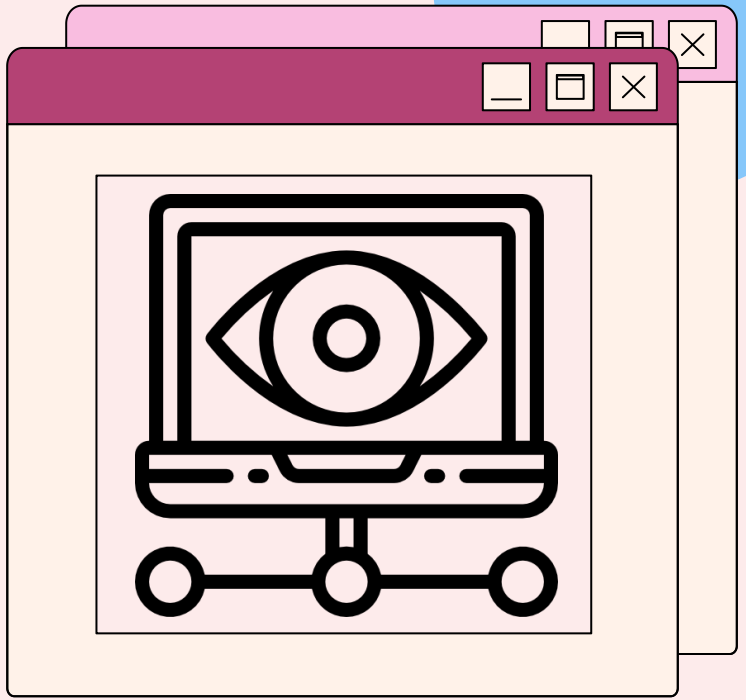
# Automatic Camera Pose Estimation



Hui Shi
28.03.2023

# CONTENTS

# 1. Introduction

## Motivation

- Camera pose estimation
- Exterior orientation estimation
  – widely used
- 3D –> 2D correspondences
  – not always available

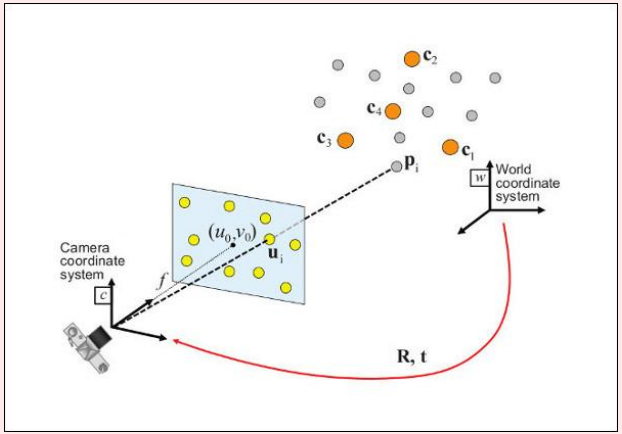



Figure 1. Exterior orientation estimation

# 1. Introduction

## Solution

- Estimate 3D –> 2D correspondences
- Fiore's exterior orientation estimation[1]

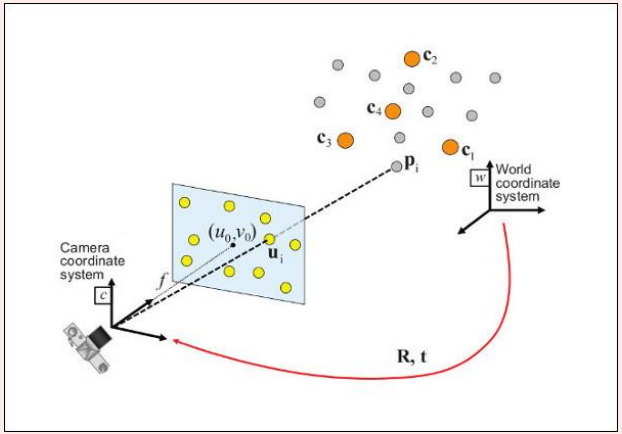
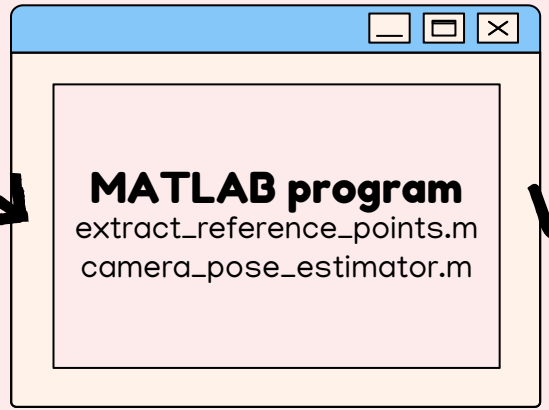



Figure 1. Exterior orientation estimation

# 2. Design

## Input

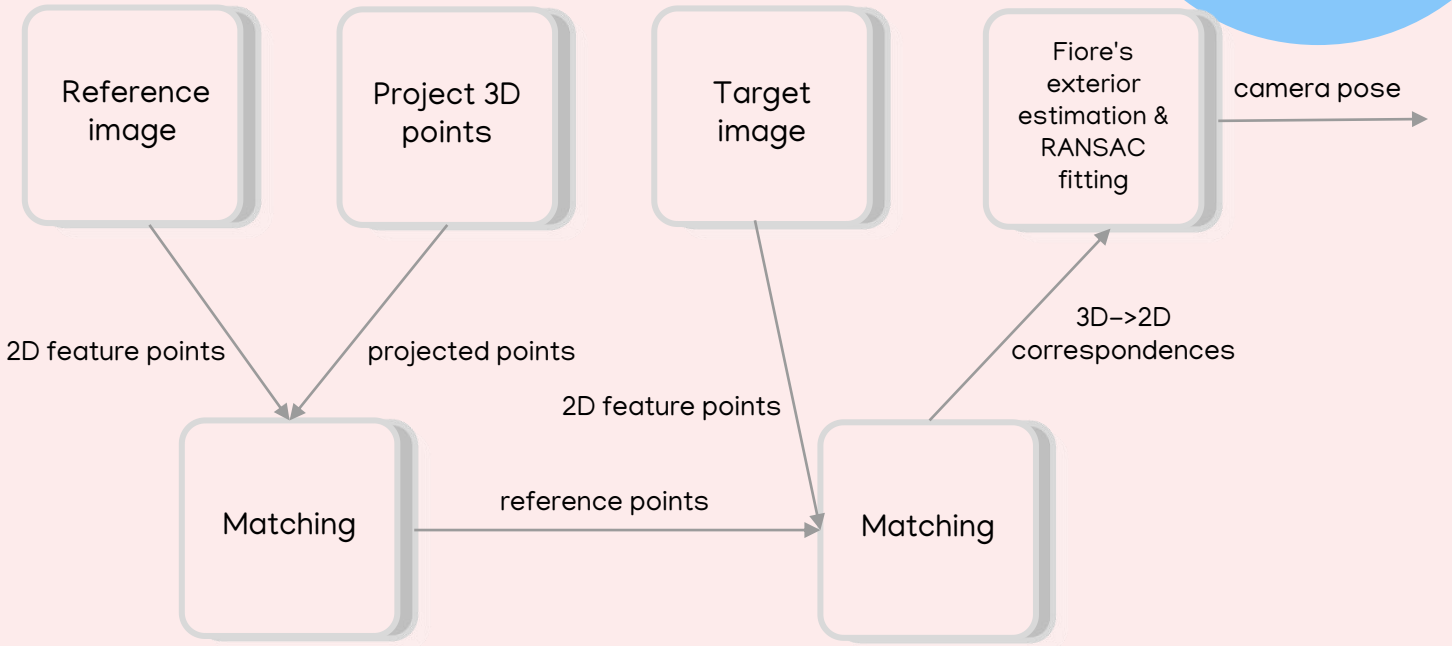
- 3D model
- Reference image
  & calibration data
- Target image
  & camera intrinsic parameters



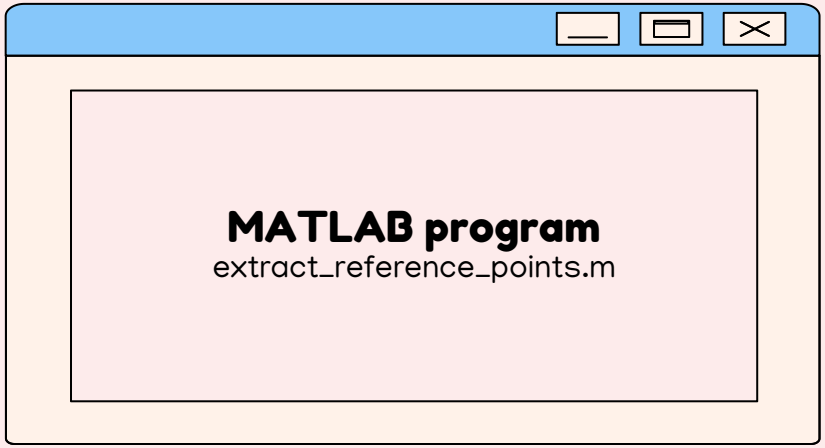

## Camera pose

# 2. Design

## Algorithm



# 2. Design



# 2. Design

## Extract 2D feature points from reference image,

using SIFT(Scale Invariant Feature Transform) from VLFeat library [2][3]

extract_reference_points.m

```
% 1. Extract 2D feature points from the reference image, using SIFT from the vlfeat lib.
% ref : https://www.vlfeat.org/overview/sift.html#tut.sift.match
img_ref = rgb2gray(img_ref) ;
img_ref = im2single(img_ref);
[frame_ref, descriptor_ref] = vl_sift(img_ref) ;

% sort the scale of the extracted frames and take the biggest ones
scale_f_ref = frame_ref(3,:);
[dump,scaleind_ref] = sort(scale_f_ref,'descend');
% change the num_feature (number of biggest frames) if the image size is too small
if size(img_ref,1) <= 1000 && size(img_ref,2) <= 1000
    num_feature = 1000;
else
    num_feature = 5000;
end
scaleind_ref = scaleind_ref(:,1:num_feature)';
frame_ref = frame_ref(:,scaleind_ref);
descriptor_ref = descriptor_ref(:,scaleind_ref);

% show the selected feature points
figure('Name','Reference points')
imshow(img_ref_RGB);
hold on
h_ref = vl_plotframe(frame_ref) ;
set(h_ref,'color','y','linewidth',2) ;
```

**Figure 2.** *Implementation of extracting 2D feature points from the reference image*

# 2. Design

## Extract 2D feature points from reference image,

using SIFT from VLFeat library [2][3].

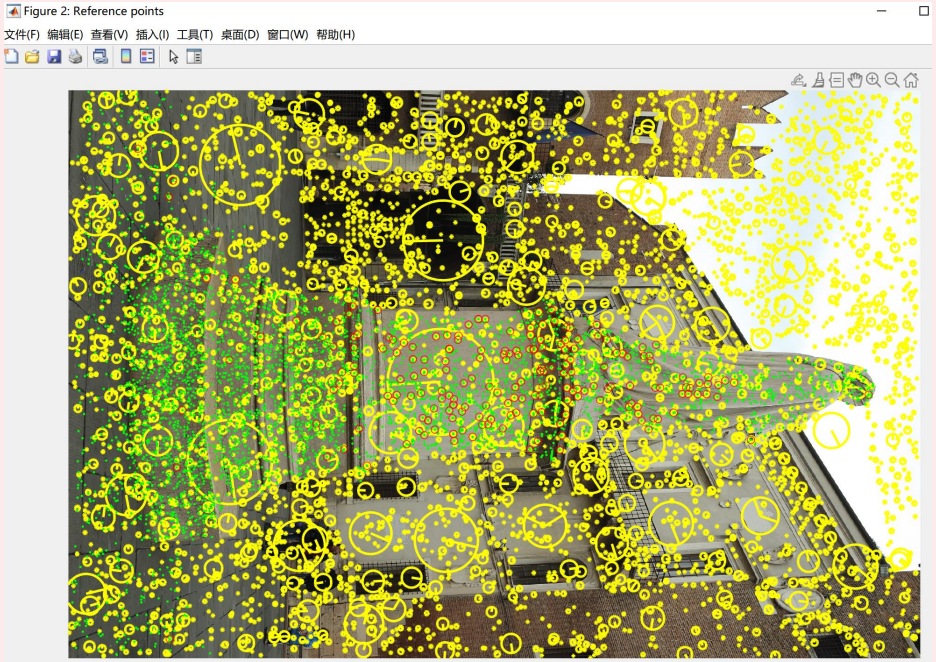



**Figure 3.** *Selected 2D feature points(yellow), projected 3D points(green), and their matches(red) as the reference points*

# 2. Design

**Project 3D points to reference image,**
done by the proj() function

# 2. Design

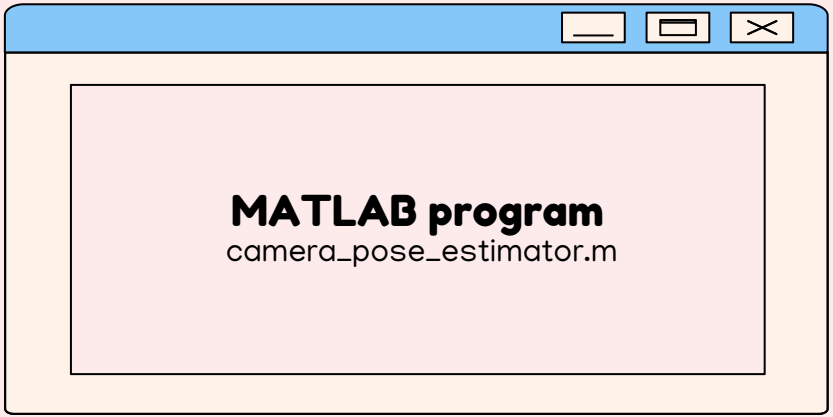
## Match largest 2D feature points with projected points,

take the matches as reference points

```
% 3. Match the largest 2D feature points with the 3D points, and take the
% matches as reference points
window_size=3; % change the window size if necessary
ind_match_3D = [];
ind_match_feature = [];
% search around the feature points, find the cooresponding projected 3D points
for i=1:size(frame_ref,2) % for every feature point
    for j =1:size(u) % for every projected u
        %  if u is within x +/- window_size
        if frame_ref(1,i) >= round(u(j))-window_size && frame_ref(1,i) <= round(u(j))+ window_size
            % if v is within y +/- window_size
            if frame_ref(2,i) >= round(v(j))-window_size && frame_ref(2,i) <= round(v(j))+ window_size
                % record the matches
                ind_match_3D = [ind_match_3D j];
                ind_match_feature = [ind_match_feature i];
            end
        end
    end
end
% show the reference points as red circles
plot(u(ind_match_3D(1,:)), v(ind_match_3D(1,:)), 'ro');

ref_feature = frame_ref(:,ind_match_feature(1,:));
ref_p3D = point_3D(ind_match_3D(1,:), :);
% assign to reference points the 2D descriptor inherited from the 2D feature points
ref_descriptor = descriptor_ref(:,ind_match_feature(1,:));

% save the ref_feature, ref_p3D, ref_descriptor, num_feature to reference_points.mat
% to be used by camera_pose_estimator.m
save('reference_points.mat', "ref_p3D", "ref_feature", "ref_descriptor", "num_feature");
```

**Figure 4.** *Implementation of matching the largest 2D feature points with the projected 3D points*

# 2. Design



# 2. Design

## Extract 2D feature points from target image

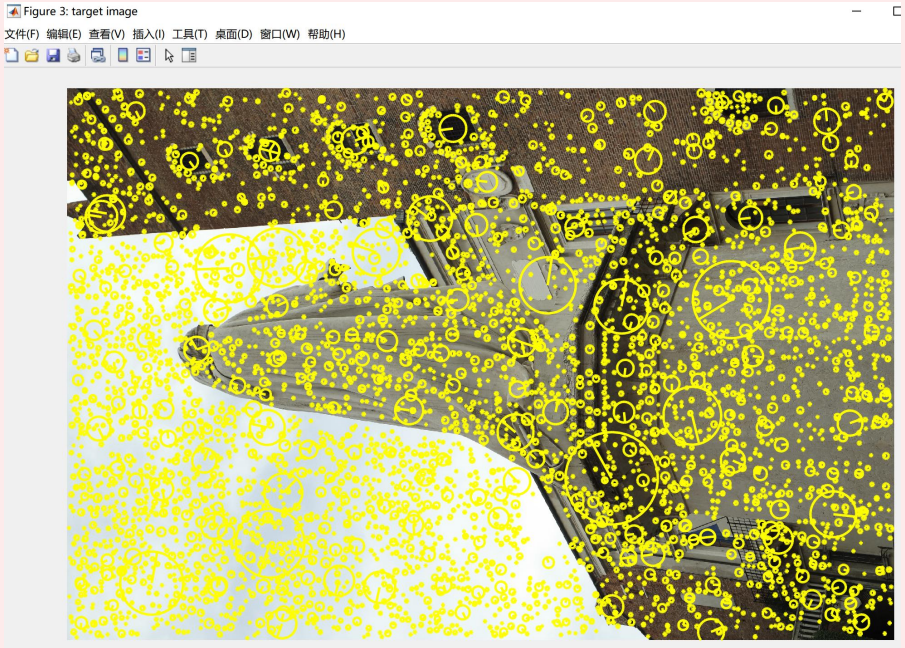



**Figure 5.** *Selected 2D feature points of the target image*

# 2. Design

## Match 2D feature points of target image with reference points,

by comparing their descriptors

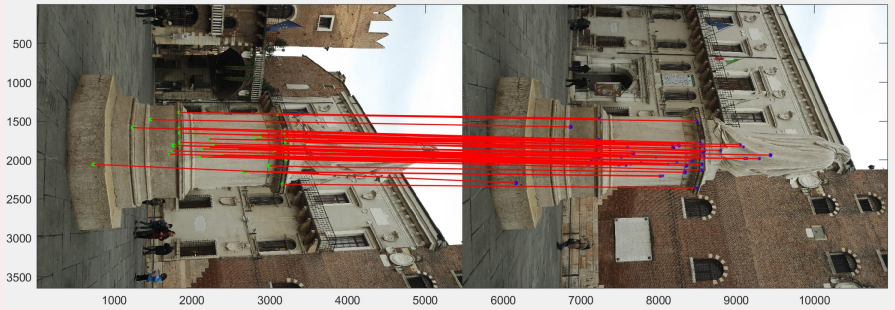
```
% 2. Match the 2D feature points of the target image with the reference points,
% by comparing their descriptors
[matches, scores] = vl_ubcmatch(ref_descriptor, descriptor_target) ;
% sort the scores and take the best matches
[dump,scoreind] = sort(scores,'ascend');
score_ratio = 0.5; % change the score_ratio if necessary
best_matchesind = scoreind(1,1:round(size(scoreind,2)*score_ratio));
% obtain 3D-->2D correspondences
best_match_3D = ref_p3D(matches(1,best_matchesind),:);
best_match_2D = frame_target(:,matches(2,best_matchesind));

% show the best matches
% combine reference image and target image
newfig=zeros(size(img_ref,1), size(img_ref,2)+size(img_target,2),3);
newfig(:,1:size(img_ref,2),:) = img_ref_RGB;
newfig(1:size(img_target,1) ,(size(img_ref,2)+1):end,:)=img_target_RGB;
newfig=uint8(newfig);
figure('Name','Best matches');
image(newfig);
axis image;
hold on
% plot the best matches
f_targ_shifted = frame_target;
f_targ_shifted(1,:) = f_targ_shifted(1,:)+size(img_ref,2); % shift the target image
h1 = vl_plotframe(ref_feature(:,matches(1,best_matchesind)));
h2 = vl_plotframe(f_targ_shifted(:,matches(2,best_matchesind)));
set(h1,'color','g','linewidth',2);
set(h2,'color','b','linewidth',2);
hold on
% plot lines between the best matches
for i = 1:size(best_matchesind,2)
    indx = best_matchesind(i);
    line([ref_feature(1,matches(1,indx)) f_targ_shifted(1,matches(2,indx))],...
    [ref_feature(2,matches(1,indx)) f_targ_shifted(2,matches(2,indx))], 'linewidth',1, 'color','r')
end
```

**Figure 6.** *Implementation of matching the 2D feature points of the target image with the reference points*

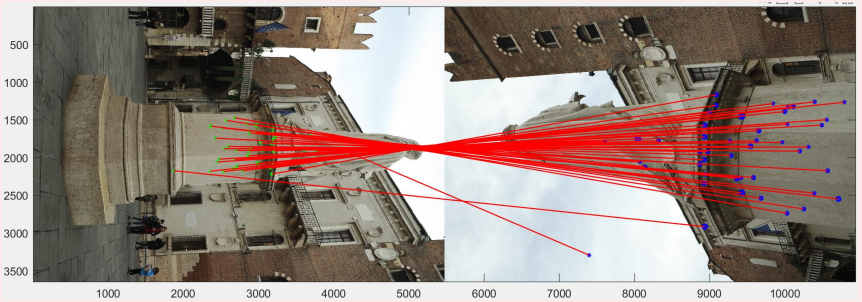
# 2. Design

## Match 2D feature points of target image with reference points,

by comparing their descriptors




(a)




(b)

**Figure 7.** *Best matches between the feature points of the target image and the reference*
(a)no outliers, (b) occurrence of outliers

# 2. Design

## Camera pose estimation using Fiore's exterior estimation & RANSAC fitting[4]

```
% 3. Estimate the camera pose of the target image using Fiore's exterior
% estimation method, which takes 3D-->2D correspondences as input. Robust fitting using RANSAC.
% read the camera parameters
name_targ_xmp = [name_target(1:end-3) 'xmp'];
[K1, R1, t1] = read_xmp(name_targ_xmp);
% the known camera extrinsic parameter and ppm are used for later comparing
% with the estimated result.
G1 = [R1 t1; 0 0 0 1];
ppm_known = K1 * [1 0 0 0
        0 1 0 0
        0 0 1 0] * G1;

% run Fiore's method and robust fitting using RANSAC, obtain the estimated camera pose
tollerance = 10; % acceptable distance used to filter out the outliers
max_iterations = 100; % change the RANSAC iteration times if needed
[G_estimated, inliers, outliers] = ransac_fiore(K1, best_match_3D', best_match_2D(1:2,:), tollerance, max_iterations);
P_estimated=K1*G_estimated;

% project the 3D points to the image using the estimated ppm (red circle)
% and compare with the projection using the known ppm (blue dot)
figure('Name','projection of 3D points using estimated ppm "red o" and known ppm "blue ."');
[u2,v2] = proj(ppm_known,point_3D);
[u3,v3] = proj(P_estimated,point_3D);
imshow(img_target_RGB);
hold on
plot(u2,v2,'b.');
plot(u3,v3,'ro')
```

(a)

**Figure 8.** *(a) Implementation of estimating the camera pose of the target image (b) Implementation of the ransac_fiore() function*

# 2. Design

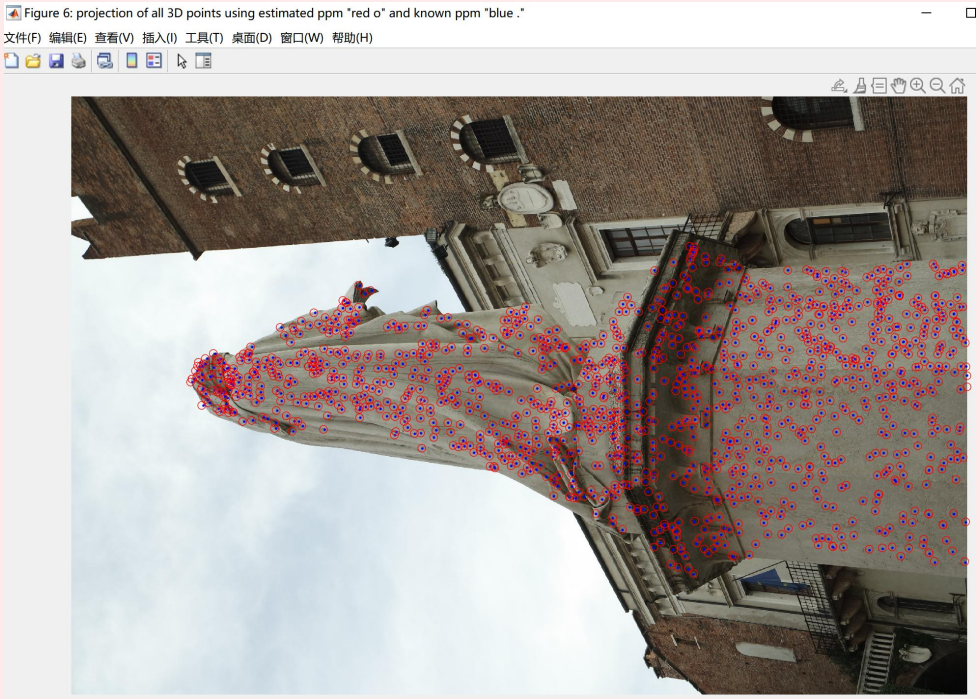
## Camera pose estimation using Fiore's exterior estimation & RANSAC fitting[4]

```
function [G, inliers, outliers] = ransac_fiore(K, points_3D, points_2D, tollerance, max_iterations)

inliers = [];
outliers = [];
p_model = 6; % minimum number of points needed for Fiore's exterior estimation

i = 0;
while (i < max_iterations) % iterate max_iterations times
    inliers_temp = [];
    outliers_temp = [];
    % make the data to be random
    rand_ind = randperm(size(points_3D,2));
    points_3D_rand = points_3D(:,rand_ind);
    points_2D_rand = points_2D(:,rand_ind);

    % run Fiore's exterior estimation, obtain the model parameters
    [G_temp,s] = exterior_fiore(K,points_3D_rand(:,1:p_model),points_2D_rand(:,1:p_model));

    % project the rest of the 3D points to the image using the estimated model parameters
    p_3D = points_3D_rand(:,p_model+1:end);
    ppm_temp=K*G_temp;
    [u,v] = proj(ppm_temp,p_3D');
    projected_3D = [u';v'];
    % take the rest of the 2D points
    p_2D = points_2D_rand(:,p_model+1:end);


    % model fitting using the rest of the data
    % filter out the outliers using the distance between the 2D points and projected 3D points
    for j = 1:size(projected_3D,2)
        distance = norm(p_2D(:,j)-projected_3D(:,j));
        if (abs(distance) <= tollerance)
            inlier = [p_3D(:,j); p_2D(:,j)];
            inliers_temp = [inliers_temp inlier];
        else
            outlier = [p_3D(:,j); p_2D(:,j)];
            outliers_temp = [outliers_temp outlier];
        end
    end
    % update the output if there is a better camera pose estimation
    % according to the number of inliers
    if (size(inliers_temp, 2) > size(inliers, 2))
        inliers = inliers_temp;
        outliers = outliers_temp;
        G = G_temp;
    end
    i = i + 1;
end

% run Fiore's exterior estimation, obtain the camera pose using all inliers from the best
% model parameters estimation
[G,s] = exterior_fiore(K,inliers(1:3,:),inliers(4:5,:));
```

**Figure 8.** *(a) Implementation of estimating the camera pose of the target image (b) Implementation of the ransac_fiore() function*

# 2. Design

## Estimate camera pose using Fiore's exterior estimation




(a)

```
known camera pose:
ans =
    0.4825   -0.1275   -0.8665   64.8652
    0.1719    0.9839   -0.0490  -60.6174
    0.8588   -0.1253    0.4967  -29.7142
estimated camera pose:
G_estimated =
    0.4773   -0.1245   -0.8699   64.9583
    0.1680    0.9846   -0.0488  -60.6419
    0.8625   -0.1228    0.4909  -29.4892
```

(b)

**Figure 9.** (a)*Projection of the 3D points using estimated ppm(red) and known ppm(blue)* (b)*Printed estimated camera pose and known camera pose*

# 3. Testing and Validation

## Experiment 1 - Dante Statue

### Reference

Image size 3648*5472

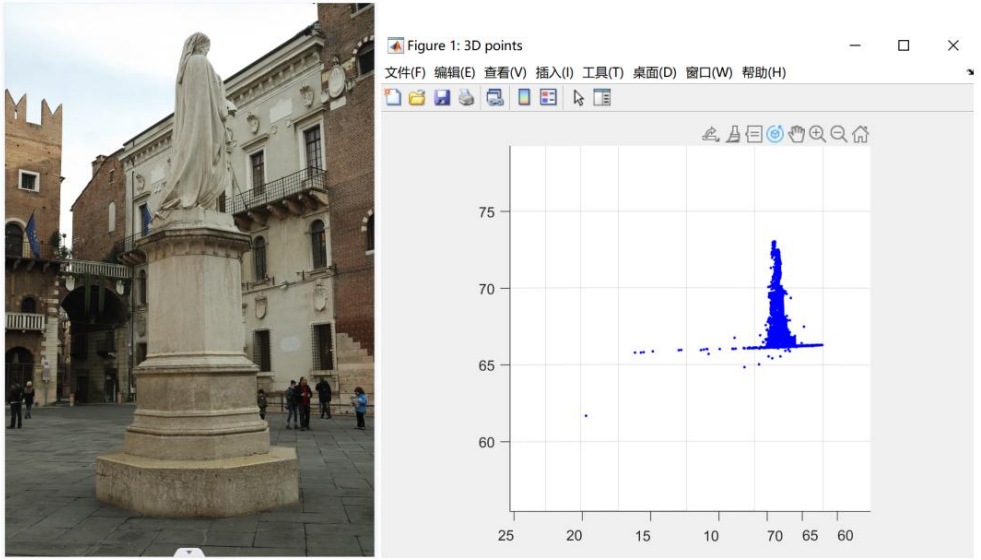



**Figure 10.** *Reference image _SAM1001.JPG and corresponding 3D points*

# 3. Testing and Validation

## Experiment 1 - Dante Statue

### Target

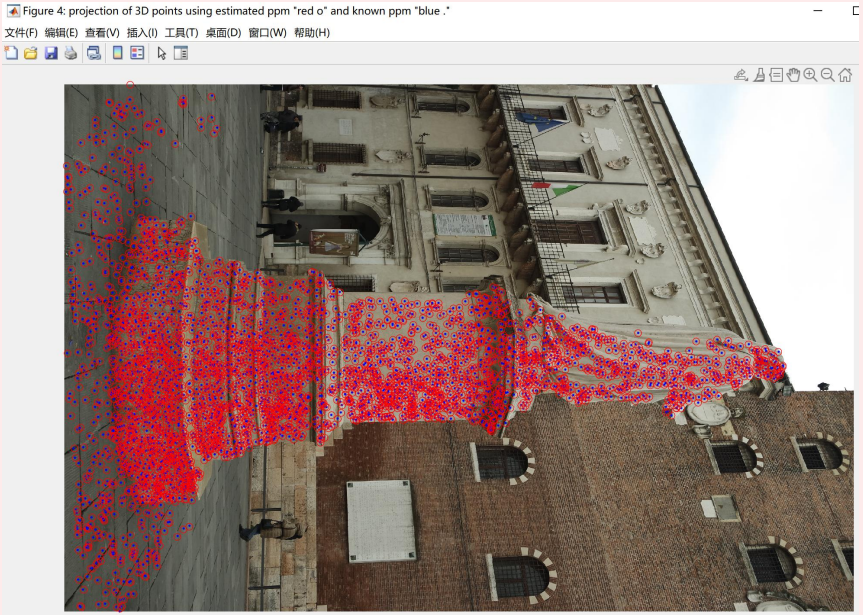
_SAM1002.JPG _SAM1079.JPG _SAM1080.JPG

_SAM1107.JPG _SAM1108.JPG

**Figure 11.** *Target images*

# 3. Testing and Validation

## Experiment 1 - Dante Statue



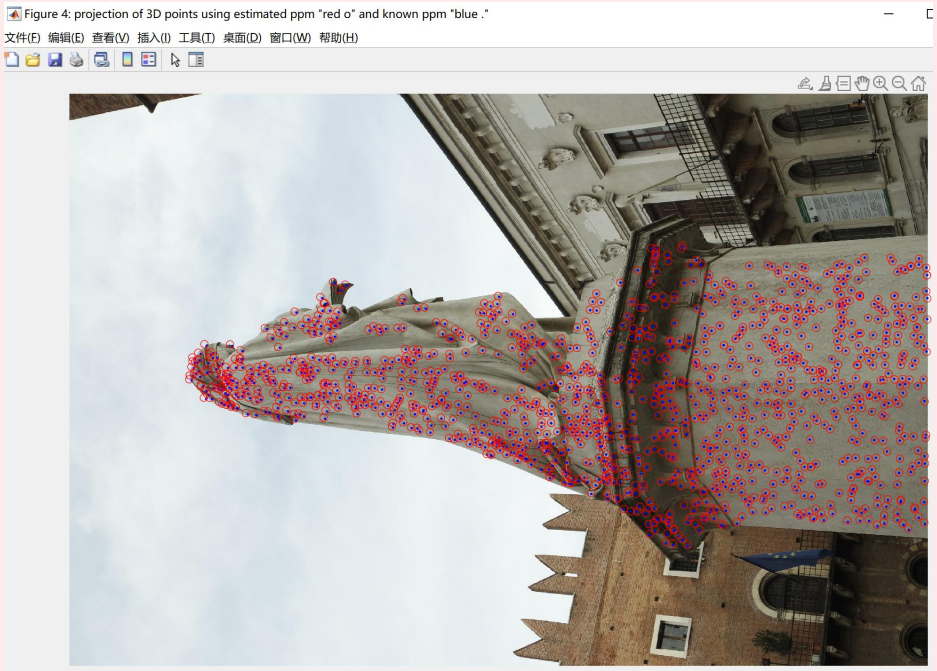

_SAM1002.JPG

## Result

```
known camera pose:

ans =

   -0.1579    0.0831    0.9840  -72.1277
   -0.3066   -0.9513    0.0311   61.2189
    0.9386   -0.2968    0.1757    6.6508

estimated camera pose:

G_estimated =

   -0.1599    0.0828    0.9837  -72.0699
   -0.3046   -0.9520    0.0306   61.2794
    0.9390   -0.2947    0.1774    6.4004
```

# 3. Testing and Validation

## Experiment 1 - Dante Statue



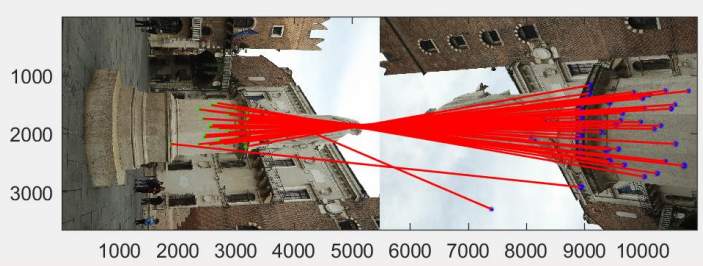

_SAM1079.JPG

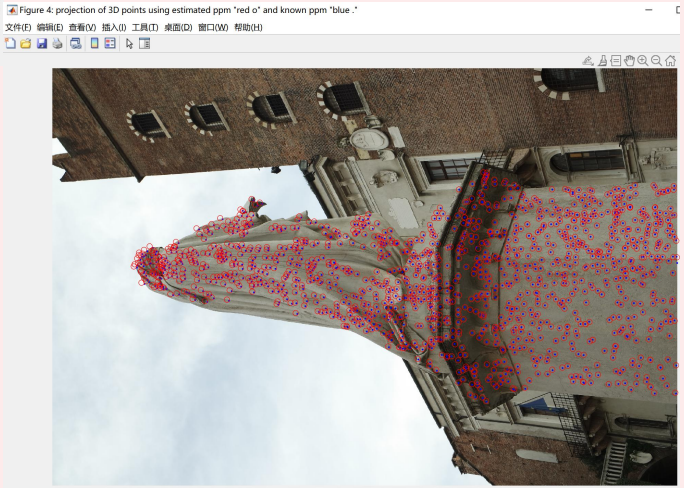
## Result

```
known camera pose:

ans =

    0.4922    0.0141   -0.8704   56.0879
   -0.0736    0.9970   -0.0255  -60.8032
    0.8674    0.0766    0.4917  -42.2190

estimated camera pose:

G_estimated =

    0.4853    0.0176   -0.8742   56.1944
   -0.0836    0.9962   -0.0264  -60.5997
    0.8703    0.0859    0.4849  -42.3515
```

# 3. Testing and Validation

## Experiment 1 - Dante Statue




## Result

known camera pose:

ans =

| | | | |
|---|---|---|---|
| 0.4825 | -0.1275 | -0.8665 | 64.8652 |
| 0.1719 | 0.9839 | -0.0490 | -60.6174 |
| 0.8588 | -0.1253 | 0.4967 | -29.7142 |

estimated camera pose:

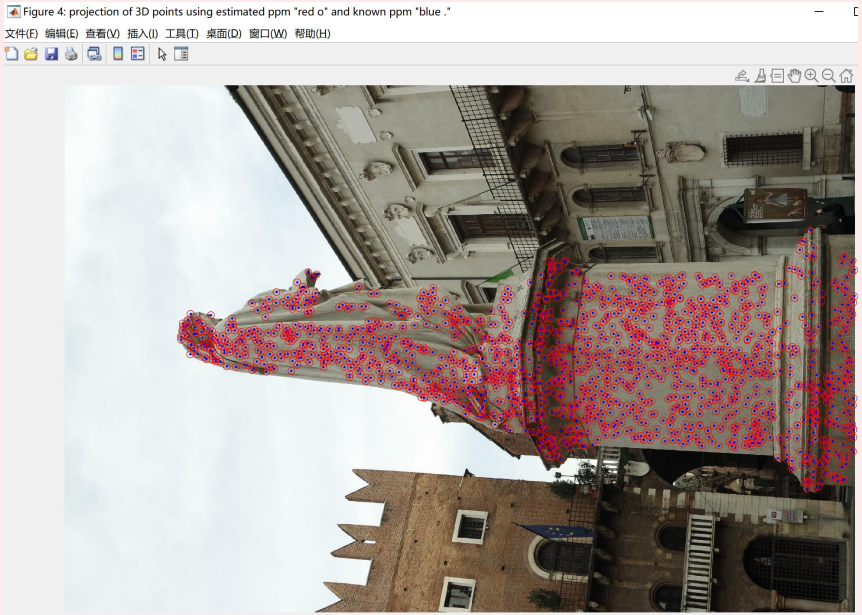
G_estimated =

| | | | |
|---|---|---|---|
| 0.4734 | -0.1242 | -0.8720 | 65.1194 |
| 0.1642 | 0.9851 | -0.0512 | -60.4777 |
| 0.8654 | -0.1190 | 0.4868 | -29.4799 |

_SAM1080.JPG

# 3. Testing and Validation

## Experiment 1 - Dante Statue



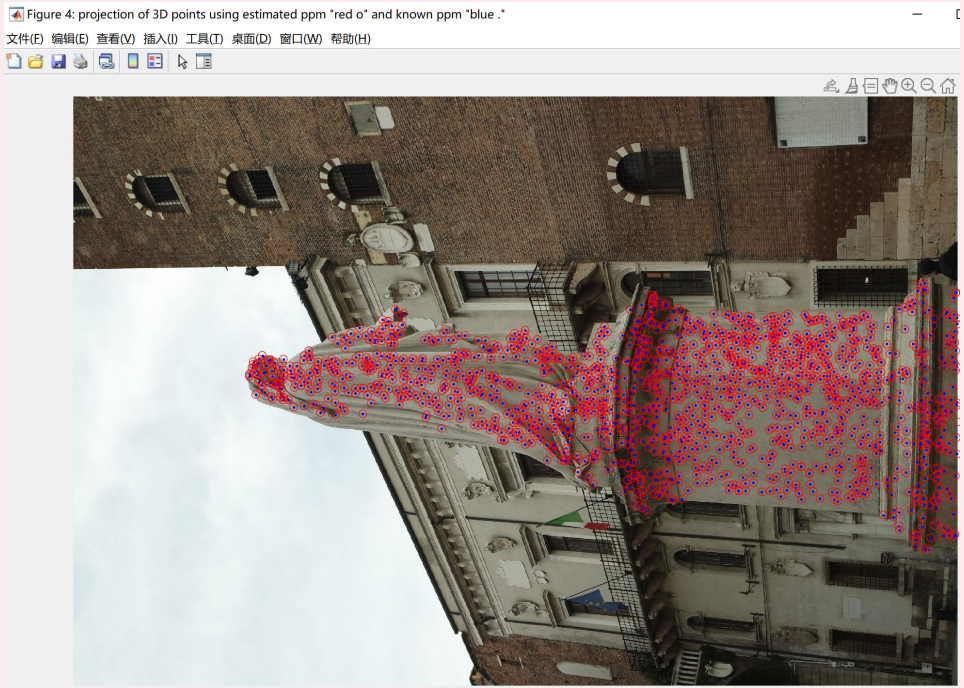

_SAM1107.JPG

## Result

```
known camera pose:

ans =

    0.3290   -0.0343   -0.9437   65.6387
   -0.0368    0.9981   -0.0491  -59.5786
    0.9436    0.0508    0.3272  -27.8686

estimated camera pose:

G_estimated =

    0.3253   -0.0331   -0.9450   65.6927
   -0.0407    0.9980   -0.0490  -59.5449
    0.9448    0.0543    0.3232  -27.8155
```

# 3. Testing and Validation

## Experiment 1 - Dante Statue




### Result

```
known camera pose:

ans =

    0.3198   -0.1355   -0.9377   71.9663
    0.2218    0.9729   -0.0650  -59.2561
    0.9212   -0.1872    0.3412  -13.1343

estimated camera pose:

G_estimated =

    0.3160   -0.1317   -0.9396   71.8874
    0.2128    0.9749   -0.0651  -59.2961
    0.9246   -0.1794    0.3361  -13.2917
```

_SAM1108.JPG **Figure 12.** *Projection of the 3D points using estimated ppm(red) and known ppm(blue), printed estimated camera pose and known camera pose*

# 3. Testing and Validation

## Experiment 2 - Ca' Vignal 2 building

### Reference

Image size 2448*3264

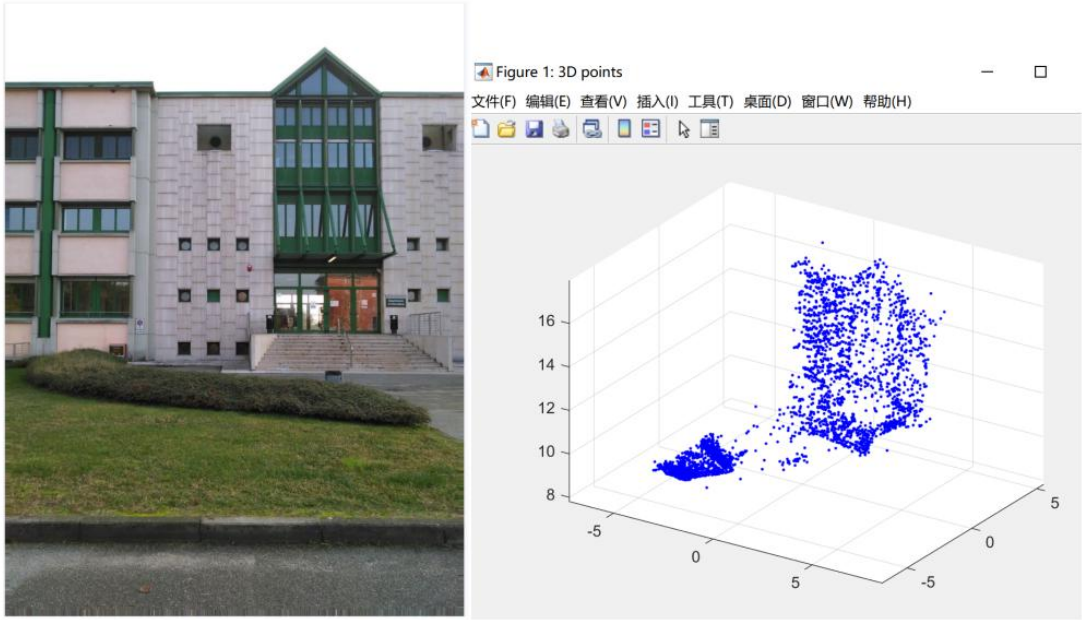
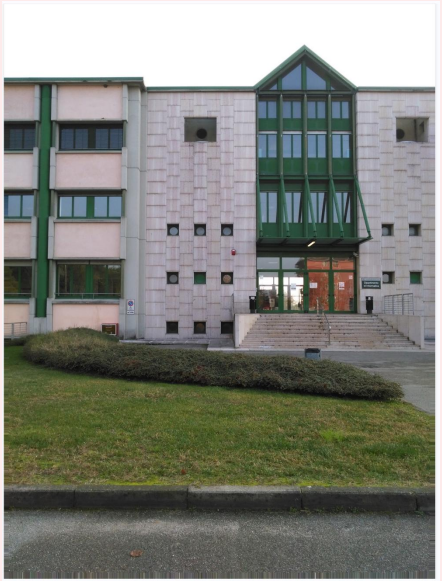



**Figure 13.** *Image cav.jpg and corresponding 3D points*

# 3. Testing and Validation

## Experiment 2 - Ca' Vignal 2 building

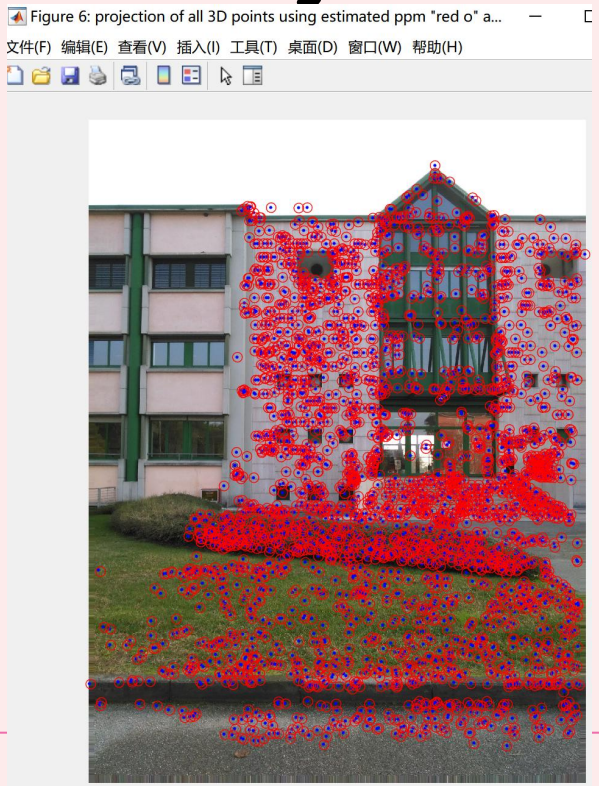
Target



cav.jpg

# 3. Testing and Validation

## Experiment 2 - Ca' Vignal 2 building




### Result

known camera pose:

ans =

| | | | |
|---|---|---|---|
| 0.9871 | -0.1603 | -0.0012 | 1.7549 |
| 0.0014 | 0.0161 | -0.9999 | 10.7369 |
| 0.1603 | 0.9869 | 0.0161 | 7.8101 |

estimated camera pose:

G_estimated =

| | | | |
|---|---|---|---|
| 0.9871 | -0.1601 | -0.0010 | 1.7557 |
| 0.0016 | 0.0158 | -0.9999 | 10.7408 |
| 0.1601 | 0.9870 | 0.0159 | 7.8192 |

**Figure 14.** *Projection of the 3D points using estimated ppm(red) and known ppm(blue), printed estimated camera pose and known camera pose*

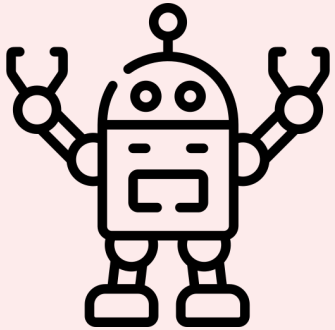
# 4. Conclusions And Future Perspective

## Conclusions

- Estimate 3D –> 2D correspondences
- Fiore's exterior orientation estimation
- Performs well on images with different sizes

## Future Work

- Real-time robot pose estimation
- Camera is fixed on robot
- Video stream as input



# References

[1]P. D. Fiore, "Efficient linear solution of exterior orientation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 2, pp. 140–148, Feb. 2001, doi: 10.1109/34.908965.
[2] https://www.vlfeat.org/overview/sift.html#tut.sift.match
[3] https://www.vlfeat.org/api/sift.html
[4] M. A. Fischler, R. C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, Readings in Computer Vision, 1987, Pages 726–740, https://doi.org/10.1016/B978-0-08-051581-6.50070-2.

# THANKS