Università di Padova – Facoltà di ingegneria

Corso di laurea in Ingegneria Informatica

CANNY EDGE DETECTOR

Professori: Pagello Enrico, Menegatti Emanuele

Studente: Maglie Andrea, matr. 456188

Anno Accademico 2005/2006

Indice

1 Canny Edge Detector **1**

2 Codice sorgente **3**

2.1 Canny.java 3

2.2 CannyShow.java 15

3 Esempi **23**

Capitolo 1

Canny Edge Detector

Nel 1986 John Canny individuò tre importanti caratteristice che un edge detector deve avere:

1. **Tasso di errore** - l'edge detector deve restituire solo i lati, e dovrebbe trovarli tutti.

2. **Localizzazione** - la distanza tra i pixel del lato trovato e del lato reale deve essere la minima possibile.

3. **Risposta** - l'edge detector non deve identificare più lati quando ne esiste solo uno.

L'edge detector è un filtro di convoluzione che smussa il rumore e localizza i lati; l'obiettivo è identificare il filtro che ottimizza i tre criteri sopra riportati. Ciò può essere raggiunto cercando il filtro che massimizza il prodotto **SNR*localizzazione**, dove SNR è il rapporto segnale/rumore e la localizzazione è un valore che rappresenta il reciproco della distanza del lato localizzato dal lato reale. Una buona approssimazione è data dalla derivata prima della funzione gaussiana:

$$G(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{1.1}$$

$$G'(x) = \left(-\frac{x}{\sigma^2}\right) e^{\left(-\frac{x^2}{2\sigma^2}\right)} \tag{1.2}$$

In due dimensioni la gaussiana è data da:

$$G(x,y) = \sigma^2 e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)} \tag{1.3}$$

la quale ha derivate sia lungo x che lungo y. Facendo la convoluzione tra un'immagine e G' otteniamo un'immagine con i lati evidenziati, anche in presenza di

rumore. La convoluzione con una gaussiana a due dimensioni può essere scomposta in due convoluzioni con una gaussiana unidimensionale. In conclusione, l'algoritmo di Canny consiste nei seguenti passi:

1. Leggere l'immagine I che dovrà essere processata.

2. Creare una maschera gaussiana unidimensionale G da convolvere con I. La deviazione standard è un parametro dell'algoritmo.

3. Creare una maschera unidimensionale per la derivata prima della gaussiana nelle direzioni x e y, ottenendo G_x e G_y.

4. Calcolare la convoluzione dell'immagine I con G lungo le righe per ottenere la componente I_x dell'immagine lungo x, e lungo le colonne per ottenere la componente I_y lungo y.

5. Calcolare la convoluzione di I_x con G_x per ottenere I'_x, cioè la componente lungo x di I convoluta con la derivata della gaussiana, e la convoluzione di I_y con G_y per ottenere I'_y.

6. Il modulo di ogni pixel dell'immagine risultante è dato da:

$$M(x, y) = \sqrt{I'_x(x, y)^2 + I'_y(x, y)^2} \tag{1.4}$$

7. Applicare la "non-maxima suppression", cioè la rimozione dei pixel che non costituiscono massimi locali. Infatti il modulo del gradiente del pixel di un lato è maggiore del modulo del gradiente dei pixel adiacenti e non facenti parti del lato.

8. Applicare la sogliatura con isteresi. Vengono usate una soglia alta T_h e una bassa T_l: ogni pixel avente un valore maggiore di T_h è assunto come facente parte di un lato; successivamente, ogni pixel collegato al precedente avente un valore più grande di T_l viene assunto come facente parte del lato.

Capitolo 2

Codice sorgente

2.1 Canny.java

```
// Classe Canny.java
// Implementa l'algoritmo di Canny.

import java.awt.*;
import java.awt.image.*;

class EdgeDetector extends Component
{
  // scale a 8 bit di modulo e direzione
  public static final double ORI_SCALE = 40.0D;
  public static final double MAG_SCALE = 20.0D;

  final float ORIENT_SCALE = 40F;
  private int height;
  private int width;
  private int picsize;
  private int data[];
  private int derivative_mag[];
  private int magnitude[];
  private int orientation[];
  private Image sourceImage;
  private Image edgeImage;
  // soglia superiore
  private int threshold1;
  // soglia inferiore
  private int threshold2;
  private int threshold;
```

```
  private int widGaussianKernel;
  private float sigma;
  int j1;

  // costruttore
  public EdgeDetector()
  {
    threshold1 = 10;
    threshold2 = 1;
    setThreshold(128);
    setGaussKernel(3);
    setSigma((float)1.0);
  }

  // processa l'immagine
  public void process() throws EdgeDetectorException
  {
    if (threshold < 0 || threshold > 255)
    {
      throw new EdgeDetectorException("Threshold out of range.");
    }

    if (widGaussianKernel < 3 || widGaussianKernel > 40)
    {
      throw new EdgeDetectorException("widGaussianKernel out of its range.");
    }

    width = sourceImage.getWidth(this);
    height = sourceImage.getHeight(this);
    picsize = width * height;
    data = new int[picsize];
    magnitude = new int[picsize];
    orientation = new int[picsize];

    canny(sigma, widGaussianKernel);

    thresholding(threshold1, threshold2);

    for (int i = 0; i < picsize; i++)
    {
      if (data[i] <= threshold)
```

```
      data[i] = 0xff000000;
    else
      data[i] = -1;
  }

  edgeImage = pixels2image(data);
  data = null;
  magnitude = null;
  orientation = null;
}

// Algoritmo di Canny
// i = gaussian kernel
private void canny(float f, int gkernel)
{
  boolean flag = false;
  boolean flag1 = false;

  derivative_mag = new int[picsize];

  float convy[] = new float[picsize];
  float convx[] = new float[picsize];
  // array delle medie gaussiane
  float meanGauss[] = new float[gkernel];
  float af5[] = new float[gkernel];
  float tmp1, tmp2, tmp3, tmp4, tmp5;
  float tmp6, tmp7, tmp8, tmp9, tmp10;
  float tmp11;

  data = image2pixels(sourceImage);

  int k4 = 0;

  // calcolo dei valori discreti
  // della distribuzione gaussiana
  do
  {
    System.out.println("k4 = "+k4);
    if (k4 >= gkernel)
      break;
```

```
    if (gauss(k4, f) <= 0.005F)
      break;

    // media gaussiana
    meanGauss[k4] = gauss(k4, f) + gauss((float) k4 - 0.5F, f)
                  + gauss((float) k4 + 0.5F, f);
    meanGauss[k4] = meanGauss[k4] / 3F / (6.283185F * f * f);
    af5[k4] = gauss((float) k4 + 0.5F, f) - gauss((float) k4 - 0.5F, f);
    k4++;
  }
  while (true);

  // convoluzione lungo x e lungo y con la gaussiana
  int j = k4;
  j1 = width - (j - 1);
  int l = width * (j - 1);
  int i1 = width * (height - (j - 1));

  for (int l4 = j - 1; l4 < j1; l4++)
  {
    for (int l5 = l; l5 < i1; l5 += width)
    {
      int k1 = l4 + l5;

      tmp1 = (float) data[k1] * meanGauss[0];
      tmp2 = tmp1;

      int l6 = 1;
      int k7 = k1 - width;

      for (int i8 = k1 + width; l6 < j; i8 += width)
      {
        tmp1 += meanGauss[l6] * (float) (data[k7] + data[i8]);
        tmp2 += meanGauss[l6] * (float) (data[k1 - l6] + data[k1 + l6]);
        l6++;
        k7 -= width;
      }
      // convoluzione lungo x con la gaussiana
      convy[k1] = tmp1;
      // convoluzione lungo y con la gaussiana
      convx[k1] = tmp2;
```

```
  }
}

// convoluzione dello smoothed con la derivata
float sconvy[] = new float[picsize];

for (int i5 = j - 1; i5 < j1; i5++)
{
  for (int i6 = l; i6 < i1; i6 += width)
  {
    tmp1 = 0.0F;
    int l1 = i5 + i6;
    for (int i7 = 1; i7 < j; i7++)
      tmp1 += af5[i7] * (convy[l1 - i7] - convy[l1 + i7]);

    sconvy[l1] = tmp1;
  }
}

convy = null;
float sconvx[] = new float[picsize];
for (int j5 = k4; j5 < width - k4; j5++)
{
  for (int j6 = l; j6 < i1; j6 += width)
  {
    tmp1 = 0.0F;
    int i2 = j5 + j6;
    int j7 = 1;
    for (int l7 = width; j7 < j; l7 += width)
    {
      tmp1 += af5[j7] * (convx[i2 - l7] - convx[i2 + l7]);
      j7++;
    }
    sconvx[i2] = tmp1;
  }

}
convx = null;

// non-maximal suppression
j1 = width - j;
```

```
l = width * j;
i1 = width * (height - j);
for (int k5 = j; k5 < j1; k5++)
{
  for (int k6 = l; k6 < i1; k6 += width)
  {
    int j2 = k5 + k6;
    int k2 = j2 - width;
    int l2 = j2 + width;
    int i3 = j2 - 1;
    int j3 = j2 + 1;
    int k3 = k2 - 1;
    int l3 = k2 + 1;
    int i4 = l2 - 1;
    int j4 = l2 + 1;
    tmp1 = sconvy[j2];
    tmp2 = sconvx[j2];
    float f12 = modulus(tmp1, tmp2);

    int k = (int) ((double) f12 * MAG_SCALE);

    if ( k >= 256 )
      derivative_mag[j2] = 255;
    else
      derivative_mag[j2] = k;

    tmp3 = modulus(sconvy[k2], sconvx[k2]);
    tmp4 = modulus(sconvy[l2], sconvx[l2]);
    tmp5 = modulus(sconvy[i3], sconvx[i3]);
    tmp6 = modulus(sconvy[j3], sconvx[j3]);
    tmp7 = modulus(sconvy[k3], sconvx[k3]);
    tmp8 = modulus(sconvy[l3], sconvx[l3]);
    tmp9 = modulus(sconvy[i4], sconvx[i4]);
    tmp10 = modulus(sconvy[j4], sconvx[j4]);
    boolean vabene = false;

    // se y*x <= 0 (ci troviamo nel secondo o quarto quadrante)
    if ( tmp1 * tmp2 <= 0 )
    {
      //se y >= x
      if ( Math.abs(tmp1) >= Math.abs(tmp2) )
```

```
    {

      if ( Math.abs(tmp1 * f12) >=
           Math.abs(tmp2 * tmp8 - (tmp1 + tmp2) * tmp6)
           && Math.abs(tmp1 * f12) >
           Math.abs(tmp2 * tmp9 - (tmp1 + tmp2) * tmp5) )
      {
        vabene = true;
      }
      else vabene = false;
    }
    else if ( Math.abs(tmp2 * f12) >=
              Math.abs(tmp1 * tmp8 - (tmp2 + tmp1) * tmp3)
              && Math.abs(tmp2 * f12) >
              Math.abs(tmp1 * tmp9 - (tmp2 + tmp1) * tmp4) )
    {
        vabene = true;
    }
    else vabene = false;
  }
  else
  {
    if ( Math.abs(tmp1) >= Math.abs(tmp2) )
    {
      if ( Math.abs(tmp1 * f12) >=
           Math.abs(tmp2 * tmp10 + (tmp1 - tmp2) * tmp6)
           && Math.abs(tmp1 * f12) >
           Math.abs(tmp2 * tmp7 + (tmp1 - tmp2) * tmp5) )
      {
        vabene = true;
      }
      else vabene = false;
    }
    else if ( Math.abs(tmp2 * f12) >=
              Math.abs(tmp1 * tmp10 + (tmp2 - tmp1) * tmp4)
              && Math.abs(tmp2 * f12) >
              Math.abs(tmp1 * tmp7 + (tmp2 - tmp1) * tmp3) )
    {
        vabene = true;
    }
    else vabene = false;
```

```
      }
      if ( vabene )
      {
        magnitude[j2] = derivative_mag[j2];
        orientation[j2] = (int) (Math.atan2(tmp2, tmp1) * ORI_SCALE);
      }
    }
  }

  derivative_mag = null;
  sconvy = null;
  sconvx = null;
}

// ritorna 0 se sono entrambi zero,
// altrimenti ritorna il modulo
private float modulus(float f, float f1)
{
  if (f == 0.0F && f1 == 0.0F)
    return 0.0F;
  else
    return (float) Math.sqrt(f * f + f1 * f1);
}

// funzione gaussiana
private float gauss(float f, float f1)
{
  return (float) Math.exp((-f * f) / ((float) 2 * f1 * f1));
}

// thresholding con isteresi
private void thresholding(int i, int j)
{
  if ( i < j )
  {
    System.out.println("Errore: soglia superiore < soglia inferiore!");
  }
  else
  {
    for (int k = 0; k < picsize; k++)
      data[k] = 0;
```

```
    // per ogni lato con magnitude maggiore della soglia superiore
    // traccia i lati che sono maggiori della soglia inferiore
    for (int l = 0; l < width; l++)
    {
      for (int i1 = 0; i1 < height; i1++)
        if (magnitude[l + width * i1] >= i)
          linking(l, i1, j);
    }
  }
}

// k è la soglia inferiore
private boolean linking(int i, int j, int k)
{
  j1 = i + 1;
  int k1 = i - 1;
  int l1 = j + 1;
  int i2 = j - 1;
  int j2 = i + j * width;
  if (l1 >= height)
    l1 = height - 1;
  if (i2 < 0)
    i2 = 0;
  if (j1 >= width)
    j1 = width - 1;
  if (k1 < 0)
    k1 = 0;

  if (data[j2] == 0)
  {
    data[j2] = magnitude[j2];
    boolean flag = false;
    int l = k1;

    do
    {
      if (l > j1)
        break;
      int i1 = i2;
      do
```

```
      {
        if (i1 > l1)
          break;
        int k2 = l + i1 * width;
        if ( (i1 != j || l != i) && magnitude[k2] >= k && linking(l, i1, k) )
        {
          flag = true;
          break;
        }
        i1++;
      }
      while (true);

      if (!flag)
        break;
      l++;
    }
    while (true);
    return true;
  }
  else
  {
    return false;
  }
}

private Image pixels2image(int ai[])
{
  MemoryImageSource memoryimagesource = new MemoryImageSource(
      width,
      height,
      ColorModel.getRGBdefault(),
      ai,
      0,
      width);

  return Toolkit.getDefaultToolkit().createImage(memoryimagesource);
}

private int[] image2pixels(Image image)
{
```

```
int ai[] = new int[picsize];
PixelGrabber pixelgrabber =
new PixelGrabber(image, 0, 0, width, height, ai, 0, width);
try
{
  pixelgrabber.grabPixels();
}
catch (InterruptedException interruptedexception)
{
  interruptedexception.printStackTrace();
}
boolean flag = false;
int k1 = 0;
do
{
  if (k1 >= 16)
    break;
  int i = (ai[k1] & 0xff0000) >> 16;
  int k = (ai[k1] & 0xff00) >> 8;
  int i1 = ai[k1] & 0xff;
  if (i != k || k != i1)
  {
    flag = true;
    break;
  }
k1++;
}
while (true);
if (flag)
{
  for (int l1 = 0; l1 < picsize; l1++)
  {
    int j = (ai[l1] & 0xff0000) >> 16;
    int l = (ai[l1] & 0xff00) >> 8;
    j1 = ai[l1] & 0xff;
    ai[l1] =
      (int) (0.29799999999999999D * (double) j
        + 0.58599999999999997D * (double) l
        + 0.113D * (double) j1);
  }
```

```
    }
    else
    {
      for (int i2 = 0; i2 < picsize; i2++)
        ai[i2] = ai[i2] & 0xff;
    }
    return ai;
  }

  public void setSourceImage(Image image)
  {
    sourceImage = image;
  }

  public Image getEdgeImage()
  {
    return edgeImage;
  }

  public void setThreshold(int i)
  {
    threshold = i;
    System.out.println("Threshold: " + i );
  }

  public void setHighThreshold(int i)
  {
    threshold1 = i;
    System.out.println("High Threshold: " + i );
  }

  public void setLowThreshold(int i)
  {
    threshold2 = i;
    System.out.println("Low Threshold: " + i );
  }

  public void setGaussKernel( int i )
  {
    widGaussianKernel = i;
    System.out.println("Gauss kernel: " + i );
```

```
  }

  public void setSigma( float i )
  {
    sigma = i;
    System.out.println("sigma: " + i );
  }
}
```

2.2 CannyShow.java

```
// Classe CannyShow.java
// Implementa l'interfaccia grafica per l'algoritmo di Canny.

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.swing.event.*;


public class CannyShow extends JApplet
{

  public static EdgeDetector edgeDetector;
  public static JComboBox gaussKerBox;
  public static JPanel pan;
  public static JPanel pan4;
  public static JLabel outputLabel;
  public static JLabel outputImage;
  public static JTextField highthresh;
  public static JTextField lowthresh;
  public static JTextField sigma;
  public static JFrame frame1;
  public static String imageFile = "golf.gif";
  public static JTextField inputImg;
  public static JButton loadImage;
  public static String image_url;
  public static URL theURL;
  public static Image image1;
```

```
public static Toolkit tool;
public static JLabel inputImage;
public static JPanel pan3;
public static JPanel pan2;
public static JPanel pan2_1;
public static JPanel pan3_1;
public static JLabel inputLabel;
public static JLabel kerlabel = new JLabel( "Kernel" );
public static JLabel highlabel = new JLabel("Soglia superiore");
public static JLabel lowlabel = new JLabel("Soglia inferiore");
public static JLabel sigmalabel = new JLabel("Deviazione standard");

public static void kerSize()
{
    int index = gaussKerBox.getSelectedIndex();
    if ( index == 0 )
    {
      edgeDetector.setGaussKernel(3);
    }
    else if ( index == 1 )
    {
      edgeDetector.setGaussKernel(4);
    }
    else if ( index == 2 )
    {
      edgeDetector.setGaussKernel(5);
    }
    else if ( index == 3 )
    {
      edgeDetector.setGaussKernel(6);
    }
    else if ( index == 4 )
    {
      edgeDetector.setGaussKernel(7);
    }
    else if ( index == 5 )
    {
      edgeDetector.setGaussKernel(8);
    }
    else if ( index == 6 )
    {
```

```
      edgeDetector.setGaussKernel(9);
    }
    else if ( index == 7 )
    {
      edgeDetector.setGaussKernel(10);
    }
    else if ( index == 8 )
    {
      edgeDetector.setGaussKernel(12);
    }
    else if ( index == 9 )
    {
      edgeDetector.setGaussKernel(15);
    }
    else if ( index == 10 )
    {
      edgeDetector.setGaussKernel(20);
    }
}

public static void loadImage()
{
  pan3.remove(inputImage);
  pan4.remove(outputImage);

  imageFile = inputImg.getText();
  System.out.println("Carico file "+imageFile);
  image1 = tool.getImage(imageFile);
  image1 = image1.getScaledInstance(300, 300, Image.SCALE_DEFAULT);

  inputImage = new JLabel(new ImageIcon(image1));
  inputImage.setPreferredSize(new Dimension(300, 300));

  outputImage = new JLabel(new ImageIcon(image1));
  outputImage.setPreferredSize(new Dimension(300, 300));

  edgeDetector.setSourceImage(image1);

  pan3.add(inputImage);
  pan3.repaint();
  pan4.add(outputImage);
```

```
    pan4.repaint();
    pan.repaint();
    frame1.setContentPane(pan);
    frame1.pack();
    frame1.show();
  }

  public static void showOutputImage ( Image img, JPanel pane )
  {
    pane.remove(outputImage);
    outputImage = new JLabel(new ImageIcon(img));
    outputImage.setPreferredSize(new Dimension(300, 300));
    pane.add(outputImage);
    pane.repaint();
    pan.repaint();
    frame1.setContentPane(pan);
    frame1.pack();
    frame1.show();
  }

  public static void startCanny()
  {
    edgeDetector.setHighThreshold(
              Integer.valueOf(highthresh.getText()).intValue());
    edgeDetector.setLowThreshold(
              Integer.valueOf(lowthresh.getText()).intValue());
    edgeDetector.setSigma(
              Float.valueOf(sigma.getText()).floatValue());

    // processa l'immagine
    try {
      edgeDetector.process();
    }
    catch(EdgeDetectorException e) {
      System.out.println(e.getMessage());
    }

    Image edgeImage = edgeDetector.getEdgeImage();

    showOutputImage ( edgeImage, pan4 );
  }
```

```
public static void main (String [] args)
{
  JButton startButton = new JButton("Avvia processo");
  startButton.setAlignmentY(CENTER_ALIGNMENT);
  startButton.setAlignmentX(CENTER_ALIGNMENT);

  edgeDetector = new EdgeDetector();

  // immagine in input
  tool = Toolkit.getDefaultToolkit();
  image1 = tool.getImage(imageFile);
  image1 = image1.getScaledInstance(300, 300, Image.SCALE_DEFAULT);
  edgeDetector.setSourceImage(image1);
  inputLabel = new JLabel("Immagine in input");

  inputImg = new JTextField("golf.gif",10);
  loadImage = new JButton("Carica immagine");
  loadImage.addActionListener(new ActionListener()
            {
              public void actionPerformed(ActionEvent e)
              {
                loadImage();
              }
            });

  // frame principale
  frame1 = new JFrame ("Canny Edge Detector");
  frame1.setPreferredSize(new Dimension(900, 500));
  frame1.setDefaultCloseOperation(frame1.EXIT_ON_CLOSE);

  // pannelli
  pan = new JPanel();
  pan.setPreferredSize(new Dimension(800, 500));
  pan.setLayout(new BoxLayout(pan, BoxLayout.PAGE_AXIS));
  pan2 = new JPanel();
  pan2.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
  pan2_1 = new JPanel();
  pan2_1.setLayout(new BoxLayout(pan2_1, BoxLayout.LINE_AXIS));
  pan2_1.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
  pan2_1.setAlignmentY(CENTER_ALIGNMENT);
```

```
pan2_1.setAlignmentX(CENTER_ALIGNMENT);
pan3_1 = new JPanel();
pan3_1.setLayout(new BoxLayout(pan3_1, BoxLayout.LINE_AXIS));
pan3_1.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
pan3 = new JPanel();
pan3.setPreferredSize(new Dimension(320, 500));
pan3.setLayout(new BoxLayout(pan3, BoxLayout.PAGE_AXIS));
pan3.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
pan4 = new JPanel();
pan4.setPreferredSize(new Dimension(320, 500));
pan4.setLayout(new BoxLayout(pan4, BoxLayout.PAGE_AXIS));
pan4.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

// immagine di input
inputImage = new JLabel(new ImageIcon(image1));
inputImage.setPreferredSize(new Dimension(300, 300));

// immagine di output
outputImage = new JLabel(new ImageIcon(image1));
outputImage.setPreferredSize(new Dimension(300, 300));

// combo selezione kernel
gaussKerBox = new JComboBox();
gaussKerBox.addItem("3x3");
gaussKerBox.addItem("4x4");
gaussKerBox.addItem("5X5");
gaussKerBox.addItem("6x6");
gaussKerBox.addItem("7x7");
gaussKerBox.addItem("8x8");
gaussKerBox.addItem("9x9");
gaussKerBox.addItem("10x10");
gaussKerBox.addItem("12x12");
gaussKerBox.addItem("15x15");
gaussKerBox.addItem("20x20");

gaussKerBox.addActionListener(new ActionListener()
          {
            public void actionPerformed(ActionEvent e)
            {
              kerSize();
            }
```

```
            });

// input soglia superiore
highthresh = new JTextField("10",5);

// input soglia inferiore
lowthresh = new JTextField("1",5);

// input sigma
sigma = new JTextField("1.0",5);

pan2.add(loadImage);
pan2.add(inputImg);
pan2.add(kerlabel);
pan2.add(gaussKerBox);
pan2.add(highlabel);
pan2.add(highthresh);
pan2.add(lowlabel);
pan2.add(lowthresh);
pan2.add(sigmalabel);
pan2.add(sigma);
pan2_1.add(startButton);
pan3.add(inputLabel);
pan3.add(inputImage);

// action listener del pulsante di avvio
startButton.addActionListener(new ActionListener()
            {
              public void actionPerformed(ActionEvent e)
              {
                startCanny();
              }
            });

outputLabel = new JLabel("Immagine in output");
pan4.add(outputLabel);
pan4.add(outputImage);
pan.add(pan4);
pan3_1.add(pan3);
pan3_1.add(pan4);
pan.add(pan2);
```

```
    pan.add(pan2_1);
    pan.add(pan3_1);
    frame1.setContentPane(pan);
    frame1.pack();
    frame1.show();
  }
}
```

Capitolo 3

Esempi

L'applicazione viene avviata tramite il comando `java CannyShow`; l'interfaccia grafica è riportata in figura 3.1. Come si può vedere, i parametri impostabili sono: dimensione del kernel, soglia superiore, soglia inferiore e deviazione standard. Una volta caricato il file immagine e impostati i parametri desiderati, è sufficente cliccare sul pulsante *Avvia processo* per iniziare la computazione dell'algoritmo di Canny. Non appena la computazione è completa, l'immagine risultante verrà visualizzata nella parte destra (indicata con *Immagine in output*). In figura 3.2 c'è un esempio di utilizzo dell'applicazione con i seguenti parametri:

- Kernel: 3x3
- Soglia Superiore: 50
- Soglia Inferiore: 20
- Deviazione Standard: 1.0

Nell'esempio di figura 3.3 sono stati utilizzati invece i seguenti valori per i parametri:

- Kernel: 10x10
- Soglia Superiore: 50
- Soglia Inferiore: 20
- Deviazione Standard: 1.5

Ulteriori esempi sono riportati in figura 3.4, figura 3.5 e figura 3.6.

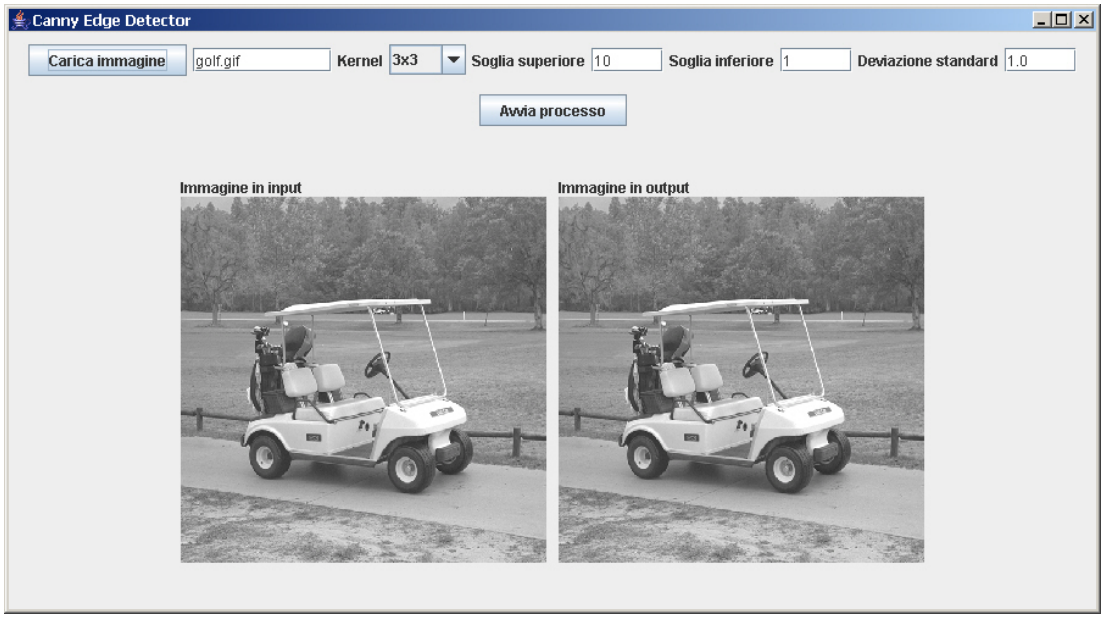

Figura 3.1

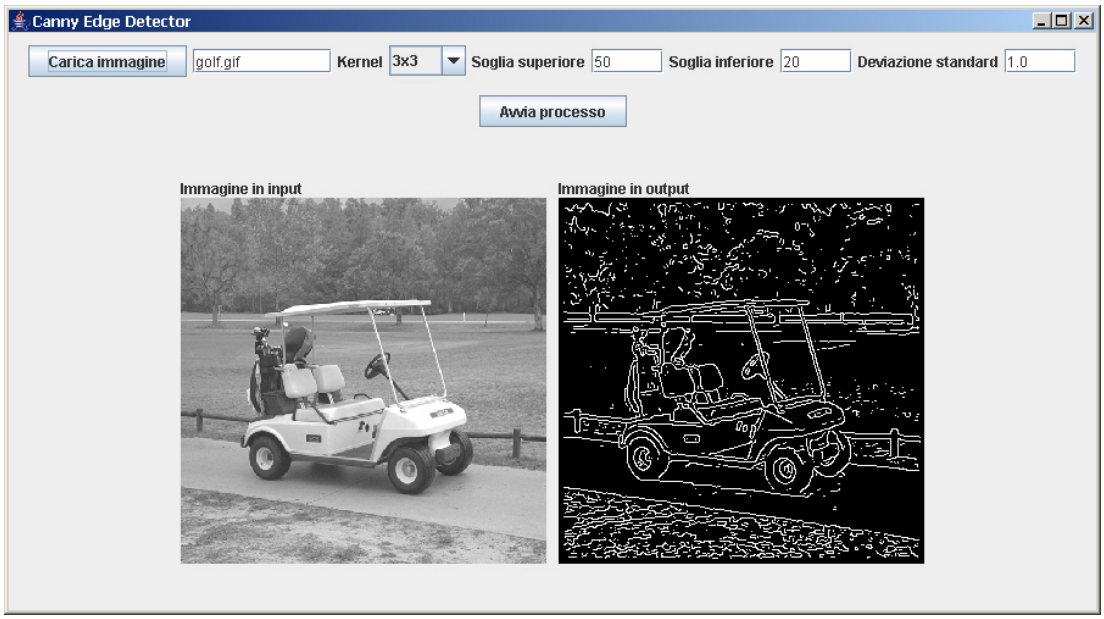

Figura 3.2

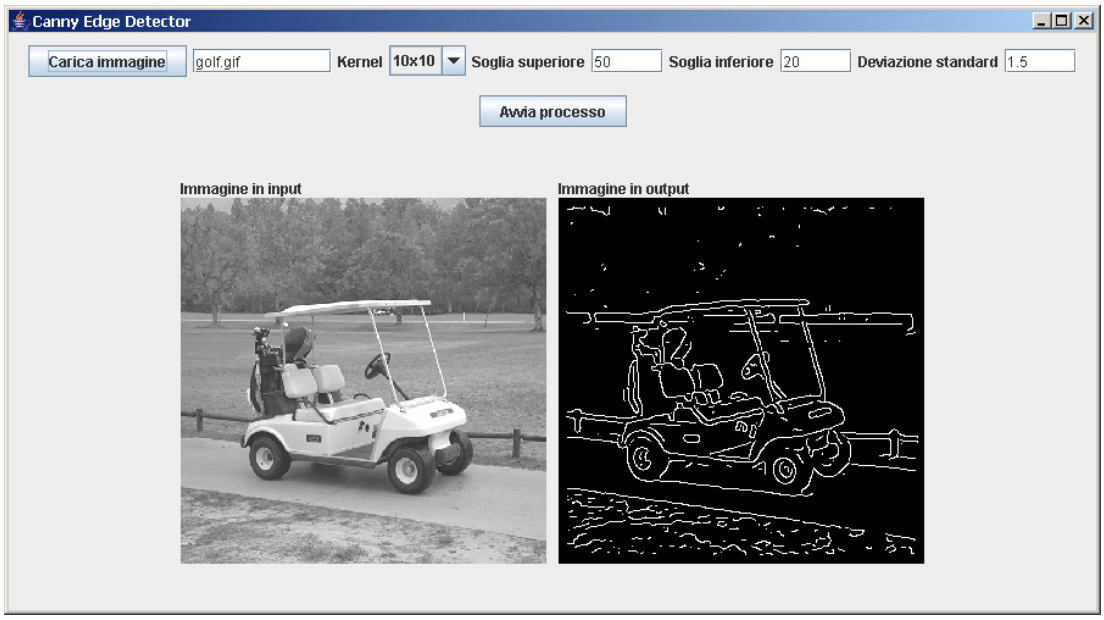

Figura 3.3

Figura 3.4: *Parametri utilizzati: kernel 3x3, soglia superiore 100, soglia inferiore 50, deviazione standard 1.0.*

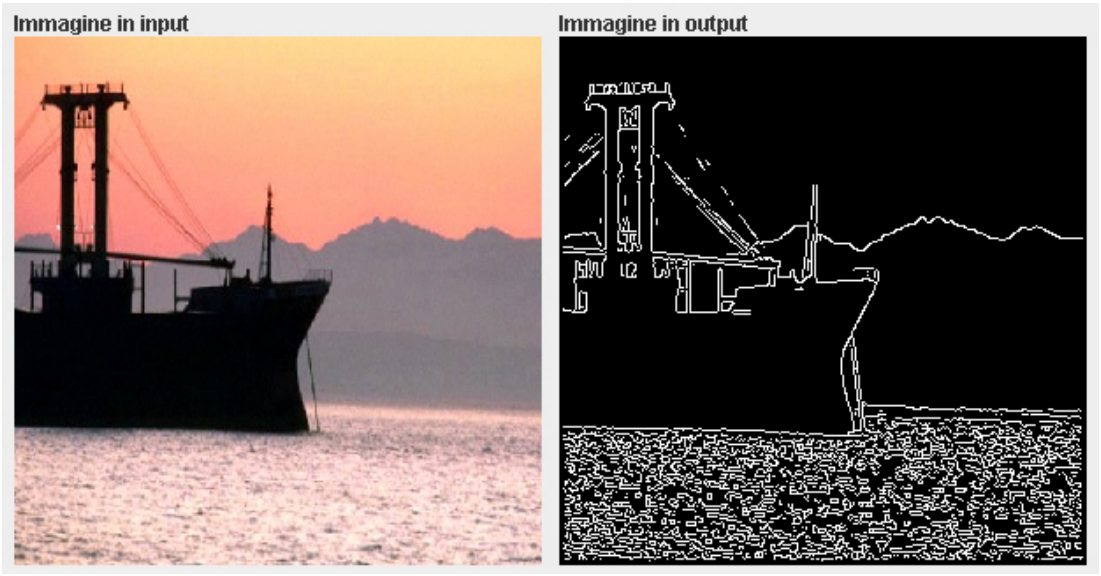

Figura 3.5: *Parametri utilizzati: kernel 10x10, soglia superiore 200, soglia inferiore 100, deviazione standard 0.45.*

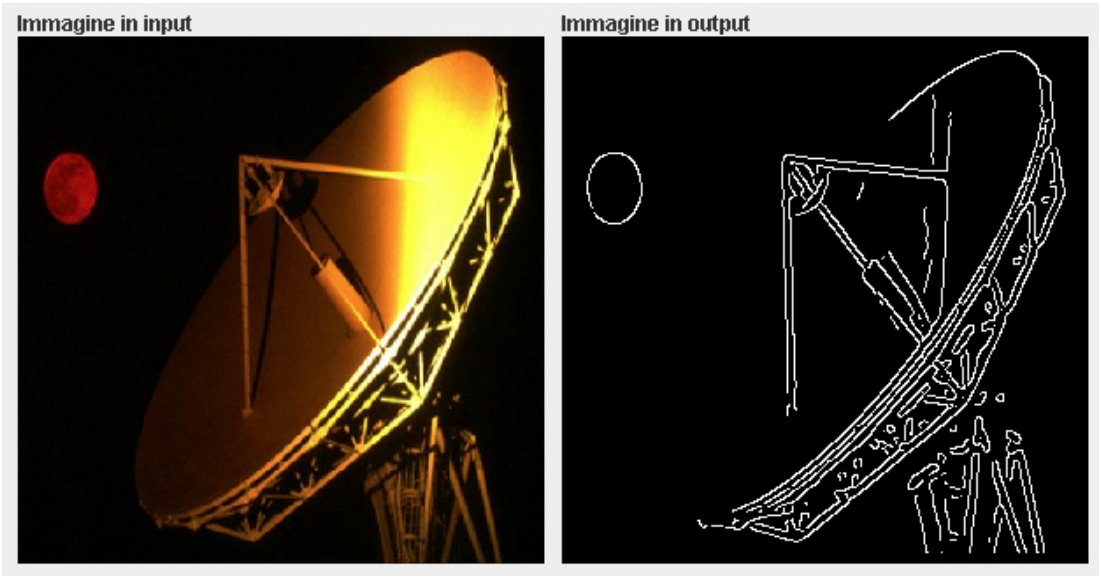

Figura 3.6: *Parametri utilizzati: kernel 7x7, soglia superiore 50, soglia inferiore 10, deviazione standard 1.6.*