Pattern Recognition

Guillaume Lemaître

22 mai 2008
Plan

1. Introduction
2. Methods
3. Performance
4. Plan
Plan

1. Introduction
2. Methods
3. Performance
4. Plan
Create a system which detect different objects:

- Mid water target
- Bottom target
- Tyre
- Cone
Constraints :

- « Real-Time » system
- Use minimum ressources (CPU, memory, ...
Use technologie:

- Acquisition with analog camera and card MPEG 4
- Intel OpenCV (Computer Vision) library
- Programmation in C++
Plan

1. Introduction
2. Methods
3. Performance
4. Plan
Color detection:

**Processing:**

- Convert BGR Image to YCrCb
- Take only channel Cr
- Threshold to keep the equivalence of orange color
- Morphology operation: open
Color detection:

**Processing for mid water target:**

- Search contours
- Calculate perimeter and area
- Calculate circularity: \[ \text{circularity} = \frac{4 \times \pi \times \text{area}}{\left( \text{perimeter}^2 \right)} = 1 \text{ if perfect circle} \]

**Decision:**

- If circularity superior to a specific threshold
Form detection:

**Lign Hough Transform:**

- Each points admit an infinity of straight lines.
- The general equation of lines which pass by one point is:
  \[ y_i = a \cdot x_i + b \quad \text{with point coordinates} \ (x_0, y_0) \]
- But we use the polar representation which is:
  \[ \rho(\Theta) = x_0 \cdot \cos(\Theta) + y_0 \cdot \sin(\Theta) \quad \text{with point coordinates} \ (x_0, y_0) \]
Form detection:

**Lign Hough Transform**:

- Each line is characteristic of two parameters $\Theta$ and $\rho$.

- Plan of Hough:

  $$\rho(\Theta) = x_0 \cos(\Theta) + y_0 \sin(\Theta)$$

  *with point coordinates $(x_0, y_0)$*
**Form detection:**

**Lign Hough Transform:**

- Each point has a representation in space of Hough

- Intersection in space of Hough represents a straight line in space cartesien
Form detection:

**Lign Hough Transform:**

- Implementation of Line Hough Transform in OpenCV:
  
  - `CvSeq* cvHoughLines2( CvArr* image, void* line_storage, int method, double rho, double theta, int threshold, double param1=0, double param2=0 )`

- Rho: Distance resolution in pixel
- Theta: Angle resolution in pixel
- Threshold: Number minimum of points
Form detection:

**Circle Hough Transform:**

- The general equation of circle is:
  \[(x_0 - a)^2 + (y_0 - b)^2 = r^2\]
  with point coordinates \((x_0, y_0)\)

- The parametric equations are:
  \[x = a + R \cdot \cos(\Theta) \quad \Leftrightarrow \quad a = x - R \cdot \cos(\Theta)\]
  \[y = b + R \cdot \sin(\Theta) \quad \Leftrightarrow \quad b = y - R \cdot \sin(\Theta)\]
Form detection:

**Circle Hough Transform:**

Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the \((a, b)\) that is the center in geometric space.
Form detection:

**Circle Hough Transform**:

- Implementation of Circles Hough Transform in OpenCV:

  ```cpp
  CvSeq* cvHoughCircles( CvArr* image, void* circle_storage, int method, double dp, double min_dist, double param1=100, double param2=100 )
  ```

  - min-dist : minimum distance between centers
  - param1 : threshold Canny
  - param2 : Number minimum of points on circles
Tracking Object:

**Mean-shift Algorithm:**

- **Aim:** Search a model in image

- **Two stages:**
  - Initialisation: definition model
  - Processing: search model in image
Mean-shift Algorithm:

- Initialisation: definition model:
  - Create a histogram with discretisation of the representation chosen (hue, saturation ...)
  - Calculate density gradient estimation of the representation chosen.

\[
\hat{q}_u = C \cdot \sum_{i=1}^{n} k(||x_i^*||^2) \cdot \delta(c(x_i^*), u) \]
\[
C = \frac{1}{\sum_{i=1}^{n} k(||x_i^*||^2)}
\]
Tracking Object:

**Mean-shift Algorithm:**

- Initialisation : definition model :

  - Initialisation of the current position in \( y_0 \leftarrow y(t) \)
Mean-shift Algorithm:

- Iteration: search model in current image:
  - Calculate density gradient estimation of current candidate in:
    \[ p(y_0) = (p_u(y_0))_{u=1 \ldots m} \]
  - Calculate the Bhattacharyya distance
    \[ \rho(y_0) = \rho(p(y_0), q) = \sum_{u=1}^{m} \sqrt{p_u(y_0) \cdot q_u} \]
**Mean-shift Algorithm**:

- Iteration: search model in current image:
  - The Bhattacharya distance measures the similarity between two discrete probability which are here the density gradient estimation of model and current candidate

$$\hat{p}(y) = \hat{p}(\hat{p}(y), \hat{q}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(y) \cdot \hat{q}_u}$$
Mean-shift Algorithm:

- Iteration: search model in current image:
  - Calculate weight vector
    \[ w_i = \sum_{u=1}^{m} \delta(c(x_i), u) \cdot \sqrt{q_u/p_u(y_0)} \]
  - Calculate position of next candidate
    \[
    y_1 = \frac{\sum_{i=1}^{n_h} x_i \cdot w_i \cdot g\left(\frac{y_0-x_i}{h}\right)}{\sum_{i=1}^{n_h} w_i \cdot g\left(\frac{y_0-x_i}{h}\right)}
    \]
Mean-shift Algorithm:

• Implementation of Mean-Shift in OpenCV:
  
• `int cvCamShift( const CvArr* prob_image, CvRect window, CvTermCriteria criteria, CvConnectedComp* comp, CvBox2D* box=NULL );`

• `void cvCalcBackProject( IplImage** image, CvArr* back_project, const CvHistogram* hist );`
Plan

1. Introduction
2. Methods
3. Performance
4. Plan

Comparison
## Comparison

<table>
<thead>
<tr>
<th>Methods</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color detection</td>
<td>• Detection less fast than other presented</td>
<td>• Working very nice near of object</td>
</tr>
<tr>
<td></td>
<td>• Only for orange object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Don't working far of object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Use 40% of CPU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• High dependance of thresholds</td>
<td></td>
</tr>
<tr>
<td>Hough Transformation</td>
<td>• Many false alarm</td>
<td>• More fast than previous method</td>
</tr>
<tr>
<td></td>
<td>• Use 25% of CPU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Don't working far of object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• High dependance of parameters</td>
<td></td>
</tr>
<tr>
<td>Mean-Shift algorithm</td>
<td>• High dependance of histogram of start</td>
<td>• Method very fast</td>
</tr>
<tr>
<td></td>
<td>• Sensitive at noise</td>
<td>• Working far of object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use 1% CPU</td>
</tr>
</tbody>
</table>
Plan

1. Introduction
2. Methods
3. Performance
4. Plan
Application of methods to detect different objects