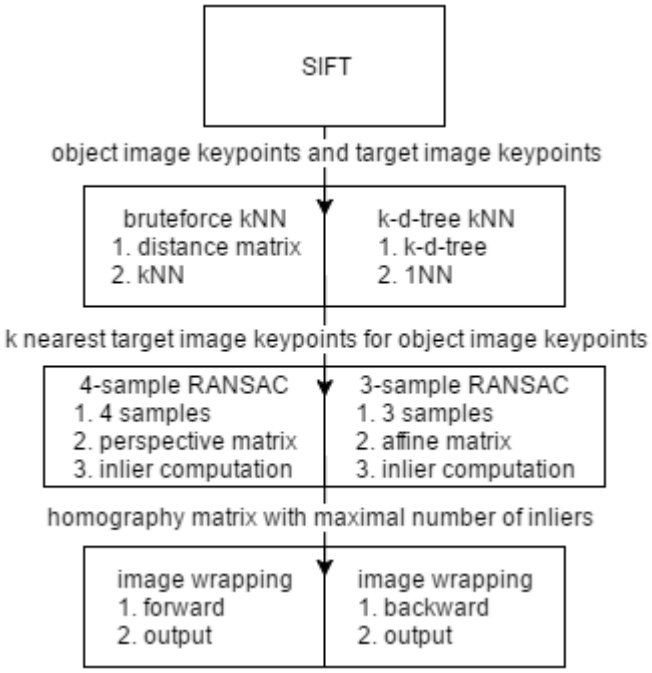
Computer Vision Homework #2

- 程式流程



• PART 1 : bruteforce kNN vs. k-d-tree kNN

1NN	bruteforce 1NN		k-d-tree 1NN
整體速度		<	
資料建構速度	distance matrix	<	k-d-tree
搜尋最近點速度		>	
搜尋最近點精確度		>	
			(output/output_11/kdtree/..)

- 在搜尋最近點速度的比較中，
  k-d-tree 1NN 理論上應該比 bruteforce 1NN 還快，可能是回溯查找的部分降低效率。
- 因為測資 keypoint 數量不算太多，
  kNN 在整個程式所占耗時比重低，因此選擇 bruteforce kNN 以確保搜尋最近點精確度。

```
--- bruteforce 1NN start
----- distance matrix start
11.284sec
----- distance matrix end
11.52sec
--- bruteforce 1NN end

--- k-d-tree 1NN start
----- k-d-tree start
0.402sec
----- k-d-tree end
1.456sec
--- k-d-tree 1NN end
```

• PART 2 : 4-sample RANSAC vs. 3-sample RANSAC

4-sample RANSAC	3-sample RANSAC
perspective mapping $H(x_i, y_i) = (u_i, v_i)$ for $i = 0, 1, 2, 3$ $\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wu \\ wv \\ w \end{bmatrix}$	affine mapping $H(x_i, y_i) = (u_i, v_i)$ for $i = 0, 1, 2$ $\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$
`getPerspectiveTransform([x,y],[u,v])`	`getAffineTransform([x,y],[u,v])`
solve $DOF(H) = 8$; $h_{22} = 1$ $\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -u_0y_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -v_0x_0 & -v_0y_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -v_2x_2 & -v_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -u_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -v_3x_3 & -v_3y_3 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \\ u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$ `solve(B, [u,v], H);`	solve $DOF(H) = 6$; $h_{20} = 0, h_{21} = 0, h_{22} = 1$ $\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \\ u_1 \\ v_1 \\ u_2 \\ v_2 \end{bmatrix}$ `solve(B, [u,v], H);`

solve $DOF(H) = 9$
$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -u_0y_0 & -u_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -v_0x_0 & -v_0y_0 & -v_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -u_3y_3 & -u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -v_3x_3 & -v_3y_3 & -v_3 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
`SVD::solveZ(A, H);`
`eigen(A.t()*A, eigenvalue, eigenvector);`	

	`(output/output_11/affine/..)`

◦ 即使兩種矩陣所得的 inlier 數目相當，affine mapping 會有較嚴重的變形，
因此還原品質以 perspective mapping 優於 affine mapping。

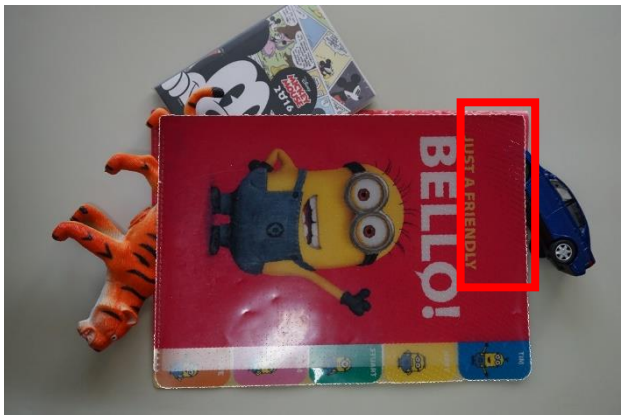
	
perspective mapping	affine mapping

```
SVD::solveZ(A,H) - (0,0,0,0)
[-0.00054715836, -0.0021845563, 0.93470877;
 -0.00020804224, -0.00083059911, 0.3553924;
 -1.8386522e-06, -7.3411602e-06, 0.003141033]

eigen(A.t()*A) the eigenvector with smallest eigenvalue
[-0.00055129215, -0.0021751074, 0.93448734;
 -0.00021053404, -0.00082731637, 0.35597461;
 -1.8403931e-06, -7.3078422e-06, 0.0031326653]

solve(B,C,H)
[-0.17418434, -0.69549739, 297.57477;
 -0.066226803, -0.26443702, 113.14154;
 -0.00058535073, -0.0023372096, 1]

getPerspectiveTransform
[-0.17418464, -0.69549733, 297.57495;
 -0.066226825, -0.26443705, 113.14156;
 -0.00058535108, -0.0023372089, 1]
```

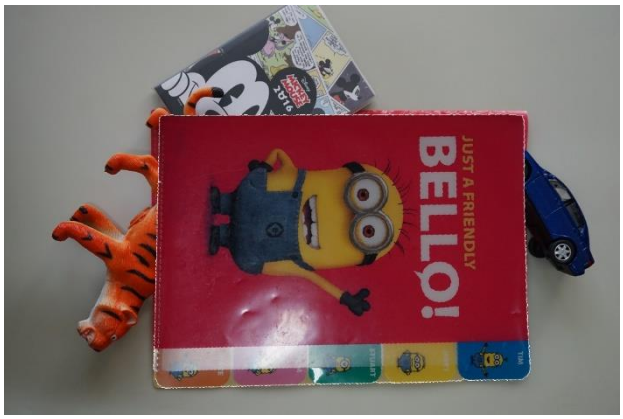
• PART 3 : forward wrapping vs. backward wrapping

forward wrapping	backward wrapping
$H(x_i, y_i) = (u_i, v_i)$	$H^{-1}(u_i, v_i) = (x_i, y_i)$
Nf_n.jpg	Nb_n.jpg

◦ forward wrapping 從實數座標轉成整數座標會遺漏某些像素，而 backward wrapping 無此問題，因此還原品質以 backward wrapping 優於 forward wrapping。



forward wrapping



backward wrapping