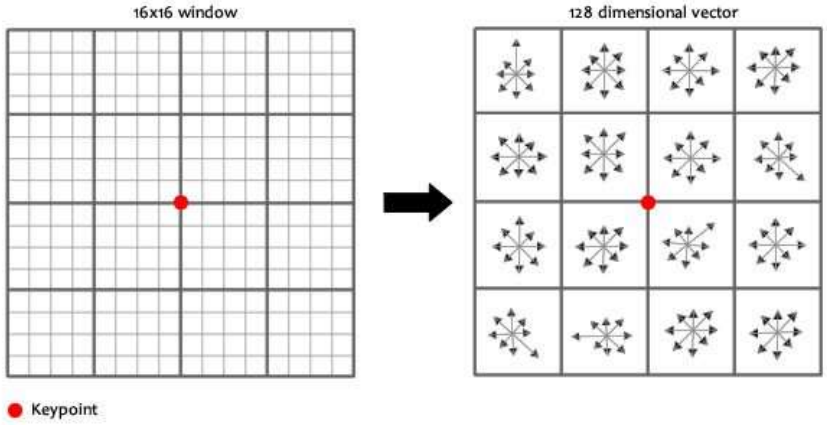
컴퓨터 비전 세미나

12191584 김창수

SIFT descriptor

For each sub-block, 8 bin orientation histogram is created.



In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation.

These 4 x 4 x 8 = 128 numbers form the "feature vector".

Keypoint is uniquely identified by this feature vector $\mathbf{x}$. $(x,\ y,\ \sigma,\ \theta_i,\ \mathbf{x}_i)$

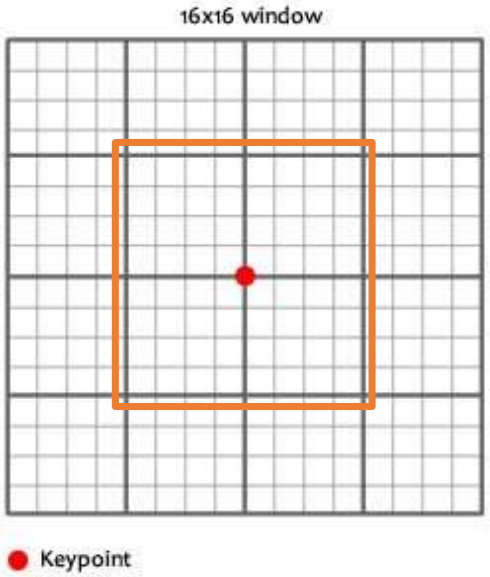
Orientation histogram

```python
num_bins = 8
raw_histogram = np.zeros(num_bins)

def orientations(image, x, y):
    keypoints_with_orientations = []

    for i in range(-4, 5):
        region_y = y + i
        if region_y > 0 and region_y < image.shape[0] - 1:

            for j in range(-4, 5):
                region_x = x + j
                if region_x > 0 and region_x < image.shape[1] - 1:
```

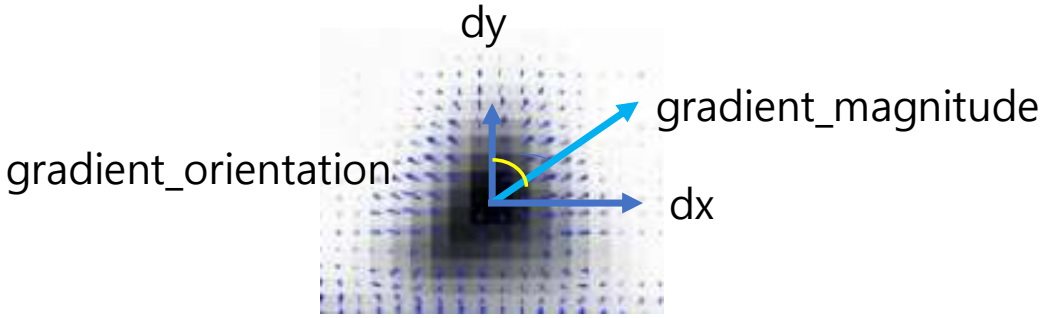


Orientation histogram

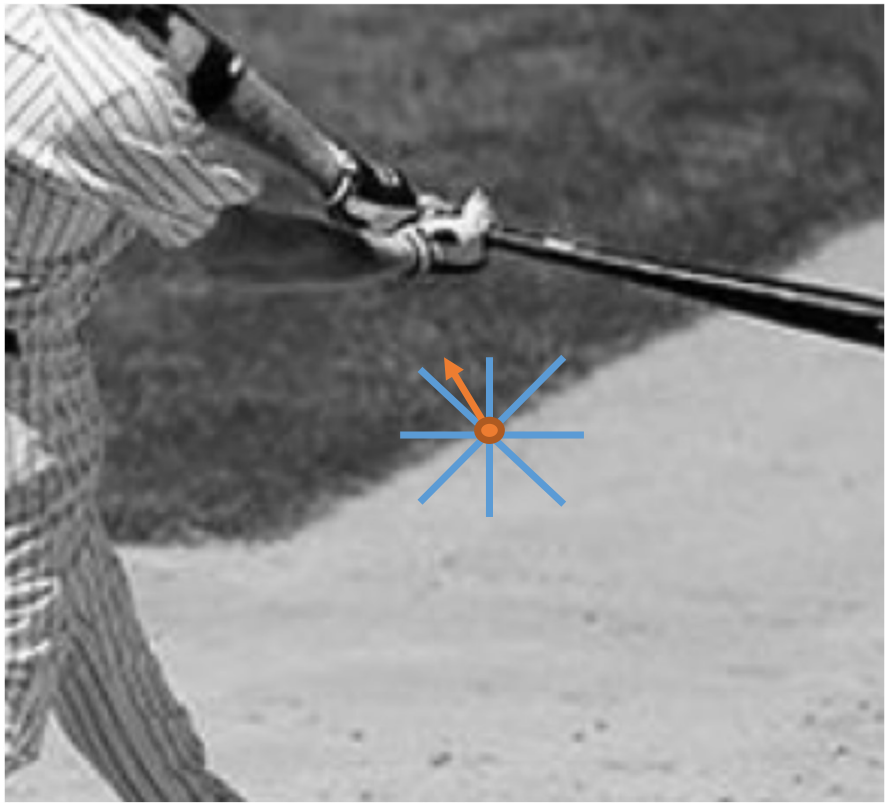
```
14            dx = image[region_y, region_x + 1] - image[region_y, region_x - 1]
15            dy = image[region_y - 1, region_x] - image[region_y + 1, region_x]
16            gradient_magnitude = np.sqrt(dx * dx + dy * dy)
17            gradient_orientation = np.rad2deg(np.arctan2(dy, dx))
18
19            histogram_index = int(round(gradient_orientation * num_bins / 360.))
20            raw_histogram[histogram_index % num_bins] += gradient_magnitude
```

$$Edge\_Gradient\,(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle(\theta) = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

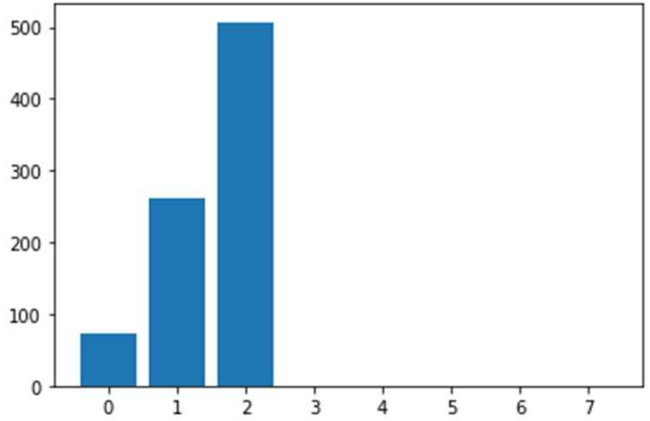


Orientation histogram



```
[ 72.72265625 260.484375   507.0234375    0.           0.
   0.           0.           0.         ]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: I

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: I
  from ipykernel import kernelapp as app
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: I
  app.launch_new_instance()
<BarContainer object of 8 artists>
```



Index

Image stitching
- Panoramas

Segmentation
-Graph cut and Mean shift

Image Stitching

Once we have extracted features from images, the next stage in many vision algorithms is to match these features across different images.

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points.

Transformation

Given a set of matched feature points $\{(x_i,\ x'_i)\}$

and a planar parametric transformation $\mathbf{x}' = \mathbf{f}(\mathbf{x}; \mathbf{p})$

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$

Least squares

The usual way estimate the motion parameters is to use least squares, to minimize the sum of squared residuals.

$$E_{\mathrm{LS}} = \sum_i \|\mathbf{r}_i\|^2 = \sum_i \|\mathbf{f}(\mathbf{x}_i; \mathbf{p}) - \mathbf{x}'_i\|^2$$

$\boldsymbol{r}$ is the residual between the measured location and its current predicted location

translation, similarity, and affine, have a linear relationship

between the amount of motion and the unknown parameters

$$\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} = \mathbf{J}(\mathbf{x})\mathbf{p}$$

where $\boldsymbol{J}(\mathbf{x}) = \partial\mathbf{f}/\partial\mathbf{p}$ is the Jacobian of the transformation f

Least squares

In this case, a simple linear regression can be formulated as

$$
\begin{aligned}
E_{\mathrm{LLS}} &= \sum_i \|\mathbf{J}(\mathbf{x}_i)\mathbf{p} - \Delta\mathbf{x}_i\|^2 \\
&= \mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i)\right] \mathbf{p} - 2\mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i\right] + \sum_i \|\Delta\mathbf{x}_i\|^2 \\
&= \mathbf{p}^T\mathbf{A}\mathbf{p} - 2\mathbf{p}^T\mathbf{b} + c.
\end{aligned}
$$

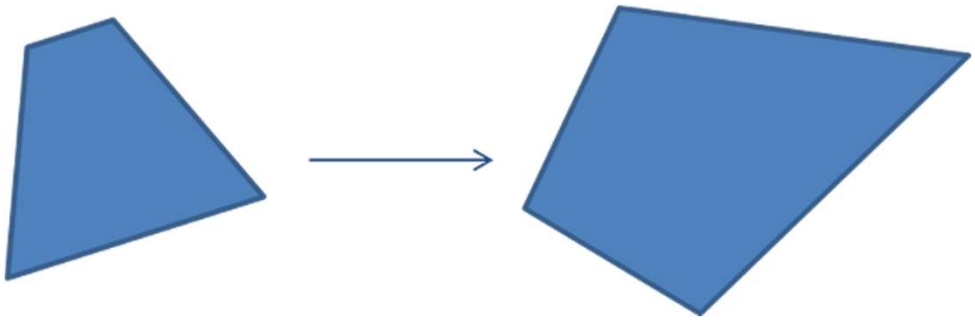
The minimum can be found by solving the $\mathbf{A}\mathbf{p} = \mathbf{b}$

$$
\mathbf{A} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i) \qquad \mathbf{b} = \textstyle\sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i
$$

The translation is the average translation between corresponding points

Homography

The derivatives are all fairly straightforward, except for the motion homography, which arises in image-stitching applications.



$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$$

$$\begin{bmatrix} ax_i' \\ ay_i' \\ a \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

Homography

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix}}_{\mathbf{A}_{2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\mathbf{h}_{9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{0}_{2n}}$$

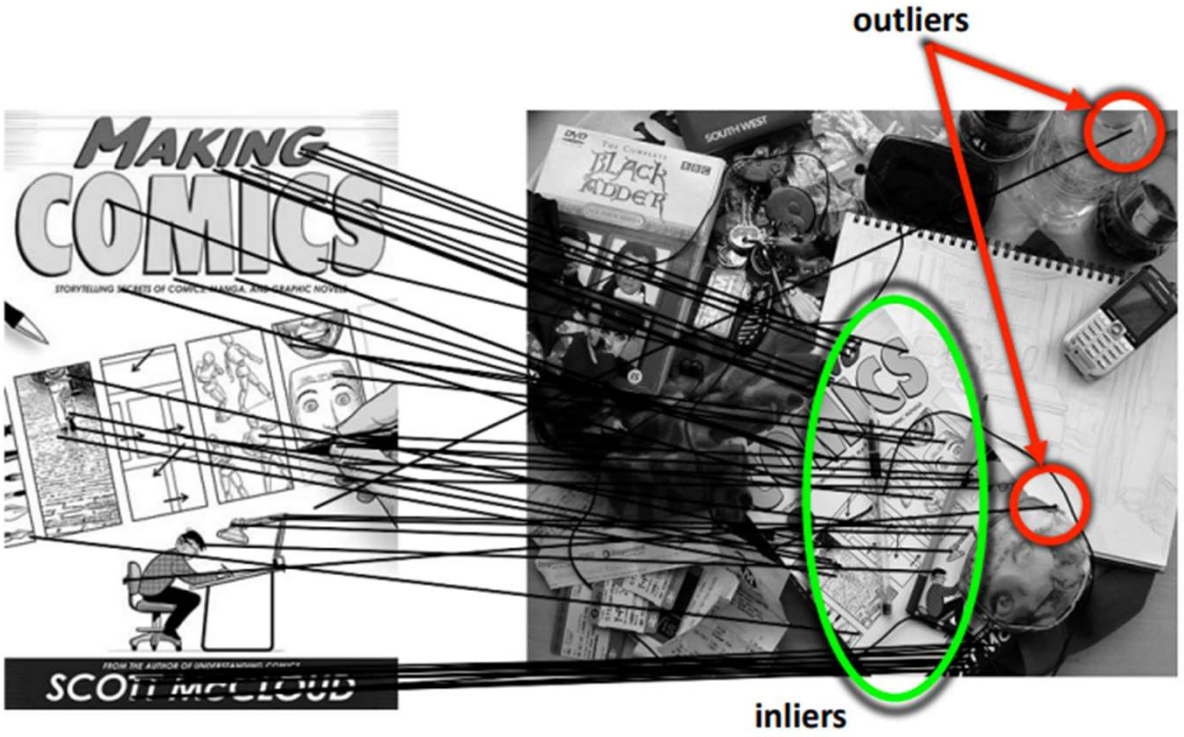
$\mathbf{Ah} = 0$ is not possible, so find $min_h \|\mathbf{Ah}\|$

Works with 4 or more points.

Solution: h = eigenvector of $\mathbf{A^T A}$ with smallest eigenvalue

RANSAC

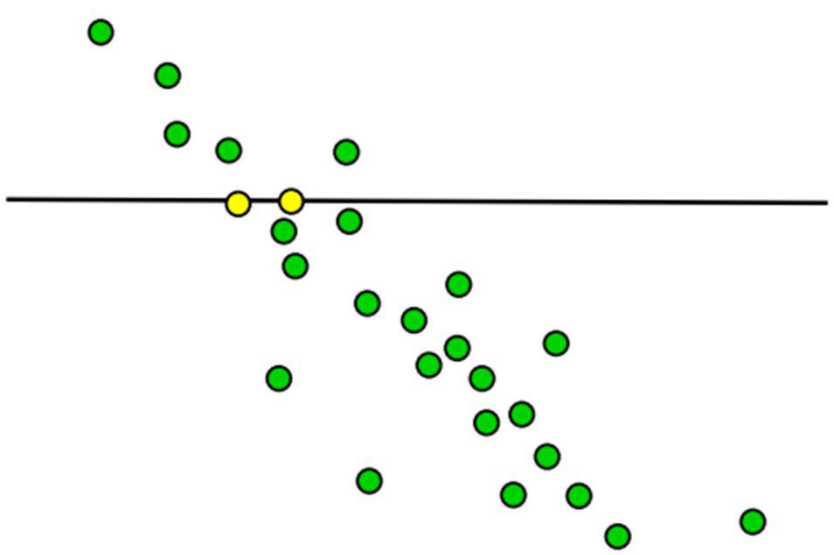
While regular least squares is the method of choice for measurements where the noise follows a normal distribution, more robust versions of least squares are required when there are outliers among the correspondences

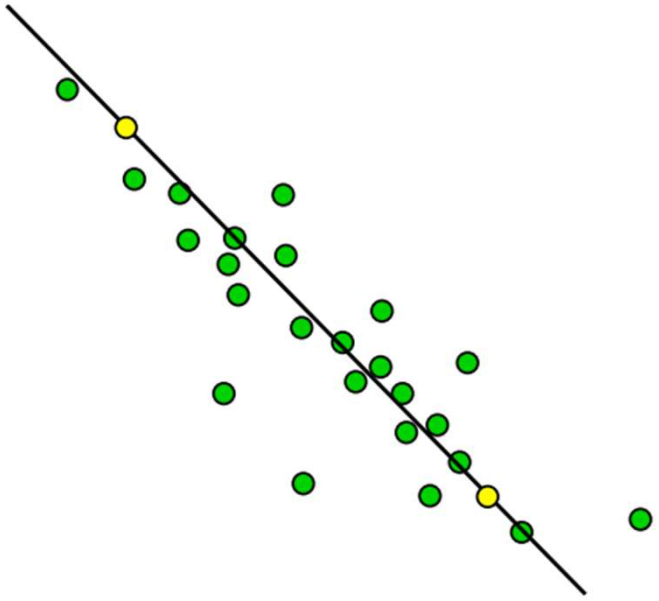


RANSAC

1. Select randomly the minimum number of points(n) required to determine the model parameters.

   And solve for the parameters of the model.

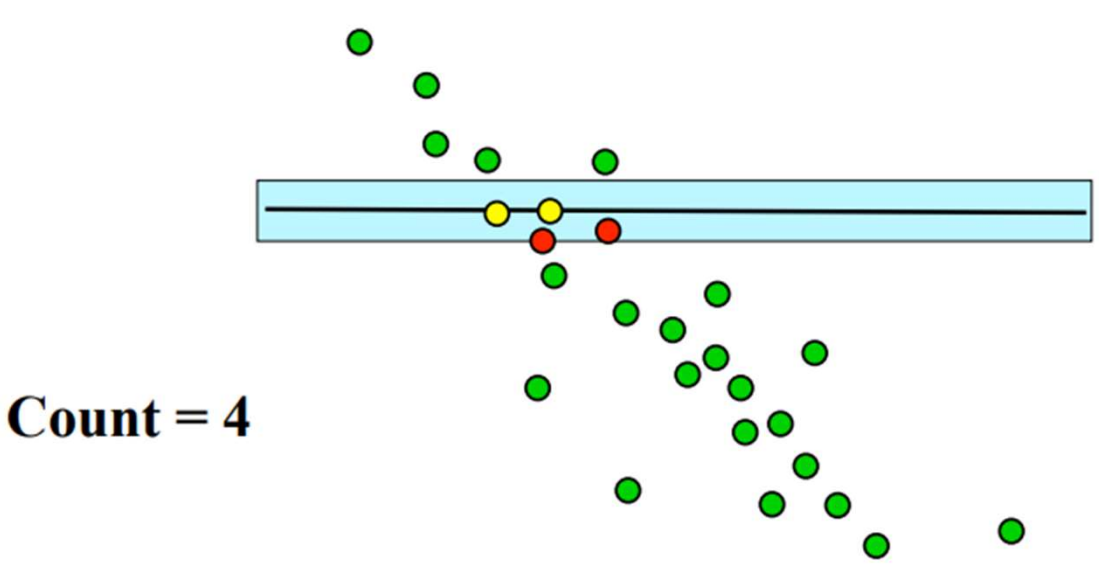




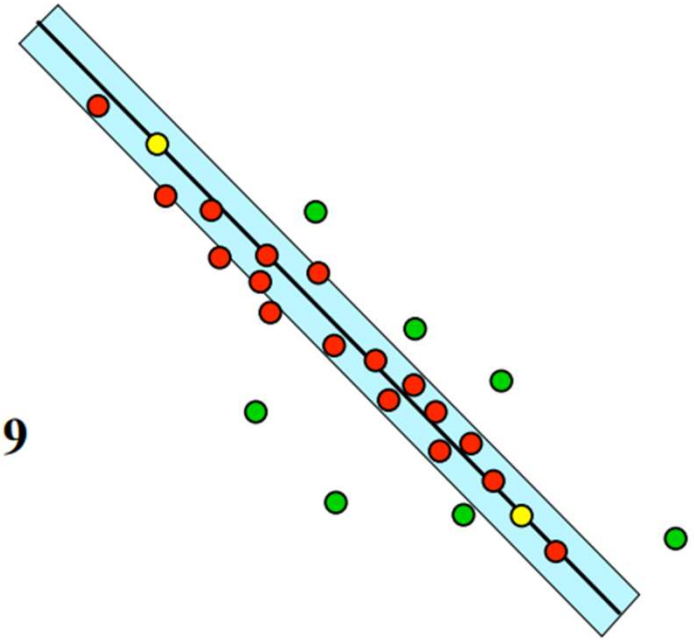
n = 2

RANSAC

2. Determine how many points from the set of all points fit with a predefined tolerance(ε)



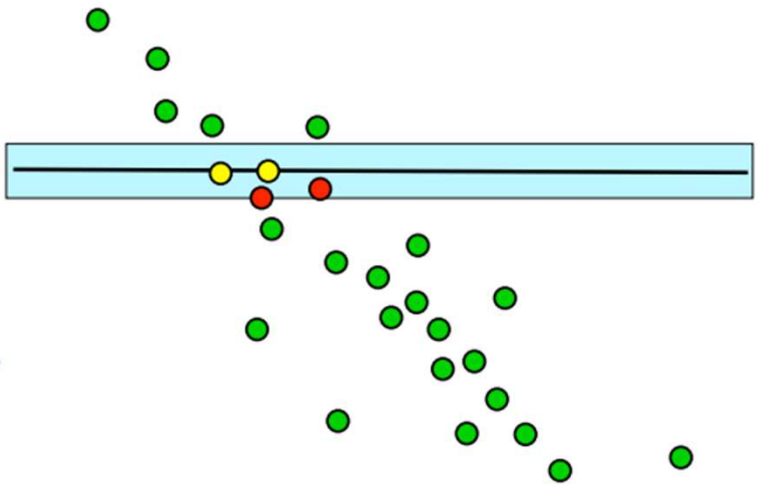
Count = 4



Count = 19

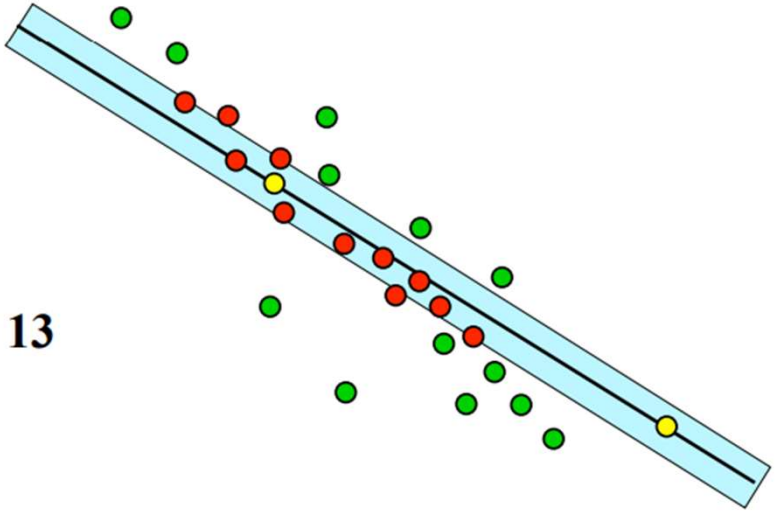
RANSAC

3. If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold(τ), save the model.



Count = 4

4 / 26



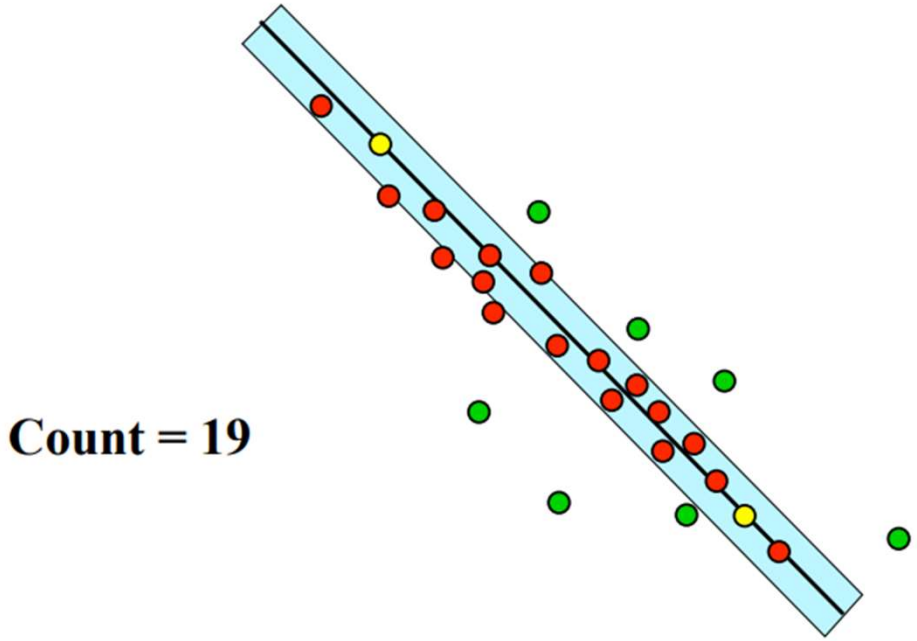
Count = 13

13 / 26

RANSAC

4. Repeat steps 1 through 4 (maximum of k times)

5. re-estimate the model parameters using all the identified inliers and terminate.



RANSAC

The number of iterations, $\boldsymbol{k}$, is chosen high enough to ensure the probability $\boldsymbol{p}$ that at least **one of the sets** of random samples which **all are inliers**.

Let $\boldsymbol{w}$ be the probability of **choosing an inlier.**

$\boldsymbol{1 - w^n}$ is the probability that **at least one of the $\boldsymbol{n}$ points is an outlier.**

That probability to the power of $\boldsymbol{k}$ is the probability $\boldsymbol{1 - p}$

that the algorithm **never selects a set** of n points which **all are inliers**.

$$1 - p = (1 - w^n)^k$$

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

RANSAC

n	w	k
3	0.5	35
6	0.6	97
6	0.5	293

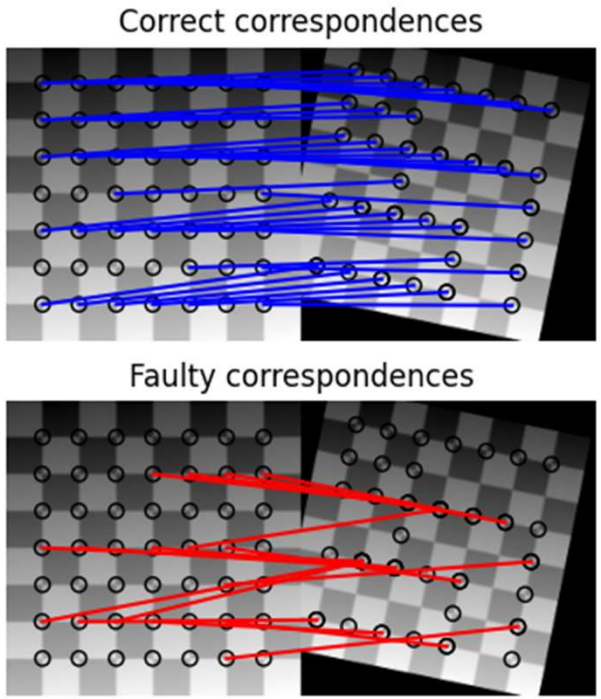
The number of trials grows quickly with the number of sample points used.

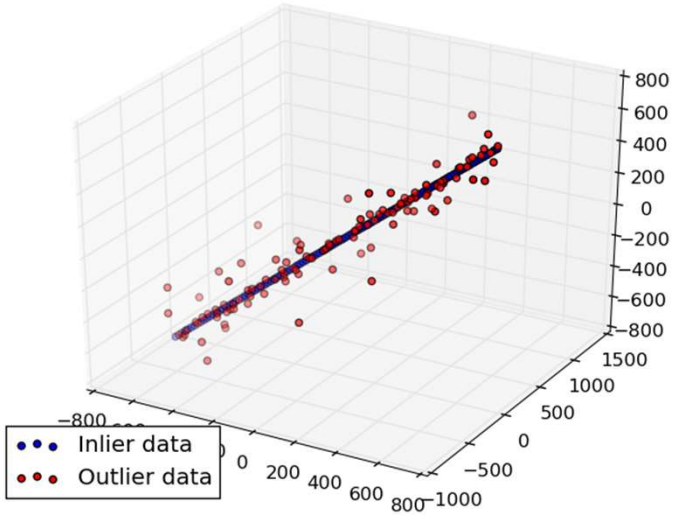
This provides a strong incentive to use the minimum number of sample points n

RANSAC can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set.

RANSAC can only estimate one model for a particular data set.

RANSAC





RANSAC robustly estimate the parameter set

Compositing

If only a few images are stitched together:

- select one of the images as the reference.
- warp all of the other images into its reference coordinate system.

```python
sift = cv2.xfeatures2d.SIFT_create()

# finding the keypoints and descriptors in the images using SIFT
keyPoints1, descriptors1 = sift.detectAndCompute(img1, None)
keyPoints2, descriptors2 = sift.detectAndCompute(img2, None)
print('img1 - %d features, img2 - %d features' % (len(keyPoints1), len(keyPoints2)))


# saving the coordinates of a keypoints
keyPoints1 = np.float32([keypoint.pt for keypoint in keyPoints1])
keyPoints2 = np.float32([keypoint.pt for keypoint in keyPoints2])


# matching keypoint and estimating the homography transformation
matches, H, status = matchKeypoint(keyPoints1, keyPoints2, descriptors1, descriptors2)


# drawing keypoint matches on images
img_with_matches = drawMatch(img1, img2, keyPoints1, keyPoints2, matches, status)


# warping one image to other
result = cv2.warpPerspective(img1, H, (img1.shape[1] + img2.shape[1], img1.shape[0]))
result[0:img2.shape[0], 0:img2.shape[1]] = img2

draw_result(img_with_matches, result)
```

```python
def matchKeypoint(keyPoints1, keyPoints2, descriptors1, descriptors2):
    index_params = dict(algorithm = 0, trees = 5)
    search_params = dict(checks = 50)

    # Fast Library for Approximate Nearest Neighbors.
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # get 2 best matches.
    raw_matches = flann.knnMatch(descriptors1, descriptors2, k=2)

    H, status = None, None
    matches = []
    for m in raw_matches:
        # check enough difference between the best and second-best matches.
        if len(m) == 2 and m[0].distance < m[1].distance * 0.79:
            matches.append((m[0].trainIdx, m[0].queryIdx))


    # need at least 4 points because dof is 8
    if len(matches) >= 4:
        kp1 = np.float32([ keyPoints1[i] for (_, i) in matches ]).reshape(-1,1,2)
        kp2 = np.float32([ keyPoints2[i] for (i, _) in matches ]).reshape(-1,1,2)

        # find homography using RANSAC
        H, status = cv2.findHomography(kp1, kp2, cv2.RANSAC, 5.0)

        print('%d / %d  inliers/matched' % (np.sum(status), len(status)))

    return matches, H, status
```

```python
def drawMatch(image1, image2, keyPoints1, keyPoints2, matches, status):
    h1, w1 = image1.shape[:2]
    h2, w2 = image2.shape[:2]

    # image2 -> image1
    img_with_matches = np.zeros((max(h1, h2), w1 + w2, 3), dtype="uint8")
    img_with_matches[0:h2, 0:w2] = image2
    img_with_matches[0:h1, w2:] = image1


    #
    for ((trainidx, queryidx), s) in zip(matches, status):
        if s == 1:
            keyPoint2 = (int(keyPoints2[trainidx][0]), int(keyPoints2[trainidx][1]))
            keyPoint1 = (int(keyPoints1[queryidx][0]) + w2, int(keyPoints1[queryidx][1]))
            cv2.line(img_with_matches, keyPoint1, keyPoint2, (0, 255, 0), 1)

    return img_with_matches

```

Implement image stitching

```
img1 - 5490 features, img2 - 4823 features
754 / 1013  inliers/matched
```

Implement image stitching