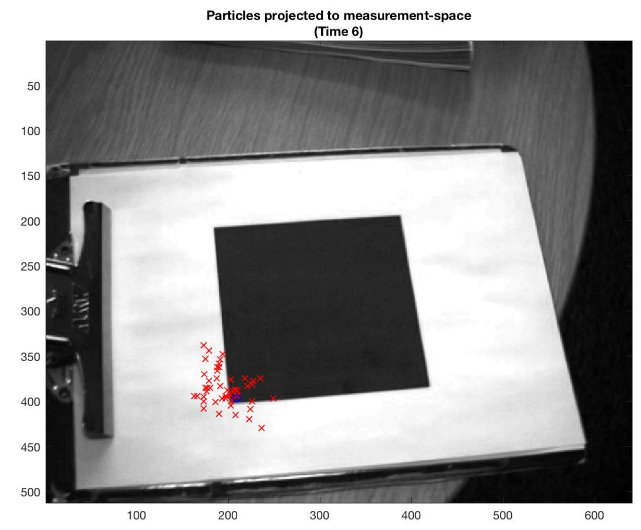

Figure 10: Augmented reality marker corner tracking. Each corner is tracked for all frames in turn. Increasing noise increases the 'search area' (red crosses) but reduces the coverage density.

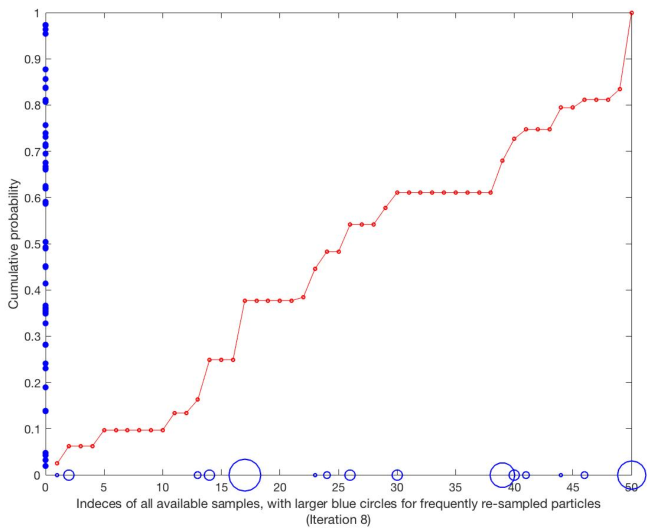

Figure 11: Cumulative probability over particle samples for a single corner in one frame.

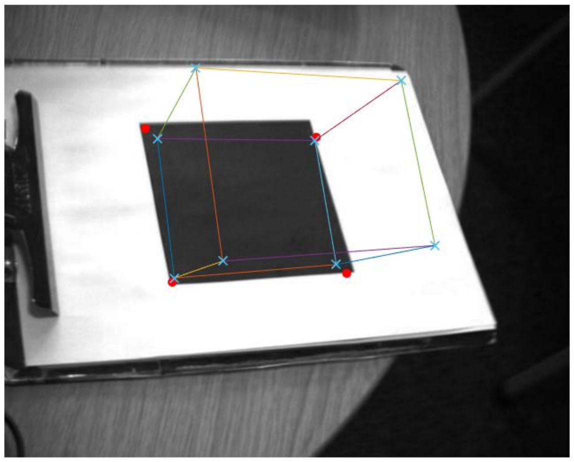

Figure 12: A particularly poor frame from the final video with the 'augmented reality' cube rendered over the marker. Note that in this frame, neither the corner tracking nor the wireframe cube aligns perfectly. The cause of this is most likely fast camera movements.

6 Extra Credit

6.1 Edge Detection

Edge detection reduces the search space of where particles may be located. This is achieved through use of MATLAB's **edge()** function. Note that use of the function requires two add-ons for MATLAB:

1. Statistics and Machine Learning
2. Image Processing

Edge detection is done during the iterative loop and the search space is reduced by limiting it to the size returned by the **edge()** function. Sample weights are re-adjusted accordingly. See **HW2_Practical9c.m** for exactly how this is done.