Image Processing Using OpenCV

Jos Elfring

Embedded Motion Control 2013

Technische Universiteit
Eindhoven
University of Technology

October 9, 2013

OpenCV

- Open source computer vision library

OpenCV

- Open source computer vision library
- Supports Windows, Linux, Mac OS, iOS and Android

OpenCV

- Open source computer vision library
- Supports Windows, Linux, Mac OS, iOS and Android
- Written in C++, interfaces in C++, C, Python and Java

OpenCV

OpenCV

- Open source computer vision library
- Supports Windows, Linux, Mac OS, iOS and Android
- Written in C++, interfaces in C++, C, Python and Java
- Within ROS: just add dependencies to manifest.xml:
  - <depend package="opencv2"\>
  - <depend package="cv_bridge"\>

Technische Universiteit
Eindhoven
University of Technology

Listen to a camera stream

demo_opencv.cpp

```
...

#include <sensor_msgs/Image.h>

void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    // ...process image
}

int main() {
    // Initialize ros and create node handle
    ros::init(argc, argv, "demo_opencv");
    ros::NodeHandle nh;

    // Subscribe to image from camera
    ros::Subscriber cam_img_sub =
          nh.subscribe("/pico/camera", 1, &imageCallback);

    ...
}
```

Converting a ROS image to an OpenCV image


```
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>

void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    cv_bridge::CvImagePtr img_ptr;
    cv::Mat img_rgb;
    try {
        img_ptr = cv_bridge::toCvCopy(color_img,
              sensor_msgs::image_encodings::BGR8);
        img_rgb = img_ptr->image;
    }
    catch (cv_bridge::Exception& e) {
        ROS_ERROR("cv_bridge exception: \%s", e.what());
        return;
    }
    // ...continue processing
}
```

Showing an OpenCV image

- Images can be displayed in a separate window

```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    cv::imshow("Camera image", img_rgb);
    cv::waitKey(3);

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Showing an OpenCV image

- Images can be displayed in a separate window

```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    cv::imshow("Camera image", img_rgb);
    cv::waitKey(3);

    ...
}
```

- `cv::waitkey(0)` → wait for user to press button

Converting an RGB image to HSV

- Images can be converted from one color space to another, *e.g.*, from RGB to Hue, Saturation, Value (HSV)

```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    cv::Mat img_hsv;
    cv::cvtColor(img_rgb, img_hsv, CV_BGR2HSV);

    cv::imshow("HSV image", img_hsv);
    cv::waitKey(3);

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Color detection: theory

- Each RGB or HSV pixel has three values:
  - $h \in [0, 180]$, $s \in [0, 255]$, $v \in [0, 255]$
  - $r \in [0, 255]$, $g \in [0, 255]$, $b \in [0, 255]$

Color detection: theory

- Each RGB or HSV pixel has three values:
  - $h \in [0, 180]$, $s \in [0, 255]$, $v \in [0, 255]$
  - $r \in [0, 255]$, $g \in [0, 255]$, $b \in [0, 255]$
- A color can be represented by a region in color space:
  - Red: $hue \in [165, 179] \cap sat \in [240, 255] \cap val \in [100, 175]$

Technische Universiteit Eindhoven University of Technology

Color detection: theory

- Each RGB or HSV pixel has three values:
  - $h \in [0, 180]$, $s \in [0, 255]$, $v \in [0, 255]$
  - $r \in [0, 255]$, $g \in [0, 255]$, $b \in [0, 255]$
- A color can be represented by a region in color space:
  - Red: $hue \in [165, 179] \cap sat \in [240, 255] \cap val \in [100, 175]$
- Find all red pixels:

$$pixel = \begin{cases} 255 & h \in [165, 179] \cap s \in [240, 255] \cap v \in [100, 175] \\ 0 & \text{otherwise.} \end{cases}$$

Technische Universiteit Eindhoven University of Technology

Color detection: theory

- Each RGB or HSV pixel has three values:
  - $h \in [0, 180]$, $s \in [0, 255]$, $v \in [0, 255]$
  - $r \in [0, 255]$, $g \in [0, 255]$, $b \in [0, 255]$

- A color can be represented by a region in color space:
  - Red: $hue \in [165, 179] \cap sat \in [240, 255] \cap val \in [100, 175]$

- Find all red pixels:

$$pixel = \begin{cases} 255 & h \in [165, 179] \cap s \in [240, 255] \cap v \in [100, 175] \\ 0 & \text{otherwise.} \end{cases}$$

- Result is a binary image:
  - White means original pixel was red
  - Black means original pixel was not red

TU/e Technische Universiteit Eindhoven University of Technology

Color detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    // Create a new image in which the result can be stored
    cv::Mat img_binary;

    // Set the thresholds (hue, saturation, value)
    cv::Scalar min_vals(165, 240, 100);
    cv::Scalar max_vals(179, 255, 175);

    // Perform thresholding
    cv::inRange(img_hsv, min_vals, max_vals, img_binary);

    ...
}
```

Color detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    // Create a new image in which the result can be stored
    cv::Mat img_binary;

    // Set the thresholds (hue, saturation, value)
    cv::Scalar min_vals(165, 240, 100);
    cv::Scalar max_vals(179, 255, 175);

    // Perform thresholding
    cv::inRange(img_hsv, min_vals, max_vals, img_binary);

    ...
}
```


TU/e Technische Universiteit Eindhoven University of Technology

Color detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    // Create a new image in which the result can be stored
    cv::Mat img_binary;

    // Set the thresholds (hue, saturation, value)
    cv::Scalar min_vals(165, 240, 100);
    cv::Scalar max_vals(179, 255, 175);

    // Perform thresholding
    cv::inRange(img_hsv, min_vals, max_vals, img_binary);

    ...
}
```

Color detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...

    // Create a new image in which the result can be stored
    cv::Mat img_binary;

    // Set the thresholds (hue, saturation, value)
    cv::Scalar min_vals(165, 240, 100);
    cv::Scalar max_vals(179, 255, 175);

    // Perform thresholding
    cv::inRange(img_hsv, min_vals, max_vals, img_binary);

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Reading values from a cv::Mat

- Each element in a `cv::Mat` represents a pixel

Technische Universiteit
Eindhoven
University of Technology

Reading values from a cv::Mat

- Each element in a `cv::Mat` represents a pixel
- Dependent on the color space (number of channels), each pixel can be
  - `unsigned char` for a grayscale image
  - `cv::Vec3b` (= vector of three unsigned chars) for HSV image
  - ...

Reading values from a cv::Mat

- Each element in a `cv::Mat` represents a pixel

- Dependent on the color space (number of channels), each pixel can be
  - `unsigned char` for a grayscale image
  - `cv::Vec3b` (= vector of three unsigned chars) for HSV image
  - ...

- Reading pixel (i, j):
  - Grayscale image: `img_grayscale.at<unsigned char>(i,j)`
  - HSV image: `img_hsv.at<cv::Vec3b>(i,j)`
  - ...

Iterating over a binary image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {

    ...
    unsigned int n_pixels_thr = 0;

    // Loop over the image
    for (int y = 0; y < img_binary.rows; y++)
    {
        for (int x = 0; x < img_binary.cols; x++)
        {
            if (img_binary.at<unsigned char>(y,x) == 255)
            {
                ++n_pixels_thr;  // Count 'red' pixels
            }
        }
    }
    ...
}
```

Iterating over an HSV image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Loop over all pixels
    for (int y = 0; y < img_hsv.rows; ++y) {
        for (int x = 0; x < img_hsv.cols; ++x) {

            // Get value current channel and current pixel
            const cv::Vec3b& s = img_hsv.at<cv::Vec3b>(y, x);

            // For each of the three channels (hue, sat, val)
            for (int c = 0; c < 3; ++c) {
                unsigned int pxl_val = (unsigned int)s.val[c];
                // ... do stuff with pxl_val
            }
        }
    }
    ...
}
```

Flood fill algorithm

- Give connected pixels within the same range the same color

Flood fill algorithm

- Give connected pixels within the same range the same color

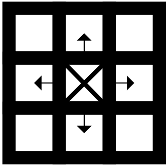
- Used in, *e.g.*, Minesweeper game

Flood fill algorithm

- Give connected pixels within the same range the same color
- Used in, *e.g.*, Minesweeper game
- Pick a seed pixel $pix(i_s, j_s)$ and explore neighborhood, two options:

Flood fill algorithm

- Give connected pixels within the same range the same color
- Used in, *e.g.*, Minesweeper game
- Pick a seed pixel $pix(i_s, j_s)$ and explore neighborhood, two options:



Technische Universiteit
Eindhoven
University of Technology

Flood fill algorithm

- Give connected pixels within the same range the same color
- Used in, *e.g.*, Minesweeper game
- Pick a seed pixel $pix(i_s, j_s)$ and explore neighborhood, two options:

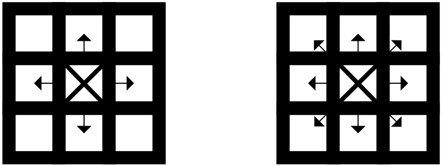
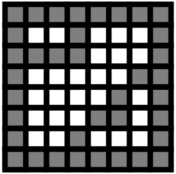


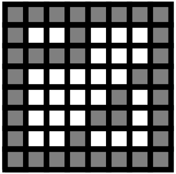
Figure: 4- versus 8-connectivity

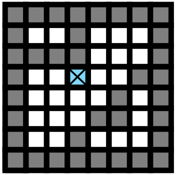
Technische Universiteit Eindhoven University of Technology

Flood fill algorithm: example

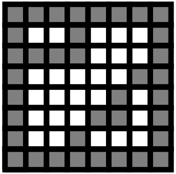


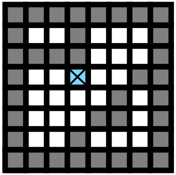
Flood fill algorithm: example

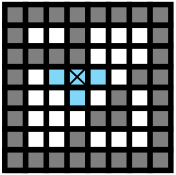




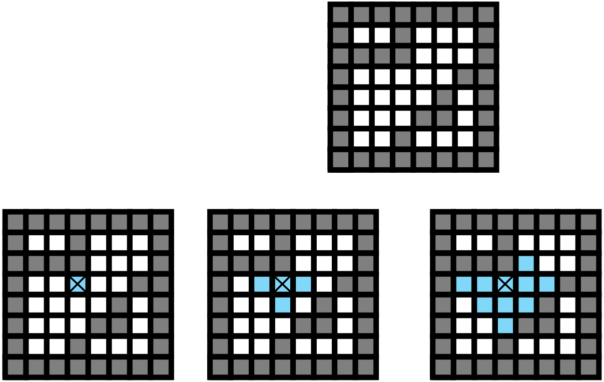
Flood fill algorithm: example



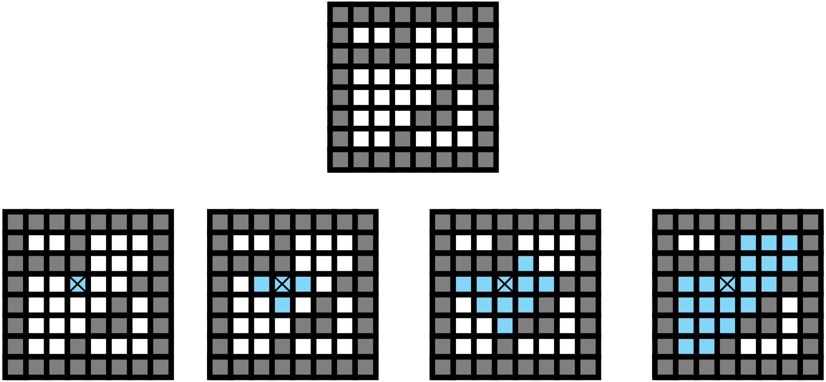




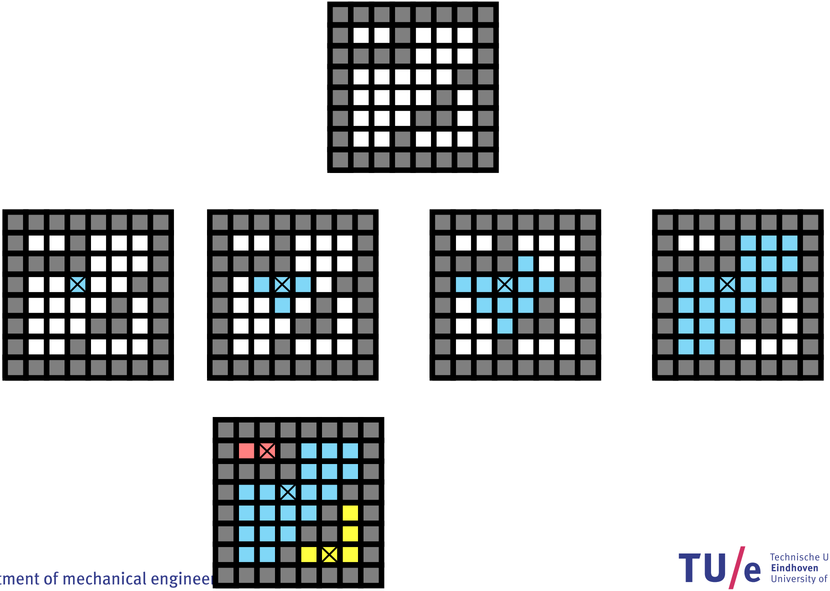
Flood fill algorithm: example



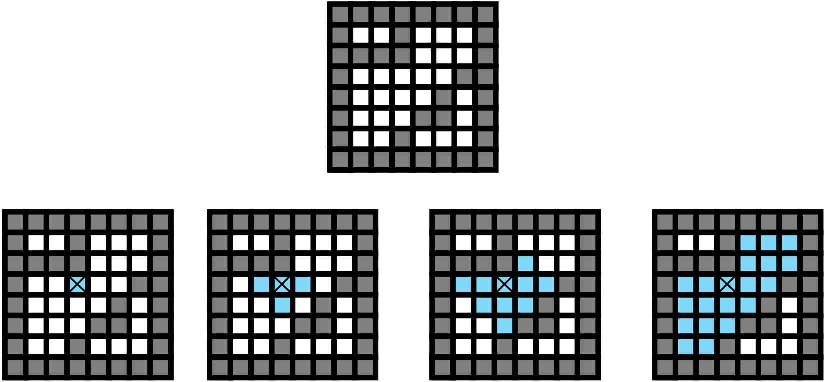
Flood fill algorithm: example

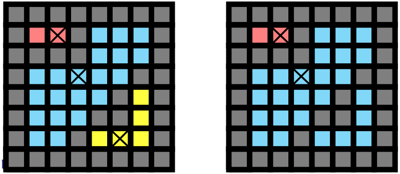


Flood fill algorithm: example

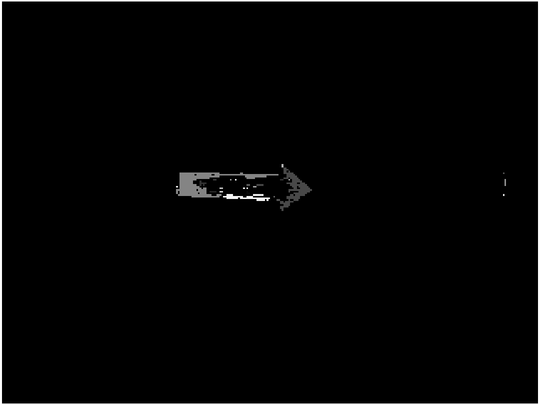


Flood fill algorithm: example





Flood fill algorithm on real data



Flood fill algorithm in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...
    // Image used to show results
    cv::Mat img_blobs = img_binary.clone();

    for(int y = 0; y < img_blobs.rows; y++) {
        for(int x = 0; x < img_blobs.cols; x++) {

            // Only 'red' pixels are used as seed pixels
            if (img_blobs.at<unsigned char>(y,x) == 255) {
                cv::Rect rect;
                unsigned int conn_val = 4; // or 8
                cv::floodFill(img_blobs, cv::Point(x,y),
                  cv::Scalar(rand()&255), &rect, cv::Scalar(0),
                  cv::Scalar(0), conn_val);
            }
        }
    }
    ...
}
```


Technische Universiteit Eindhoven University of Technology

Flood fill algorithm in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...
    // Image used to show results
    cv::Mat img_blobs = img_binary.clone();

    for(int y = 0; y < img_blobs.rows; y++) {
        for(int x = 0; x < img_blobs.cols; x++) {

            // Only 'red' pixels are used as seed pixels
            if (img_blobs.at<unsigned char>(y,x) == 255) {
                cv::Rect rect;
                unsigned int conn_val = 4; // or 8
                cv::floodFill(img_blobs, cv::Point(x,y),
                  cv::Scalar(rand()&255), &rect, cv::Scalar(0),
                  cv::Scalar(0), conn_val);
            }
        }
    }
    ...
}
```

Flood fill algorithm in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...
    // Image used to show results
    cv::Mat img_blobs = img_binary.clone();

    for(int y = 0; y < img_blobs.rows; y++) {
        for(int x = 0; x < img_blobs.cols; x++) {

            // Only 'red' pixels are used as seed pixels
            if (img_blobs.at<unsigned char>(y,x) == 255) {
                cv::Rect rect;
                unsigned int conn_val = 4; // or 8
                cv::floodFill(img_blobs, cv::Point(x,y),
                  cv::Scalar(rand()&255), &rect, cv::Scalar(0),
                  cv::Scalar(0), conn_val);
            }
        }
    }
    ...
}
```

Edge detection: theory

Edge is ‘jump’ in intensity $\rightarrow$ find peaks after discrete differentiation.

Edge detection: theory

Edge is 'jump' in intensity → find peaks after discrete differentiation.

For example, the Sobel operator uses two 3×3 kernels which are convolved with the original image (left → right and up → down):

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad \text{and} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I.$$

TU/e Technische Universiteit Eindhoven University of Technology

Edge detection: theory

Edge is ‘jump’ in intensity $\rightarrow$ find peaks after discrete differentiation.

For example, the Sobel operator uses two 3×3 kernels which are convolved with the original image (left $\rightarrow$ right and up $\rightarrow$ down):

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad \text{and} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I.$$

Example:

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & -8 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

Edge detection: theory

Edge is ‘jump’ in intensity $\rightarrow$ find peaks after discrete differentiation.

For example, the Sobel operator uses two 3×3 kernels which are convolved with the original image (left $\rightarrow$ right and up $\rightarrow$ down):

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad \text{and} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I.$$

Example:

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & -8 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

If the ‘derivative’ falls within some range $\rightarrow$ edge.

Technische Universiteit Eindhoven University of Technology

Canny edge detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Blur the image
    cv::Mat img_edges;
    unsigned int kernel_size = 3;
    cv::blur(img_binary, img_edges,
         cv::Size(kernel_size, kernel_size));

    // Detect edges (hysteresis thresholding)
    double low_thr = 50;
    cv::Canny(img_edges, img_edges,
         low_thr, 3*low_thr, kernel_size);

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Canny edge detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Blur the image
    cv::Mat img_edges;
    unsigned int kernel_size = 3;
    cv::blur(img_binary, img_edges,
         cv::Size(kernel_size, kernel_size));

    // Detect edges (hysteresis thresholding)
    double low_thr = 50;
    cv::Canny(img_edges, img_edges,
         low_thr, 3*low_thr, kernel_size);

    ...
}
```

Canny edge detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Blur the image
    cv::Mat img_edges;
    unsigned int kernel_size = 3;
    cv::blur(img_binary, img_edges,
         cv::Size(kernel_size, kernel_size));

    // Detect edges (hysteresis thresholding)
    double low_thr = 50;
    cv::Canny(img_edges, img_edges,
         low_thr, 3*low_thr, kernel_size);

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

1. Determine for an edge pixel (x_1, y_1), the family of lines passing through that pixel:

$$r_{\theta,1} = x_1 \cos(\theta) + y_1 \sin(\theta), \quad r_\theta > 0,\ 0 < \theta \leq 2\pi.$$

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

1. Determine for an edge pixel (x_1, y_1), the family of lines passing through that pixel:

$$r_{\theta,1} = x_1 \cos(\theta) + y_1 \sin(\theta), \quad r_\theta > 0,\ 0 < \theta \leq 2\pi.$$

2. Repeat for the second pixel (x_2, y_2): $r_{\theta,2} = x_2 \cos(\theta) + y_2 \sin(\theta)$

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

1. Determine for an edge pixel (x_1, y_1), the family of lines passing through that pixel:

$$r_{\theta,1} = x_1\cos(\theta) + y_1\sin(\theta), \quad r_\theta > 0,\ 0 < \theta \leq 2\pi.$$

2. Repeat for the second pixel (x_2, y_2): $r_{\theta,2} = x_2\cos(\theta) + y_2\sin(\theta)$
   - If $r_{\theta,1}$ and $r_{\theta,2}$ intersect, pixels (x_1, y_1) and (x_2, y_2) on the same line

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

1. Determine for an edge pixel (x_1, y_1), the family of lines passing through that pixel:

$$r_{\theta,1} = x_1\cos(\theta) + y_1\sin(\theta), \quad r_\theta > 0,\ 0 < \theta \leq 2\pi.$$

2. Repeat for the second pixel (x_2, y_2): $r_{\theta,2} = x_2\cos(\theta) + y_2\sin(\theta)$
   - If $r_{\theta,1}$ and $r_{\theta,2}$ intersect, pixels (x_1, y_1) and (x_2, y_2) on the same line
3. Repeat for all edge pixels

Line detection using the Hough transform

Hough transform can be used to find lines through edge pixels:

$$y = ax + b, \quad \text{or in polar coordinates:} \quad r = x\cos(\theta) + y\sin(\theta)$$

Idea:

1. Determine for an edge pixel (x_1, y_1), the family of lines passing through that pixel:

$$r_{\theta,1} = x_1\cos(\theta) + y_1\sin(\theta), \quad r_\theta > 0,\ 0 < \theta \leq 2\pi.$$

2. Repeat for the second pixel (x_2, y_2): $r_{\theta,2} = x_2\cos(\theta) + y_2\sin(\theta)$
   - If $r_{\theta,1}$ and $r_{\theta,2}$ intersect, pixels (x_1, y_1) and (x_2, y_2) on the same line
3. Repeat for all edge pixels
   - If the number of intersections is above threshold $\rightarrow$ found a line with parameters (θ, r_θ)

Hough transform function in OpenCV (for lines)

- Standard Hough transform
  - Implements previous slide
  - Output: vector of pairs (θ, r_θ)
  - `cv::HoughLines(...);`

Hough transform function in OpenCV (for lines)

- Standard Hough transform
  - Implements previous slide
  - Output: vector of pairs (θ, r_θ)
  - `cv::HoughLines(...);`

- Probabilistic Hough line transform
  - A more efficient implementation
  - Output: vector of line endpoints (x_0, y_0, x_1, y_1)
  - `cv::HoughLinesP(...);`

Line detection in OpenCV: the Hough transform


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Vector in which the lines will be stored
    std::vector<cv::Vec4i> lines;

    // Perform Hough transform
    double resolution_r = 1;
    double resolution_theta = CV_PI/180;
    unsigned int min_n_intersec = 10;
    unsigned int min_n_pts = 15;
    unsigned int max_gap = 5;
    cv::HoughLinesP(img_edges, lines, resolution_r,
        resolution_theta, min_n_intersec, min_n_pts, max_gap);

    ...
}
```

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
             line_width, CV_AA);
    }
    ...
}
```

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
            line_width, CV_AA);
    }
    ...
}
```

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
            line_width, CV_AA);
    }
    ...
}
```

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
             line_width, CV_AA);
    }
    ...
}
```

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
            line_width, CV_AA);
    }
    ...
}
```


Technische Universiteit Eindhoven University of Technology

Draw detected lines on a RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...


    // Copy RGB image
    cv::Mat img_with_lines = img_rgb.clone();
    // Set line characteristics
    cv::Scalar line_color(0, 0, 255);
    unsigned int line_width = 3;

    for (size_t i = 0; i < lines.size(); i++)
    {
        // Add line to the copied image
        cv::Vec4i line_i = lines[i];
        cv::Point point1(line_i[0], line_i[1]);
        cv::Point point2(line_i[2], line_i[3]);
        cv::line(img_with_lines, point1, point2, line_color,
            line_width, CV_AA);
    }
    ...
}
```


Technische Universiteit Eindhoven University of Technology

Harris corner detector: idea

Idea: a corner is a point with dominant but different gradients

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image
2. Change in intensity for shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y) \underbrace{\left[I(x + u, y + v) - I(x, y)\right]^2}_{\text{large for distinctive patches}}$$

Technische Universiteit Eindhoven University of Technology

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image
2. Change in intensity for shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y) \underbrace{[I(x + u, y + v) - I(x, y)]^2}_{\text{large for distinctive patches}}$$

3. First order approximation:

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y,$$

where I_x, I_y are partial derivatives of I.

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image
2. Change in intensity for shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y) \underbrace{[I(x + u, y + v) - I(x, y)]^2}_{\text{large for distinctive patches}}$$

3. First order approximation:

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y,$$

where I_x, I_y are partial derivatives of I. Now:

$$E(u, v) \approx \sum_{x,y} w(x, y) [uI_x + vI_y]^2$$

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image
2. Change in intensity for shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y) \underbrace{\left[I(x + u, y + v) - I(x, y)\right]^2}_{\text{large for distinctive patches}}$$

3. First order approximation:

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y,$$

where I_x, I_y are partial derivatives of I. Now:

$$E(u, v) \approx \sum_{x,y} w(x, y) \left[uI_x + vI_y\right]^2 = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

TU/e Technische Universiteit Eindhoven University of Technology

Harris corner detector: theory (1/2)

1. Sweep window $w(x, y)$ (*e.g.*, 1 in window, 0 outside) over image
2. Change in intensity for shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y) \underbrace{[I(x+u, y+v) - I(x, y)]^2}_{\text{large for distinctive patches}}$$

3. First order approximation:

$$I(x+u, y+v) \approx I(x, y) + uI_x + vI_y,$$

where I_x, I_y are partial derivatives of I. Now:

$$E(u, v) \approx \sum_{x,y} w(x, y) [uI_x + vI_y]^2 = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix},$$

where M is the Harris matrix:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}.$$

Harris corner detector: theory (2/2)

4. Corner has large variations $\rightarrow$ large eigenvalues, however calculating eigenvalues is computationally expensive.

TU/e Technische Universiteit Eindhoven University of Technology

Harris corner detector: theory (2/2)

4. Corner has large variations $\rightarrow$ large eigenvalues, however calculating eigenvalues is computationally expensive. Define a score:

$$R = \det M - k\,(\operatorname{trace} M)^2\,,$$

where $k \in [0.04, 0.15]$ is determined empirically and:

$$\begin{aligned} \det M &= \lambda_1 \lambda_2 \\ \operatorname{trace} M &= \lambda_1 + \lambda_2 \end{aligned}$$

TU/e Technische Universiteit Eindhoven University of Technology

Harris corner detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Harris Detector parameters
    int block_size = 2;
    int size_sobel_kernel = 3; // 1, 3, 5 or 7
    double k = 0.1;

    // Detect corners using Harris corner detector
    cv::Mat corners = cv::Mat::zeros(img_binary.size(),CV_32FC1);
    cv::cornerHarris(img_binary, corners, block_size,
         size_sobel_kernel, k, cv::BORDER_DEFAULT);

    // Normalize 'scores'
    cv::normalize(corners, corners, 0, 255, cv::NORM_MINMAX,
         CV_32FC1, cv::Mat());
    cv::convertScaleAbs(corners, corners);

    ...
}
```

Harris corner detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Harris Detector parameters
    int block_size = 2;
    int size_sobel_kernel = 3; // 1, 3, 5 or 7
    double k = 0.1;

    // Detect corners using Harris corner detector
    cv::Mat corners = cv::Mat::zeros(img_binary.size(),CV_32FC1);
    cv::cornerHarris(img_binary, corners, block_size,
         size_sobel_kernel, k, cv::BORDER_DEFAULT);

    // Normalize 'scores'
    cv::normalize(corners, corners, 0, 255, cv::NORM_MINMAX,
         CV_32FC1, cv::Mat());
    cv::convertScaleAbs(corners, corners);

    ...
}
```

Harris corner detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Harris Detector parameters
    int block_size = 2;
    int size_sobel_kernel = 3; // 1, 3, 5 or 7
    double k = 0.1;

    // Detect corners using Harris corner detector
    cv::Mat corners = cv::Mat::zeros(img_binary.size(),CV_32FC1);
    cv::cornerHarris(img_binary, corners, block_size,
         size_sobel_kernel, k, cv::BORDER_DEFAULT);

    // Normalize 'scores'
    cv::normalize(corners, corners, 0, 255, cv::NORM_MINMAX,
         CV_32FC1, cv::Mat());
    cv::convertScaleAbs(corners, corners);

    ...
}
```

Harris corner detection in OpenCV


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    // Harris Detector parameters
    int block_size = 2;
    int size_sobel_kernel = 3; // 1, 3, 5 or 7
    double k = 0.1;

    // Detect corners using Harris corner detector
    cv::Mat corners = cv::Mat::zeros(img_binary.size(),CV_32FC1);
    cv::cornerHarris(img_binary, corners, block_size,
         size_sobel_kernel, k, cv::BORDER_DEFAULT);

    // Normalize 'scores'
    cv::normalize(corners, corners, 0, 255, cv::NORM_MINMAX,
         CV_32FC1, cv::Mat());
    cv::convertScaleAbs(corners, corners);

    ...
}
```


TU/e Technische Universiteit Eindhoven University of Technology

Draw detected corners on the RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    cv::Mat img_corners = img_rgb.clone(); // copy image
    unsigned char threshold_corners = 250; // in [0, 255]

    // Draw circles around corners
    for (int y = 0; y < corners.rows; ++y) {
        for (int x = 0; x < corners.cols; ++x) {
            if (corners.at<unsigned char>(y,x) >
              threshold_corners) {

                // Draw circle with center (x,y), radius of 5 and
                   blue line with thickness 2
                cv::circle(img_corners, cv::Point(x, y), 5,
                     cv::Scalar(255), 2);
            }
        }
    }

    ...
}
```


Technische Universiteit Eindhoven University of Technology

Draw detected corners on the RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    cv::Mat img_corners = img_rgb.clone(); // copy image
    unsigned char threshold_corners = 250; // in [0, 255]

    // Draw circles around corners
    for (int y = 0; y < corners.rows; ++y) {
        for (int x = 0; x < corners.cols; ++x) {
            if (corners.at<unsigned char>(y,x) >
              threshold_corners) {

                // Draw circle with center (x,y), radius of 5 and
                   blue line with thickness 2
                cv::circle(img_corners, cv::Point(x, y), 5,
                     cv::Scalar(255), 2);
            }
        }
    }

    ...
}
```

Draw detected corners on the RGB image


```
void imageCallback(const sensor_msgs::ImageConstPtr& color_img) {
    ...

    cv::Mat img_corners = img_rgb.clone(); // copy image
    unsigned char threshold_corners = 250; // in [0, 255]

    // Draw circles around corners
    for (int y = 0; y < corners.rows; ++y) {
        for (int x = 0; x < corners.cols; ++x) {
            if (corners.at<unsigned char>(y,x) >
              threshold_corners) {

                // Draw circle with center (x,y), radius of 5 and
                   blue line with thickness 2
                cv::circle(img_corners, cv::Point(x, y), 5,
                     cv::Scalar(255), 2);
            }
        }
    }

    ...
}
```

Further Reading

OpenCV website:

http://opencv.org/

http://docs.opencv.org/