Image Processing and Computer Vision
Detecting primitives and rigid objects in Images

Martin de La Gorce
martin.de-la-gorce@enpc.fr

February 2015
How can we detect simple primitive such as lines or circles in an image? We can use the canny edge detector to get a list of edges points. We try to approximate this set of points by straight lines or circles. We will see two different methods to detect lines that generalize to other kind of primitives:

- Hough transform
- RANSAC
a line can be parameterized by two parameters

- its angle $\theta \in [0, \pi]$
- its signed distance $\rho \in \mathbb{R}$ to the center of the image
We denote $l(\theta, \rho) \subset \mathbb{R}^2$ the line corresponding to the parameters $(\theta, \rho)$. We have

$$l(\theta, \rho) = \{p \in \mathbb{R}^2 | \langle p, \hat{n}(\theta) \rangle = \rho\}$$

with $\hat{n}(\theta)$ the unit vector point in the direction $\theta$

$$\hat{n}(\theta) = [\cos(\theta), \sin(\theta)]$$

This rewrites

$$l(\theta, \rho) = \{p \in \mathbb{R}^2 | p_x \cos(\theta) + p_y \sin(\theta) = \rho\}$$
Distance of a point to the line

Given a line \( l(\theta, \rho) \) corresponding to the parameters \((\theta, \rho)\) and a point \( p \in \mathbb{R}^2 \) we denote \( d(p,l(\theta, \rho)) \) is the euclidian distance of the point \( p \) to the line \( l(\theta,\rho) \)

\[
d(p,l(\theta, \rho)) = | <p, \hat{n}(\theta)> - \rho | \tag{1}
\]

\[
= |\hat{\rho}(\theta, p) - \rho| \tag{2}
\]

With \( \hat{\rho}(\theta, p) = px \cos(\theta) + py \sin(\theta) \)
Line score

Given a set of points $p_1, \ldots, p_N$ and the parameters $(\theta, \rho)$ we can calculate a score for the line:

$$S(\theta, \rho) = \sum_{k=0}^{N} [d(p_k, l(\theta, \rho)) < \tau]$$

Where $[true] = 1$ and $[false] = 0$. $\tau$ is a limit distance beyong which we consider that the point is too far from the line to belong to the line.

The green points are referred as inliers and the black points as outliers. $S(\theta, \rho)$ is therefore the number of inliers.
Our goal is to find the parameters that give the best scores, i.e., the line that contains a maximum of points in their vicinity. We can discretize the space of parameters on a 2D grid using discretization steps $\Delta \theta$ and $\Delta \rho$.

A brute force approach consists in evaluating $S(\theta, \rho)$ for all combinations of parameters in the discretized set.
The brute force algorithm would write

```python
for j in range(len(thetas)):
    for i in range(len(rhos)):
        score[i,j]=0
        for k in range(len(points)):
            if dist_to_line(points[k],rhos[i],thetas[j])<tau:
                score[i,j]+=1
```

the complexity of the naive implementation is of order $N_\theta \times N_\rho \times N$
Given a point $p$ and an angle $\theta$ there is a unique line that contains $p$ with the angle $\theta$:

$$l(\theta, \rho) = \{ p \in \mathbb{R}^2 | px \cos(\theta) + py \sin(\theta) = \rho \}$$

So $p \in l(\theta, \rho) \iff \rho = px \cos(\theta) + py \sin(\theta)$
Given a point $p$ and an angle $\theta$ there is a set of parallel lines with angle $\theta$ that passes near $p$ at a distance smaller than $\tau$

$$d(p, l(\theta, \rho)) < \tau \iff \rho \in [\hat{\rho}(\theta, p) - \tau, \hat{\rho}(\theta, p) + \tau]$$

With $\hat{\rho}(\theta, p) = p_x \cos(\theta) + p_y \sin(\theta)$
A better approach than the brute force consist in looping over the points $p_i$ and then angles $\theta_0, \ldots, \theta_{N_\theta-1}$ and avoid the loop over all the $\rho_0, \ldots, \rho_{N_\rho-1}$ using the equivalence

$$d(p, l(\theta, \rho)) < \tau \iff \rho \in [\hat{\rho}(\theta, p) - \tau, \hat{\rho}(\theta, p) + \tau]$$

Assuming that the $\rho_0, \ldots, \rho_{N_\rho-1}$ are evenly spaced with $\Delta \rho$. We have $\rho_i = \rho_1 + i\Delta \rho$

$$d(p, l(\theta, \rho_i)) < \tau \iff \rho_i \in [\hat{\rho}(\theta, p) - \tau, \hat{\rho}(\theta, p) + \tau]$$
$$\iff \rho_0 + i\Delta \rho \in [\hat{\rho}(\theta, p) - \tau, \hat{\rho}(\theta, p) + \tau]$$
$$\iff i \in \lceil f(\hat{\rho}(\theta, p) - \tau) \rceil, \lfloor f(\hat{\rho}(\theta, p) + \tau) \rfloor$$

with $f(\rho) = ((\rho - \rho_0)/\Delta \rho)$ the function that allows to retrieve the index $i$ of some $\rho_i$ in the set $\rho_0, \ldots, \rho_{N_\rho-1}$ i.e $f(\rho_i) = i$
Acceleration

given the angle $\theta$ we can now avoid looping over the entire range of discrete $\rho$ and loop only in the the range given by:

$$d(p,l(\theta,\rho_i)) < \tau \iff i \in \left[ f(\hat{\rho}(\theta,p) - \tau), f(\hat{\rho}(\theta,p) + \tau) \right]$$

This leads to the pseudo code:

```python
for j in range(len(thetas)):
    for i in range(len(rhos)):
        score[i,j]=0
for k in range(len(points)):
    for j in range(len(thetas)):
        rho_min = hat_rho(p[k],theta[i])-tau
        rho_max = rho_min+ 2*tau
        i_rho_min = ceil((rho_min-rhos[0])/delta_rho)
        i_rho_max = floor(rho_max-rhos[0])/delta_rho)
        for i from i_rho_min to i_rho_max:
            if i>=0 and i<len(rhos):
                score[i,j]+=1
```

the complexity is now $N \times N_\theta \times \tau/\Delta \rho$
Finally, assuming that we have $\tau = \Delta \rho / 2$ the interval $[\hat{\rho}(\theta, p) - \tau, \hat{\rho}(\theta, p) + \tau]$ is of length $\Delta \rho$ and thus contains exactly one element $p_i$ of the set $\{\rho_0, \ldots, \rho_{N\rho - 1}\}$ with

$$i = \text{round}(f(\hat{\rho}(\theta, p)) = \text{round}((\rho - \rho_0)/\Delta \rho)$$

we get what is called the Hough transform

```python
for i in range(len(theta)):
    for j in range(len(rho)):
        score0[i, j] = 0
for k in range(len(points)):
    for j in range(len(theta)):
        i = round(hat_rho(p[k], theta[j]) - rhos[0]) / delta_rho
        score[i, j] += 1
```
We can visualize the accumulation in the parameter space for two points as follows:
By detecting edges in an image and accumulating for all the edge points in this image, this gives us a score image referred as the **accumulator** that looks like

![Image of accumulator](image)

A line in the original image corresponds to a peak in this transformed image. We select the local maximums above some inliers count threshold of this image to get a set of lines.
Hough transform

Détection de bords

Transformée de Hough $(s, \theta)$
Maxima locaux

Droites correspondantes
Once we found primitive candidates we can refine its location using a smoother version of the line score

$$S_{\text{Smooth}}(\theta, p) = \sum_{i=0}^{N} h(d(p_i, l(\theta, \rho)))$$

with $h$ the function

$$h(x) = \begin{cases} (1 - (x/\tau)^2)^3 & \text{if } |x| \leq \tau \\ 0 & \text{if } |x| > \tau \end{cases}$$
We can maximize $S_{smooth}(\theta, p)$ using a gradient ascent or a robust non linear least squares minimization method (see links).

Note that we could use this smoother energy within the Hough transform.
The Hough transform generalizes other primitives types that can be described with an analytic equation involving few parameters.

Will will use a N-dimensional accumulator for primitives that are parameterized with N parameters.

The size of the accumulator increases exponentially with the number of parameters.
Circles Hough transform

For example a circle detector will require a 3D accumulator for the radius and the two coordinates of the center. If we fix the radius we have two parameters left for the position of the center:

edges  hough transform  detected circle
we want to find the local maximums of

$$S(x, y, r) = \sum_{k=0}^{N} [\|p_k - (x, y)\| \in [r - \tau, r + \tau]]$$

given a fixed radius $r$ we first construct the set of points whose distance to the origin is within the range $[r - \tau, r + \tau]$:

$$C_r = \{(x, y) \in \mathbb{N}^2 | r - \tau \leq \sqrt{x^2 + y^2} \leq r + \tau \}$$

given $r$ this can be done once for all using some circle rasterization technique.

the function $S$ rewrites:

$$S(x, y, r) = \sum_{k=0}^{N} [(p_k - (x, y)) \in C_r]$$
Circles Hough transform

- we have:

\[ S(x, y, r) = \sum_{k=0}^{N} [(p_k - (x, y)) \in C_r] \]

- by posing \( \Delta = p_k - (x, y) \) we have

\[ (p_k - (x, y)) \in C_r \iff (x, y) = p_k - \Delta \text{ and } \Delta \in C_r \]

- Once \( C_r \) computed we can loop of the points in the image and \( C_r \) to increment the accumulators

```python
for j in range(len(thetas)):
    for i in range(len(rhos)):
        score[i, j] = 0
for k in range(len(points)):
    for delta in Cr
        score[p[k, 0]-delta[0], p[k, 1]-delta[1]] += 1
```
Suppose we gave a non-parametric rigid shape described by a set of points $s_1, \ldots, s_m$ this shape can be rigidly transformed $T(\theta, t, s_i) = R_\theta s_i + t$ with $R_\theta$ the rotation matrix

$$R_\theta = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix}$$

We want to find the transformation that aligns the shape points $s_1, \ldots, s_m$ with edges points $p_1, \ldots, p_n$ in the image.

We can formulate a score function:

$$S(\theta, t) = \sum_{i=1}^{n} [\min_j \|p_i - T(\theta, t, s_j)\|^2 < \tau] \quad (3)$$

This counts the number of observed point that fall near the transformed shape and thus is similar to the cost used previously.
Generalized hough transform

\[ S(\theta, t) = \sum_{i=1}^{n} [\min_j \| p_i - T(\theta, t, s_j) \|^2 < \tau] \]

- We will consider for now that the rotation angle \( \theta \) is fixed and thus we use a 2D accumulator for the translation.
- We can discretize \( t = t_1, \ldots, t_k \) and compute all \( S(\theta, t_k) \) using three nested loops over \( t, i \) and \( j \).
- We can define the set

\[ C_\theta = \{ p \in \mathbb{N}^2 | \min_j \| p - R_\theta s_i \|^2 < \tau \} \]

- We have

\[ S(\theta, t) = \sum_{k=0}^{N} [(p_k - t) \in C_\theta] \]

and we can use the same method as for circles.
$C_\theta$ might be large, instead:

- We use a new criterion $^1$

$$\tilde{S}(\theta, t) = \sum_{i=1}^{n} \sum_{j=1}^{m} [\|p_i - T(\theta, t, s_j)\| < \tau]$$

This criterion counts the number of pairs of point image/mode that are near each other after translation of the model.

- We can discretize $t = t_1, \ldots, t_k$ and compute all $\tilde{S}(\theta, t_k)$ using three nested loops over $t, i$ and $j$, this is not very efficient...

---

$^1$the fact that we removed the min operator a now use only sums gives us freedom on the order of the parameters in the nested loops when we perform summation
Generalized hough transform

- We notice that a given pair \((p_i, s_j)\) will contribute to increase the score of a very small set of translations \(t_k\):

\[
\|p_i - (R_\theta s_j + t_k)\| < \tau \iff \|t_k - (p_i - R_\theta s_j)\| < \tau
\]

- We can use nested loops for over \(i\), \(j\) and then enumerate only the few \(t\) values in the disk of center \(p_i - R_\theta s_j\) and radius \(\tau\).

- We can further speed up by accumulating only for \(t = p_i - R_\theta s_j\) and then convolve the accumulator by a disk of radius \(\tau\).

- If we do not want to keep \(\theta\) fixed, we use discretized angles \(\theta_1, \ldots, \theta_l\), use a 3D accumulator and add a loop over \(\theta\).\(^2\)

\(^2\) this approach has been proposed by Merlin and Farber
Generalized hough transform: Using normals

By computing edge normals for both the model 
\((\hat{n}_{s_1}, \ldots, \hat{n}_{s_m})\) and the edge points in the image 
\((\hat{n}_{p_1}, \ldots, \hat{n}_{p_n})\), we can change to criterion to

\[
\tilde{S}(\theta, t) = \sum_{i=1}^{n} \sum_{j=1}^{m} [(\|p_i - T((\theta, t, s_j))\| < \tau) \& \|\hat{n}_{p_i}, R_{\theta}\hat{n}_{s_j}\| < \tau_n]
\]

This now counts the number of pairs of point image/mode that are near each other and that have similar normal after the rotation of the model.

Thanks to the addition normal agreement constraint, from a given pair \((p_i, s_j)\) we can retrieve an interval for the rotation angle \(\theta\), and the pair will contribute to increase the score of a very small set of translations/rotation pairs \((t, \theta)\) which will accelerate the voting.
Some approximation that lead to an acceleration:

- Instead of having one high-dimensional array, we store a few two dimensional projections with common coordinates (e.g, \((t_x, t_y), (t_x, \theta), (t_y, \theta)\),

\[
S_{xy}(t_x, t_y) = \sum_{\theta} S(\theta, t_x, t_y) \tag{4}
\]
\[
S_{x\theta}(t_x, \theta) = \sum_{t_y} S(\theta, t_x, t_y) \tag{5}
\]
\[
S_{y\theta}(t_y, \theta) = \sum_{t_x} S(\theta, t_x, t_y) \tag{6}
\]

- Find consistent peaks in these lower dimensional arrays
Another generic approach to find primitives in a set of points is called the RANSAC method for "RANdom SAmple Consensus". The idea is to

- sample randomly just enough points to estimate the parameters of the unique primitive that contain these points (2 points for lines, 3 points for circles)
- Count how many points in the whole set of points agree with the generated primitive

repeat this step many times and keep the primitive that contains the largest number of inlier points.
Set of points:
Select two points at random:
find the parameters of the line that fit the two points:
calculate the distance of all the points to the line:
select data that support the current hypothesis:
repeat sampling:
repeat sampling:
repeat sampling:
best_score=inf
for t in range(nb_iteration):
    i=random(len(points))
    j=random(len(points))
    rho,heta=get_line_parameters(points[i],points[j])
    score=0
    for k in range(len(points)):
        if dist_to_line(points[k],rho,theta)<tau:
            score+=1
    if score>best_score:
        best_rho=rho
        best_theta=teta
It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability this probability increasing as more iterations are allowed. Using some statistics we can estimate how many test should be done to get a good chance of finding the best line the time needed to get a good solution increases with the number of outliers as we get more chance to pick an outlier at each random selection RANSAC usually performs badly when the number of inliers is less than 50%
Once we found candidate poses using the generalized hough transform we can perform refinement using an iterative approach.

A classical method to align two set of points given an initial estimate of the alignment is called the **Iterative closest point (ICP)** algorithm consists in minimizing iteratively this energy

\[
E(\theta, t) = \sum_{j=1}^{m} \min_{i} \| p_i - T(\theta, t, s_j) \|^2
\]

Note that the summation is done of the points \( s_j \) instead of the point \( p_i \) as done in eqn.3 For each point in the shape the image we look for the closest point in the image.
Refinement: ICP

\[ E(\theta, t) = \sum_{j=1}^{m} \min_{i} \| p_i - T(\theta, t, s_j) \|^2 \]

We reformulate this problem by adding a vector of auxiliary variables \( L = (l_1, \ldots, l_m) \in \{1, \ldots, n\}^m \) that encode for each point of the shape the index of the corresponding point in the image:

\[ E'(\theta, t, L) = \sum_{j=1}^{m} \| p_{l_j} - T(\theta, t, s_j) \|^2 \]
\[ E'(\theta, t, L) = \sum_{j=1}^{m} \| p_{l_j} - T(\theta, t, s_j) \|^{2} \]

Similarly to the kmeans algorithm, we will minimize \( E' \) alternatively with respect to \((\theta, t)\) and with respect to \(L\).

- The minimization with respect to \(L\) is done by solving \(m\) independant problems: \( l_j = \arg\min_{i} \| p_i - T(\theta, t, s_j) \|^{2} \). For each point of the shape we look for the closest point in the image.

- The minimization with respect to \(t\) is done by displacing the shape with the mean of the point differences:

\[
t_{k+1} = t_k + \frac{1}{m} \sum_{j=1}^{m} p_{l_j} - T(\theta, t_k, s_j)
\]
the minimization with respect to $\theta$ is more complex and require a singular value decomposition [1, 2]. Alternatively we can use a gauss-newton iterative minimization (see last slides)
ICP

iteration 1
iteration 2
iteration 3
iteration 4
iteration 5
iteration 6
Given \( n \) functions \( f_i(\Theta) \) from \( \mathbb{R}^m \) into \( \mathbb{R} \) with \( \Theta=(\theta_1,\ldots,\theta_m) \), we want to minimize the function

\[
S(\Theta) = \sum_{i=1}^{n} f_i(\Theta)^2
\]

This is referred in the literature as a \textbf{non-linear least squares} problem.
Gauss-Newton

\[ S(\Theta) = \sum_{i=1}^{n} f_i(\Theta)^2 \]

This can be solved using the iterative minimization. At each iteration, we linearize \( f_i(\Theta) \) around \( \Theta_t \):

\[ f_i(\Theta) \simeq f_i(\Theta_t) + \nabla f_i(\Theta_t)(\Theta - \Theta_t) \]

we get

\[ S(\Theta) \simeq \sum_{i=1}^{n} (f_i(\Theta_t) + \nabla f_i(\Theta_t)(\Theta - \Theta_t))^2 = \| F + J_f(\Theta - \Theta_t) \|^2 \]

With \( F \) the vector \([f_1(\Theta_t), \ldots, f_n(\Theta_t)]\) and \( J_f \) the jacobian matrix evaluated at location \( \Theta_t \):

\[
J_f = \begin{bmatrix}
\frac{\partial f_1}{\partial \theta_1} |_{\Theta_t} & \cdots & \frac{\partial f_1}{\partial \theta_m} |_{\Theta_t} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial \theta_1} |_{\Theta_t} & \cdots & \frac{\partial f_n}{\partial \theta_m} |_{\Theta_t}
\end{bmatrix}
\]
We have from the previous slide

\[ S(\Theta) \simeq \| F + J_f(\Theta - \Theta_t) \|^2 \]

We get the next estimate \( \Theta_{t+1} \) as the minimum of this approximation. The gradient write

\[ 2J_f^T (F + J_f(\Theta - \Theta_t)) \]

and we get the minium for

\[ \Theta_{t+1} = \Theta_t - (J_f^T J_f)^{-1} J_f^T F \]

This iterative method is called the Gauss-Newton algorithm.
tutorial on Robust Non-Linear Least-Squares and applications to computer vision

http://cvlabwww.epfl.ch/~fua/courses/lsq/Intro.htm