Detection of the Line of Sight

Maria Estela Orozco Ochoa

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
MEDELLÍN
2004

Detection of the Line of Sight

Maria Estela Orozco Ochoa

Final project presented to obtain the B.Sc. Diploma in Computer Science

Adviser
Prof. Dr. Oscar Ruiz S.

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
MEDELLÍN
2004

Acceptation note

Jury: Dr. Ursula Kretschmer

Jury: Dr. Juan Guillermo Lalinde

Adviser: Dr. Oscar Ruiz

Medellín, February 9^{th}, 2004

Acknowledgments

I want to thank Dr. Oscar Ruiz for his support and guidance throughout this work.

Special thanks to Dr. Uwe Jasnoch and Dr. Ursula kretschmer for having given me the opportunity to join the *GEIST* project at The Fraunhofer Institute (FhG-IGD).

Thanks to Eliana Vásquez for her advice and help.

I want to thank the students of the CAD/CAM/CAE laboratory, my parents and my colleagues at Dinario.

CONTENTS

List of Figures

List of Algorithms

Introduction

This project consists of an image feature extractor in order to detect the line of sight of a person. The detected features are edges and corners of an image. Therefore, the extractor is divided in two parts: an edge and a corner extractor.

The implemented edge and corner extractor theory was taken from [2] and [12] respectively. Based on this, an enhanced version of the edge extractor was developed. Also, the corner extractor was replaced by a corner calculation process.

This feature extractor is part of the tracking module of the *GEIST* project, still in process in the A5 division of The Fraunhofer Institute FhG-IGD.

This project was done under a research cooperation frame between The Fraunhofer Institute for Computer Graphics (FhG-IGD), Darmstadt, Germany, and EAFIT University, Medellín, Colombia. Part of this project was developed during an internship at Fraunhofer Institute in the Geographical Information Systems division (IGD-A5) during the fist semester of 2003, under the supervision of Dr. Ursula Kretschmer. The rest of the time, the project was developed in the CAD/CAM/CAE Laboratory at EAFIT University, under the supervision of Dr. Oscar E. Ruiz.

Chapter 1

Context

This chapter presents an introduction to the project in which this work is circumscribed: the *GEIST* project. Then, important topics to understand the context of the problem are explained. Such topics are: Augmented Reality (AR) and tracking. Finally an overview of the implemented algorithms and the data structures is given.

1.1 The *GEIST* project

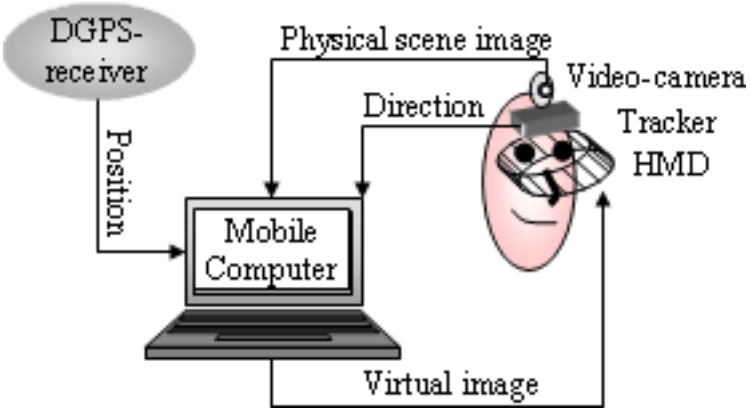
The *GEIST* project [1] is an educational game that aims to transmit historical facts both to youngsters and adults by means of AR. The game consists of following a sequence of interesting places where the history of a town is told, and where through the presence of ghosts, the user is invited to solve tasks and to inquire about the history of the environment. Game sequences are different for all users, they depend on the user's own decisions, suggestions and solved tasks.
The game uses reconstruction of buildings as they were during the Thirty Years' War in Heidelberg. Such reconstructions are superimposed in a user's viewing field. AR and tracking are applied to achieve all this. Thus, at the entrance to the site, the visitors are provided with a small mobile computer unit that they carry around during their stay. The system consists of a see-through head mounted display (HMD), a GPS, an orientation tracker and a video camera attached to a mobile computer. These elements are used to superimpose the virtual views on the user's current field of view (figure 1.1). Thus, the perception of physical reality is enhanced by computer-generated overlays displayed on the HMD.
The *GEIST* project is being developed in A5 division (Geographic information Systems (GIS)) at The Fraunhofer Institute for Computer Graphics (FhG-IGD), Darmstadt, Germany.

[1] The term *geist* is the German expression for the English word *ghost*

1.1.1 Augmented Reality

AR is a technology used to enrich a user's view of the real world with additional information. Its aim is to enhance visual perception allowing the superposition of virtual objects to the physical world. It works by combining live video input with immersive display technology.
AR allows users to work with and examine real three dimensional (3D) objects while seeing additional information about the objects or the task at hand. Alignment of 3D viewing parameters between real and virtual worlds is done to video-camera and user see-through HMD by means of calibration and tracking. An example of a video-based tracking system for AR is shown in Figure 1.1.

Figure 1.1: Tracking system for AR



By enrichening visual perception, AR focuses the user's attention on objects of interest. Such objects depend on the application area. In medicine, Doctors could use AR as a visualization and training aid for surgery, virtual instructions could remind a novice surgeon of the required steps, without the need to look away from a patient to consult a manual. In architecture, AR might aid general visualization tasks as well, architects might be able to get "X-ray vision" of a building's structure showing where the pipes, electric lines, and structural supports are inside the walls. In the assembly, maintenance, and repair of complex machinery, instructions might be easier to understand if they are available, not as a manual with text and pictures, but rather as 3D drawings superimposed upon the actual equipment. Depending on the application scenario, AR systems need to access relevant information and present it appropriately, e.g. text-based annotations, a wireframe or complete reconstructions of physical scenes.

1.1.2 Tracking

Tracking is the process of finding the position and orientation of a subject with respect to an object of interest. Tracking may be based on visual information, sensors or a mixture of both. In the following paragraphs these methods will be explained and the tracking approach of the *GEIST* project is introduced.

Tracking based on Visual Information

It is based upon the capture of images of the surrounding environment by a camera. It is also called optical tracking. Optical tracking may be divided into active and passive.

i. Active optical tracking. It is done in prepared environments where visible pre-calibrated landmarks are captured by the camera.

ii. Passive optical tracking. It is done in unprepared environments. The tracking is based solely on the analysis of images captured by the camera. This approach is more complicated than the previous one, but it broadens the field of application to outdoor environments.

Tracking based on Sensors

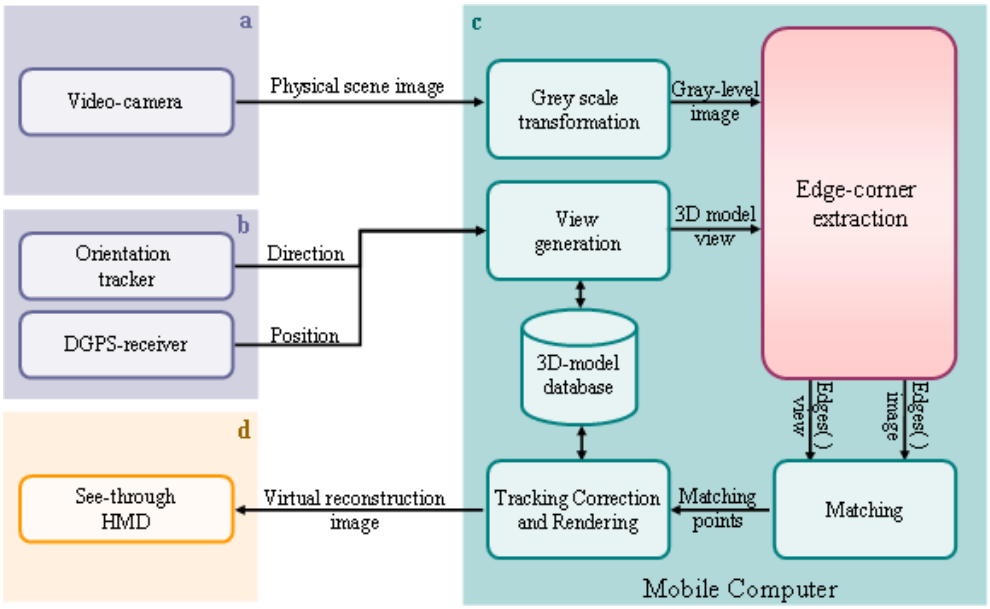
Sensors are devices that measure yaw, pitch, roll and direction of a person or object. Gyroscopes, inclinometeres, compasses are examples of sensors.
Global Positioning System (GPS) is a satellite navigation system for determining position on the Earth's surface by comparing radio signals from several satellites. A DGPS-receiver samples data from up to six satellites, it then calculates the time taken for each satellite signal to reach the DGPS-receiver, and from the difference in time of reception, determines the location. DGPS has an accuracy of up to 50 cm in case the signal of more than three satellites can be gathered by the DGPS-receiver.

Hybrid Tracking

Hybrid tracking grew up of the need to overcome the weaknesses of both: sensors and optical tracking. Tracking sensors are subject to signal noise, degradation with distance and sources of interference. Optical tracking offers accurate, passive, low cost pose estimation. Also, it detects, measures and reduces pose tracking errors derived from other technologies (i.e. sensors). But, optical tracking is time consuming, while sensor trackers are fast at giving pose estimation. Thus, the fusion of optical and sensor trackers to exploit the complementary properties of the two technologies offers promising possibilities for tracking.

Tracking for the *GEIST* project

Figure 1.2: Tracking workflow of the *GEIST* project



The hybrid tracking for the *GEIST* project is based on optical and tracking sensors (Figure 1.2). The passive optical tracking approach is based on the real-time registration of the live video-frames captured by the camera and the reference-views from a 3D model database of the surroundings. Once the matching between live video-image and reference-view has been established, the virtual information can be overlaid correctly onto the user's line of sight. The approach begins with an initial pose estimation based on measures from the DGPS and the orientation tracker. These estimates are used to query a 3D model view generator in order to obtain a reference-view similar to that of the user's line of sight. This view is processed to extract edges and corners which will be fed into a Matching Algorithm along with the edges and corners extracted from the video-frame of the camera. In case they match, accurate position and orientation are calculated for the user's line of sight, based on the known parameters of the reference 3D model view. Figure 1.2 illustrates the tracking workflow of the *GEIST* project. In the following paragraphs each block of Figure 1.2 will be explained.

i. Video camera. It captures images of the environment in real time. It is used for passive optical tracking of the user's head (Label a, Figure 1.2).

ii. Orientation tracker. It determines yaw, pitch and roll of the user's head(Label b, Figure 1.2).

iii. DGPS-receiver. It determines an approximate position of user (Label b, Figure 1.2).

iv. See-through HMD. Lenses to display virtual objects in the user's line of sight (Label d, Figure 1.2).

v. Grey scale transformation. Pre-processing step to transform a color image into a gray scale image (Label c, Figure 1.2).

vi. View generation. Process that takes the pose estimates from sensors as input, and generates a reference view from the 3D-model database. The generated view is similar to that of the user's line of sight (Label c, Figure 1.2).

vii. 3D-model database. Database that stores a 3D-model of the city of Heidelberg (Label c, Figure 1.2).

viii. Edge-corner Extraction. Process to detect edges and corners from grey scale images coming from the video-camera and from reference views of the 3D-model database. The Edge-corner Extraction step is the subject of this thesis (Label c, Figure 1.2).

ix. Matching. Process to match a pair of images coming from the previous Edge-corner Extraction step (Label c, Figure 1.2).

x. Tracking correction. Process that enhances accuracy in measure of user's pose by results obtained from the Matching Algorithm. It generates a 3D-model view of a virtual reconstruction to be displayed on the see-through HMD (Label c, Figure 1.2).

1.2 Overview of the Edge and Corner Extraction Algorithm Implemented

Existing algorithms for identification of edges and corners work on pixel domain. However, matching algorithms of the *GEIST* project require edges in vector form. In order to identify and vectorize straight edges and corners from an image the following conceptual steps are undertaken:

i. Identification of Edges in Pixel Domain.
This section partitions the set of dark pixels from an image in subsets, which represent (straight or curved) Edges. At a particular step of the algorithm filtering must be applied. Usually, two dimensional (2D) filters are applied for Edge detection. In the present work, for the sake of speed and simplicity, the application of one-dimesional (1D) filters was proposed and succesfully implemented. This is

an original contribution of the present work. It must be said that in either case (applying 1D or 2D filters), corners are erased from the image. Therefore the process of *direct* corner identification below must re-take the original image.

ii. Edge Vectorization.
This process takes sequences of pixels which are generalized (i.e. straight or curved) edges and splits them into the largest possible sub-chains which are approximately straight.

iii. Corner Extraction
This stage must produce the corners, which are junctions of two or more straight edges. Two possible approaches are possible (the both of them impemented in this work)

(a) Direct Corner Extraction.
This process acts directly on the original image (not filtered) by applying heurisctics to identify regions which have characteristics of corners. The algorithm implemented for the application of such heuristics is the S.U.S.A.N. [12]. Direct Corner Extraction requires further processing (described later) to match detected corners and edges. This process resulted in large computational burden, therefore prompting the generation of Indirect Corner Extraction, which follows.

(b) Indirect Corner Extraction
This step implemented the direct calculation of corners as intersections of straight edges resulting form step 2 above. This contribution of the present work showed to be fast and effective and it is currently used in the follow-up of the *Geist* project.

iv. Corner - Edge matching
This step searches in two sets -corners and edges- to find corners that are likely to be the endpoints of the edges. This screening was based upon an approximate inclusion test of a point on an infinite line. In practice, however, this process resulted expensive and numerically unstable.

1.2.1 Data Structures

The data structures used are: (i) TIFF files for input, and (ii) Edge Array and the user defined Class BC_Mesh2D for output. They will be explained further in the following numerals.

i. TIFF file. TIFF files are used for storing and interchanging raster images. Input information for this work are TIFF files. Basic information on this file format is shown in Table 1.1. (Additional Information on TIFF file format is found in [1]).

Tabla 1.1: TIFF file format

File:	TIFF Revision 6
Width:	640
Height:	480
Bits per plane:	8
Number of planes:	1
Number of gray levels:	256

ii. Edges Array (*E.A.*). It is an array (EA), where each row holds the 2D coordinates of the initial and final vertices of an edge. Edges are represented by a pair of vertices or endpoints. The pixel coordinates of both vertices are stored in array EA. Each row of EA holds in columns 0 and 1 the x,y (column and row) coordinates of initial vertex and positions 2 and 3 of EA store the x, y (column and row) coordinates of final vertex. Table 1.2 shows a representation of the array EA.

Tabla 1.2: EA representation

Size of EA:	$4 \times n$
EA[0][i]:	column of initial vertex of the i-th edge
EA[1][i]:	row of initial vertex of the i-th edge
EA[2][i]:	column of final vertex of the i-th edge
EA[3][i]:	row of final vertex of the i-th edge

Where n is the total number of edges obtained from the Edge Vectorization process applied to an image.

iii. Class BC_Mesh2D. It is a class that belongs to the package *de.fhg.igd.progis.geist.imagepreparation*. It stores edges in two data structures: (i) Edge array and (ii) Vertex array. The class BC_Mesh2D is used as input to the Matching Algorithm. The class prototype is shown in Apendix A.

Chapter 2

Glossary and Definitions

2.1 Glossary

1D	one dimensional
2D	two dimensional
3D	three dimensional
AR	augmented reality
DIP	digital image processing
DGPS	differential global positioning system
GPS	global positioning system
HMD	head mounted display
SUSAN	smallest univalue segment assimilating nucleus
USAN	univalue segment assimilating nucleus

2.2 Definitions

2.2.1 Basics

In this section the basic terms of image processing needed for this work are defined.

i. Live video-frame
It is an image *I()* captured by the video-camera.

ii. Reference-view
It is an image *I()* coming from the 3D model database. It is called reference-view, because the measures of orientation and position of a user can be calculated based on this image.

iii. Pose
Position and orientation of a person, or parts of a person's body, with respect to an object of interest.

iv. Coordinate System
A 2D coordinate system with origin at the top-left corner of the plane of the image. The horizontal axis is labeled x and it points to the right. The vertical axis is labeled y and it points downwards.

v. Image $I()$
Given $I : Z \times Z \longrightarrow N$
Domain $[0, X_c] \times [0, Y_r]$
Where X_c : Column pixels, usually 639; Y_r : Row pixels, usually 479.
Range of *I( )* function is $[B_L, W_L]$, where
B_L : Black level (usually 0); W_L : White level (usually 255).
An image is a rectangular array of pixels, each one of them representing a grey scale color dot. The (i, j) position (with $0 \leq i \leq X_c$ and $0 \leq j \leq Y_r$) contains an integer between B_L and W_L.

vi. Vertex or Pixel (i, j), v
A vertex is a position (i, j) within an image *I()*. A vertex will refer to both: (i) its position (i, j) and, (ii) its content $v(i, j)$.
$(i, j) \in [0, X_c] \times [0, Y_r]$
$v(i, j) \in [B_L, W_L]$
I(v) or *I(i,j)* represents the intensity of the image at pixel v or (i, j).

vii. Straight line $\overleftrightarrow{v_i v_j}$
It is the infinite weighted combination of two vertices v_i, v_j.
$\overleftrightarrow{v_i v_j} = (\lambda)\, v_i + (1 - \lambda)\, v_j, \lambda \in R$

viii. Straight segment $\overline{v_i v_j}$
A straight segment is a portion of a straight line, that starts at v_i and ends at v_j.
$\overline{v_i v_j} = (\lambda)\, v_i + (1 - \lambda)\, v_j, 0 \leq \lambda \leq 1$

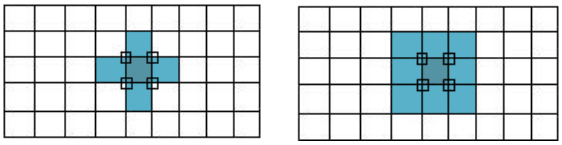
ix. Distance vertex-line $d\left(v, \overleftrightarrow{v_i v_j}\right)$
A distance vertex-line is the perpendicular euclidean distance D_E from the vertex v to the line $\overleftrightarrow{v_i v_j}$.

x. Neighborhood
The relationship between pairs of pixels (i, j) and (m, n) that share one or two end-points.

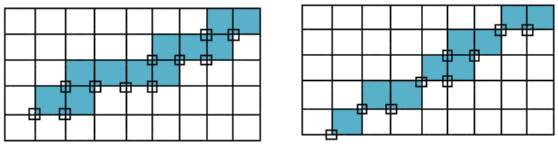
(a) 4-Neighborhood of a pixel (i, j), $N_4(i, j)$ (See figure 2.2(a)).
$N_4(i, j) = \{(m, n) | (|m - i| = 1) \otimes (|n - j| = 1)\}$

(b) 8-Neighborhood of a pixel (i, j), $N_8(i, j)$ (See figure 2.2(b))
$N_8(i, j) = \{(m, n) | (|m - i| = 1) \vee (|n - j| = 1)\}$

Figure 2.1: Examples of Neighborhood and Connectivity



(a) 4-neighborhood (b) 8-neighborhood



(c) 4-connectivity (d) 8-connectivity

xi. Distances between vertices
The amount of separation between two pixels (i, j) and (h, k)([13], p. 27).
Given two pixels $(i, j)\, and\, (h, k) \in Z \times Z$

(a) Euclidean distance:
$D_E\left[(i, j), (h, k)\right] = \sqrt{(i-h)^2 + (j-k)^2}$

(b) 4-Neighborhood distance:
$D_4\left[(i, j), (h, k)\right] = |i-h| + |j-k|$

(c) 8-Neighborhood distance:
$D_8\left[(i, j), (h, k)\right] = max\left\{|i-h|, |j-k|\right\}$

xii. Path $p\left(v_0, v_f\right)$
A path is a non-selfintersecting, unit-width sequence of vertices, starting at v_0 and ending at v_f, made up of orthogonal or diagonal steps. No assumptions are made on $I(v_i)$
$p\left(v_0, v_f\right) = [v_0, v_1, ..., v_f]$ s.t. $\left(v_i \in [0, X_c] \times [0, Y_r]\right) \wedge \left(D_8\left(v_i, v_{i+1}\right) \leq 1\right) \wedge$
$\left(\left|N_8\left(v_i\right)\right| \leq 2\right)$

xiii. Connectivity
The relationship of neighborhood between pairs of pixels $v_{i,j}$ and $v_{m,n}$ determines the connectivity between them.

(a) 4-Connectedness relation between pixels $C_4(v_i, v_j)$.
Vertices v_i and v_j are 4-connected, $C_4(v_i, v_j)$ iff $v_i \in N_4(v_j)$ (informally, v_i is

an orthogonal neighbor of v_j).
Notice that $C_4(v_i, v_j) \Rightarrow C_4(v_j, v_i)$. However, $C_4()$ *is not* a transitive relation, and therefore it *is not* an equivalence relation. Figure 2.2(c) shows a case of 4-connectivity where the small squares represent the connectedness.

(b) 8-Connectedness relation between pixels $C_8(v_i, v_j)$.
Vertices v_i and v_j are 8-connected, $C_8(v_i, v_j)$ iff $v_i \in N_8(v_j)$ (informally, v_i is an orthogonal or diagonal neighbor of v_j).
Notice that $C_8(v_i, v_j) \Rightarrow C_8(v_j, v_i)$. Again, $C_8()$ *is not* a transitive relation, and therefore it *is not* an equivalence relation. Figure 2.2(d) shows a case of 8-connectivity where the small squares represent the connectedness.

2.2.2 Edge detection

In this section the terms used in the edge generation process are defined.

i. Orthogonal differences, $\Delta_i I, \Delta_j I$
The orthogonal differences $\Delta_i I$ and $\Delta_j I$ approximate the first directional derivatives of $I()$ in the X and Y directions respectively ([13], p. 80). They are applied on functions, and return a scalar value:
$\Delta_i() : I() \rightarrow R$
$\Delta_j() : I() \rightarrow R$
calculated as:
$\Delta_i I\,(i,j) = I\,(i+1,j) - I\,(i-1,j)$
$\Delta_j I\,(i,j) = I\,(i,j+1) - I\,(i,j-1)$
Notation simplification:
if $v = (i,j) \Rightarrow \Delta_i I\,(i,j) = \Delta_i I(v)$

ii. Gradient operator $\nabla I(i,j)$
A Gradient operator is the vector formed by the directional derivatives or Orthogonal Differences ($\Delta_i I, \Delta_j I$) described above. The Gradient operator is applied on functions, and returns a vector:
$\nabla() : I() \rightarrow R^2$
$\nabla() : I() = \nabla I(i,j) = (\Delta_i I(i,j), \Delta_j I(i,j))$
Collaterally, its direction and magnitude are defined:

(a) Gradient direction, $\theta(I(i,j))$, is the angle (in radians) from the x axis to the point (i,j):
$\theta(I(i,j)) = arg(\nabla I(i,j)) = \arctan\left(\frac{\Delta_j I(i,j)}{\Delta_i I(i,j)}\right)$

(b) Gradient magnitude, $|\nabla(i,j)|$, is the norm of the Gradient operator:
$|\nabla I(i,j)| = |(\Delta_i I(i,j), \Delta_j I(i,j))| = \sqrt{(\Delta_i I\,(i,j))^2 + (\Delta_j I\,(i,j))^2}$

iii. Discretized Gradient $\lfloor\nabla\rfloor I(v_k)$
This is an approximation of the continous gradient $\nabla I(v_k)$ taken from a set with cardinality 8, formed by the vectors: $\left\{\pm\widehat{i}, \pm\widehat{j}, \pm\widehat{i} \pm \widehat{j}\right\}$

$\lfloor\nabla\rfloor I(v_k) = \lfloor\theta(v_k)\rfloor$

The $\lfloor \quad \rfloor$ operator maps real values of the gradient angle to elements of a discrete set, according to the interval which contains the actual gradient angle. The mapping is given in Table 2.1.

Tabla 2.1: Mapping of gradient angle to discrete values of angles

Interval containing $\nabla I(v_k)$	Discretized Gradient $\lfloor\nabla\rfloor I(v_k)$
$(337.5°, 22.5°]$	$0°$
$(22.5°, 67.5°]$	$45°$
$(67.5°, 112.5°]$	$90°$
$(112.5°, 157.5°]$	$135°$
$(157.5°, 202.5°]$	$180°$
$(202.5°, 247.5°]$	$225°$
$(247.5°, 292.5°]$	$270°$
$(292.5°, 337.5°]$	$315°$

iv. Thresholds $T1, T2$
A pixel location is declared an edge location if the value of its $\nabla(\ )$ exceeds a threshold. Two thresholds $T1, T2$, where $T1 > T2$ are needed for the hystersis step of the edge detection process.
Range of thresholds $T1, T2$: $[0, max\,(\nabla\,(\ ))]$

v. Generalized edge $E\,(v_0, v_f)$
A generalized edge between v_0 and v_f , $E\,(v_0, v_f)$, is a path $p\,(v_0, v_f)$ between them, built with high gradient pixels. All pixels of the path have gradient larger than T_2, and at least one pixel in the path has gradient larger than T_1. A generalized edge $E()$ does not have to be straight.
Given $T_1, T_2 \in N$, where $T_1 > T_2$,

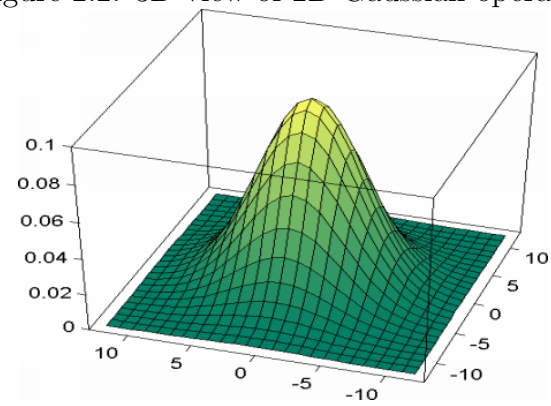
$$E\,(v_0, v_f) = [v_0, v_1, ..., v_f] s.t. (p\,(v_0, v_f)) \wedge \\ (\forall v_i \in p(v_0, v_f), \nabla I(v)_{v=v_i} > T_2) \wedge (\exists v_j \in p(v_0, v_f), \nabla I(v)_{v=v_j} > T_1)$$

vi. Straight Edge (or simply, *edge*) $e\,(v_0, v_f)$
An *edge* is an approximately straight generalized edge, in which the perpendicular distance from each pixel to the straight segment, $\overline{v_0 v_f}$, joining the extremes is

bounded by an ϵ value.
$e\left(v_0, v_f\right) = [v_0, v_1, ..., v_f]$ s.t. $E(v_0, v_f) \wedge \forall v_i \in E(v_0, v_f), d\left(v_i, \overline{v_0 v_f}\right) < \epsilon$
Typical values for ϵ: $[0, 10]$

vii. Standard deviation σ
A Standard deviation is a parameter that indicates the way in which a probability function or a probability density function is centered around its mean μ([10]). In this work σ is an input parameter of the Gaussian operator, which is explained below (see viii).

Figure 2.2: 3D view of 2D Gaussian operator



viii. 2D Gaussian operator $G_{\Delta x, \Delta y, \sigma}\left(i, j\right)$
Gaussian operator is a function that reduces the possible number of frequencies at which image intensity function changes take place. The parameter σ is the standard deviation of a normal distribution with $\mu = 0$. Pixels farther than 3σ from the μ have negligible influence, hence, the operator is truncated at 3σ ([13], p. 84).
$G_{\Delta x, \Delta y, \sigma}\left(i, j\right) : Z^2 \rightarrow R$

$$G_{\Delta x, \Delta y, \sigma}\left(i, j\right) = \begin{cases} \frac{1}{2\pi\sigma^2} e^{-\left(\frac{i^2+j^2}{2\sigma^2}\right)}, & \text{if } -\Delta x \leq i \leq \Delta x, \\ & \quad -\Delta y \leq j \leq \Delta y \\ 0, & \text{otherwise} \end{cases}$$

Usually $\Delta x = \Delta y = 3\sigma$ Typical values for σ are in $[0, 10]$.
Figure 2.2 shows 2D $G_{\Delta x=12, \Delta y=12, \sigma=4}\left(\right)$ with $\mu = 0$.

ix. 1D Gaussian operator $G_{\Delta x, \sigma}\left(i\right)$

$G_{\Delta x,\sigma}(i) : Z \to R$

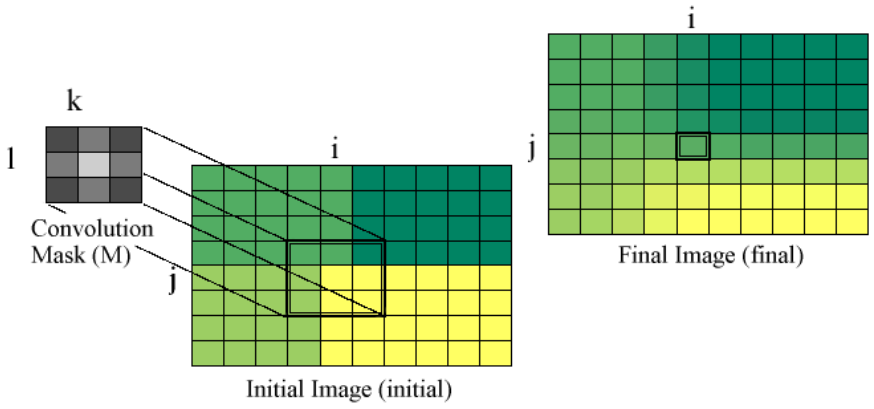
$$G_{\Delta x,\sigma}(i) = \begin{cases} \frac{1}{2\pi\sigma^2} e^{-\left(\frac{i^2}{2\sigma^2}\right)}, & \text{if } -\Delta x \leq i \leq \Delta x \\ 0, & \text{otherwise} \end{cases}$$

Usually $\Delta x = 3\sigma$
Typical values for σ are in $[0, 10]$
([13], p. 84)

x. Convolution
Convolution is a linear operation that calculates a resulting value as a linear combination of the neighborhood of the input pixel and a convolution mask ([13] pp. 68-69).

(a) Convolution with 2D filter.
In Figure 2.3 the output pixel -*final(i,j)*- is calculated as a linear combination of the neighborhood of the input pixel -*inital(i,j)*- and the coefficients of the 2D convolution mask *M*. Vertical and horizontal features are affected by a 2D filter.

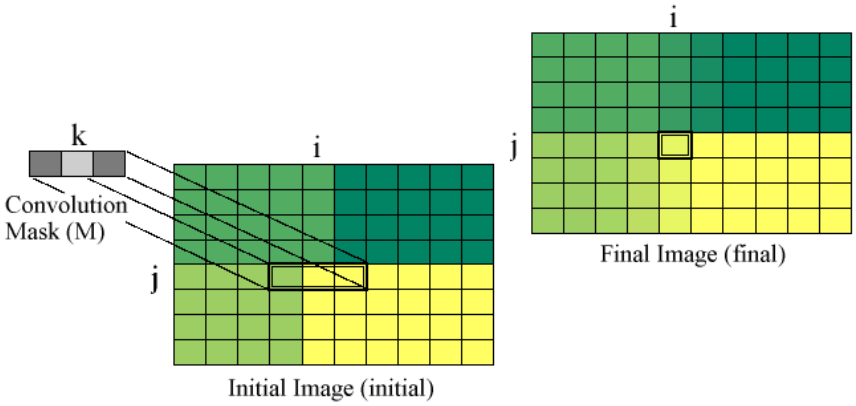
Figure 2.3: Value of pixel Final Image(i, j) is the result of convolving the neighborhood of pixel Initial Image (i, j) and the 2D Convolution Mask *M*.



$$\begin{aligned} Final(i,j) &= Initial(i,j) * M \\ &= \sum_{k=-1}^{+1} \sum_{l=-1}^{+1} Initial(i+k, j+l) M(k,l) \end{aligned}$$

(b) Convolution with 1D filter.
In Figure 2.4 the output pixel -*final(i,j)*- is calculated as a linear combination of the neighborhood of the input pixel -*inital(i,j)*- and the coefficients of the 1D convolution mask *M*. Different from case a), results of this convolution only affect vertical features of Final Image, because 1D filter was applied horizontally to Initial Image.

Figure 2.4: Value of pixel Final Image(i, j) is the result of convolving the neighborhood of pixel Initial Image (i, j) and the 1D Convolution Mask *M*.



$$\begin{aligned} Final(i,j) &= Initial(i,j) * M \\ &= \sum_{k=-1}^{+1} Initial(i+k,j)M(k) \end{aligned}$$

2.2.3 Corner Extraction

In this section terms used in the corner generation process are defined.

i. Region $R_e\left(T_l, T_h\right)$
A *Region* is a connected set of pixels with homogeneous image intensity. Values T_l and T_h are the low and high thresholds that determine the intensity of the region.

$$\begin{aligned} R_e\left(T_l, T_h\right) = \{v \in Z \times Z | \left(T_l \leq I\left(v\right) \leq T_h\right) \wedge \\ \left(\forall w \in R_e\left(T_l, T_h\right) \exists\, path\left(\right) = [v, ...w] \subset R_e\left(T_l, T_h\right)\right)\} \end{aligned}$$

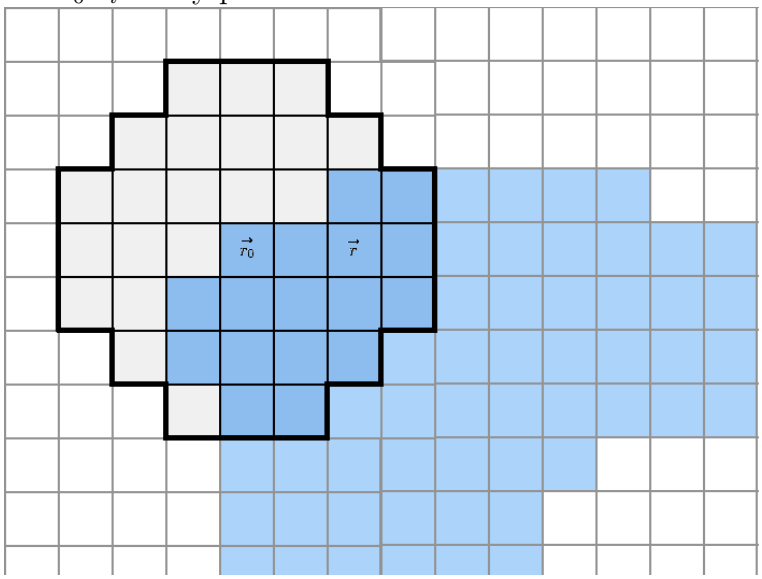
ii. Boundary of a Region $\delta R_e\left(T_l, T_h\right)$
The Boundary of a Region is a closed non self intersecting parametric (in the

parameter u) curve $\in R^2$.
$\delta R_e\left(T_l, T_h\right) = \Gamma\left(u\right) = \left(\Gamma_x\left(u\right), \Gamma_y\left(u\right)\right)$

iii. Corner *corner*
A corner is the intersection point of two or more edges that border different regions.
$corner = (x, y) s.t. (p_i(e_1, e_2) \wedge (e_1 \wedge e_2 \in \delta R_e\left(T_l, T_h\right)))$

iv. Circular Mask $M\left(x, y, R\right)$
Set of pixels (usually 37) that make up the interior and boundary of a circular subwindow of an *I()*([12]).
$M\left(x, y, R\right) = \left\{(i, j) \,|\, (x-i)^2 + (y-j)^2 \leq R^2\right\}$
With nucleus v_0 the center of the circle: $v_0 = (x, y)$
And v_i any other point in the circle: $v_i = (i, j)$.
A typical *Circular mask* is shown in Figure 2.5.

Figure 2.5: *Circular mask* $M\left(x, y, R\right)$ bounded by black contour, at the centre of the circle the *nucleus*: v_0.v_i is any pixel in the mask other than the *nucleus*.



v. Intensity difference threshold ΔI_t
A threshold of the intensity function of an image. It is used in order to define similarity in intensity among the nucleus v_0 and all other pixels of Circular mask *M( )*([12]).
Typical values for ΔI_t are in $[10, 60]$.

vi. Delta intensity function $\delta_{v_0, \Delta I_t}\left(v_i\right)$

Difference in intensity between pixels v_i and v_0 ([12]).

$$\delta_{v_0,\Delta I_t}(v_i) = \begin{cases} 1, & \text{if } |I(v_i) - I(v_0)| \leq \Delta I_t \\ 0, & \text{otherwise} \end{cases}$$

Given an intensity difference threshold ΔI_t, $\delta_{v_0,\Delta I_t}(v_i)$ measures the similarity in pixel intensity with respect to a reference pixel v_0.

vii. Univalue Segment Assimilating Nucleus (U.S.A.N.) area $USAN(v_0, R, \Delta I_t)$
It is the neighborhood of a reference pixel v_0, whose intensity is similar to the intensity of v_0 ([12]). Figure 2.5 shows the $USAN()$ area within the circular mask as a region of the same color as the nucleus (v_0).

$USAN(v_0, R, \Delta I_t) = \{v_i \in M(v_0, R) \mid (|I(v_i) - I(v_0)|) \leq \Delta I_t\}$
$|USAN()|$ is the cardinality of $USAN()$.

viii. Geometric threshold g_t
Geometric threshold is the maximum size allowed for $|USAN()|$. The maximum value of this threshold is bounded by the size of the Circular mask *M(). Corner* sharpness is determined by this threshold ([12]).
Domain
$[0, |M(\,)|]$
Typical values for g_t: $[15, 28]$.

ix. Corner response $R(v_0)$
Corner response is a number, that determines the capacity of the nucleus (v_0) to be detected as a corner. The higher the value of $R(v_0)$ the greater the possibility for v_0 to be a *corner* ([12]).

$$R(v_0) = \begin{cases} g_t - |USAN()| & \text{if } |USAN()| < g_t \\ 0 & \text{otherwise} \end{cases}$$

x. Center of mass $CM(P)$
Centroid or point whose coordinates are the averages of the corresponding coordinates of a given set of pixels P([6]).
$\overline{x} = \frac{1}{N} \sum \sum_{(i,j)\in P} x(i,j)$,
$\overline{y} = \frac{1}{N} \sum \sum_{(i,j)\in P} y(i,j)$
N : Total number of pixels in P

xi. Colinearity *colinearity(corner(), e())*
It is a number that determines if a corner and an edge lie on the same infinite line. Given, a $corner(x, y)$ and an $e((x_1, y_1), (x_2, y_2))$,
and the parametric equation of the line ([5]):

$x = x_1 + t_x (x_2 - x_1)$
$y = y_1 + t_y (y_2 - y_1)$
where,
$t_x = \frac{(x - x_1)}{(x_2 - x_1)}$
$t_y = \frac{(y - y_1)}{(y_2 - y_1)}$
colinearity(corner(), e()) $= t_x$ iff $(t_x = t_y) \vee (|t_x - t_y| < \epsilon)$.

xii. Intersection point $p_i (e_1, e_2)$
The intersection point is the pixel where two edges ($e()$) meet. Given two edges: $e_1 ((x_1, y_1), (x_2, y_2))$ and $e_2 ((x_3, y_3), (x_4, y_4))$ and the parameter t:
$t = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$
The intersection point is defined as:
$x = x_1 + t(x_2 - x_1)$,
$y = y_1 + t(y_2 - y_1)$

xiii. Tolerance
The maximum allowable value deviation from a standard or the range of variation permitted in maintaining a specified dimension.

(a) Tolerance edge-corner ec_t
Maximum Euclidean distance (D_E) allowed from a corner to an edge vertex.
Typical values for ec_t: $[0 - 10]$.

(b) Tolerance segments s_t
Maximum Euclidean distance (D_E) allowed from any pixel of the Generalized Edge ($E()$) to a rough approximation line, when transforming edges from raster representation ($E()$) into segment representation ($e()$).
Typical values for s_t: $[0 - 10]$.

Chapter 3

Literature survey

Position and orientation of a person are required for tracking. In order to track the position, systems based on GPS, smartcards ([8]) , ultrasonic sensors ([9]) have been developed. Orientation is tracked by means of accelerometers, inclinometers, gyroscopes,(which sense the Earth's magnetic field), etc.

Tracking for AR requires high precision, because virtual objects must superimpose real world objects in the line of sight of the user. Hence, little misalignments are easily noticeable.

Tracking based on sensors, such as the ones mentioned above, is useful in certain applications where accuracy in alignment is not the main goal. For example in cases where only annotations are superimposed on real world objects ([4]). But in cases where superposition has to be very precise, this approach is insufficient. Therefore, tracking prototypes based not only on sensors but also on images captured by a camera are being developed in order to obtain very accurate pose estimates.

Even though images seem a promising solution for tracking. Research toward systems that are able to track natural markers/landmarks is still an open problem. Optical tracking has been implemented in several prototypes. Jiang and Neumann ([7]) developed a prototype that is based on pre-calibrated landmarks. Uncalibrated straight lines of man-made environment are used in order to extend the tracking to an unprepared environment. Satoh et. al. ([11]) choose template landmarks from the images captured by the camera. Pose tracking is based on template matching and sensors.

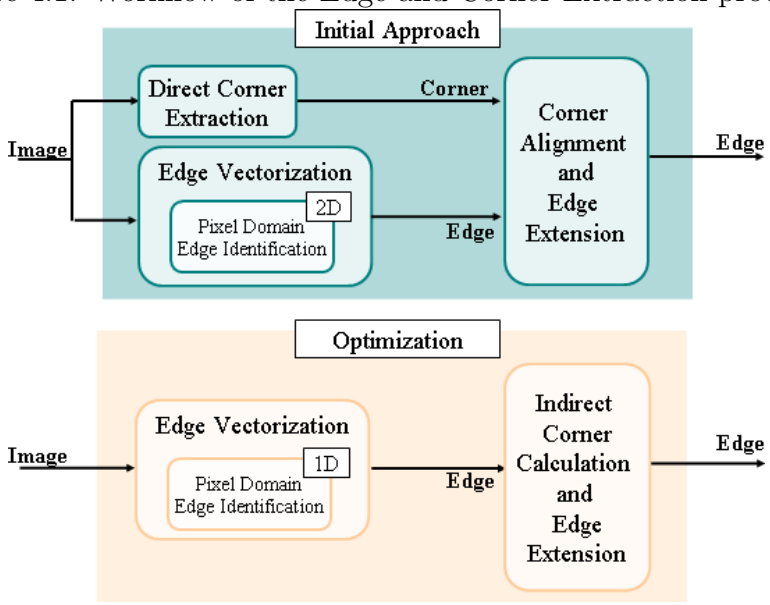
The *Geist* prototype does not use precalibrated landmarks, nor templates. The approach is based on a Matching Algorithm that matches pairs of images: one coming from the video-camera and the other one, coming from a reference 3D model database. The Matching Algorithm gives a corrected estimate of the user's pose. This approach is good because it works well at big translations of the camera. And since it has a

reference model database of the surroundings, it can be used outdoors and the user does not have to be at a fixed position. The approach aims at true mobile computing.

Chapter 4

Methodology

Figure 4.1: Workflow of the Edge and Corner Extraction processes



The goal of this work is to identify edges and corners of a given image $I(\ )$. An Initial Approach to the problem (upper part of Figure 4.1) follows three steps:

i. Edge vectorization. It is the process that identifies the generalized edges of an image based on 2D filters. Further transformation of generalized edges into vector representation (or edges) is carried out in this process.

ii. Corner Extraction. It is the process that identifies pixels corresponding to corners in an image.

iii. Corner Alignmet. This process searches for colinear corners of an edge.

An alternative approach called -the Optimization- (lower part of Figure 4.1) generates the same results as the Initial Approach. Besides, it improves the quality of results and the algorithm performance. The two methods -Initial and Optimization- are alternative ways to reach the same goal: edge and corner generation. This alternative approach consists of the following steps:

i. Edge Vectorization. Based on 1D filters the process of Edge Identification and edge vectorization is carried out.

ii. Indirect Corner calculation. It consists on the calculation of corners based on the edges obtained from Edge Vectorization step.

This chapter describes the processes of Edge Vectorization and Corner Extraction for both -the Initial and the Optimization- approaches. To end with, the steps taken for the Experimental Setup are described.

4.1 Initial Approach

4.1.1 Edge Vectorization

It is a process composed of two steps: identification of edges in a given image and transformation of the representation of the edges from raster image into vector form. This two steps are called:

i. Identification of Edges in Pixel Domain.

ii. Vectorized Edge Synthesis.

Identification of Edges in Pixel Domain

It is the process of locating pixels corresponding to edges in a given image. This process is based in the Canny edge detector ([2]). It is composed of four steps as depicted in Figure 4.2. Algorithm 1 shows the steps required to Identification of Edges in Pixel Domain.

i. Gaussian Convolution (line 9, function 1)

ii. First Derivative Convolution (line 11, function 1)

iii. Non-maxima Suppression (line 12, function 1)

iv. Hysteresis thresholding (line 13, function 1)

Function 1 Pixel Domain Edge Identification Algorithm

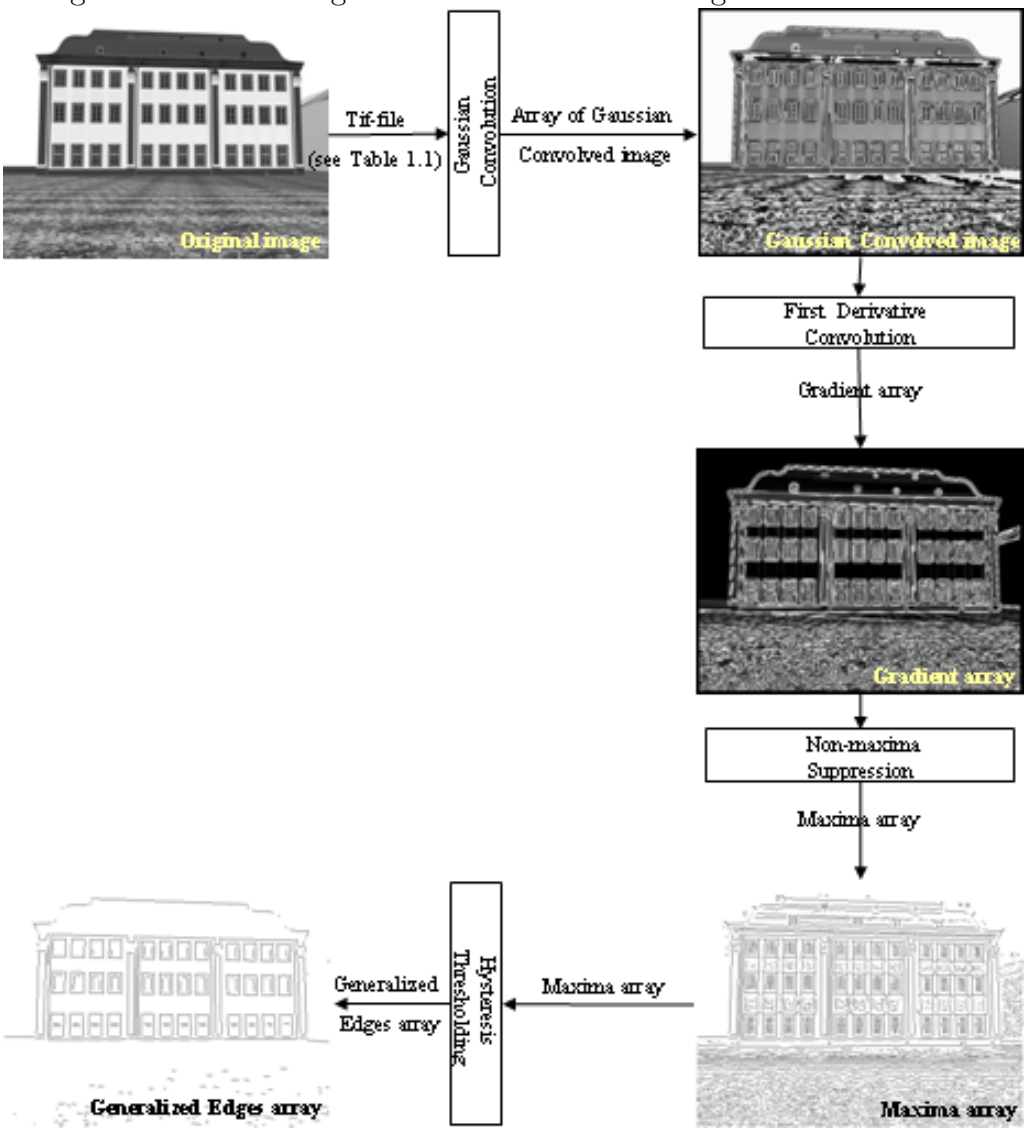
$[Maxima()]$ = **Edge_Identification**($I()$, σ, $T2$, $T1$)

Input: $I()$: Image
σ: Standard deviation
$T2$: Lower threshold
$T1$: Higher threshold

Output: $Maxima()$: Array of Generalized Edges

1: $Y_r = 480$ (Total number of rows)
2: $X_c = 640$ (Total number of columns)
3: $height = Y_r - 2(3\sigma)$
4: $width = X_c - 2(3\sigma)$
5: $Gra\_Magnitude(width, height) = 0$
6: $Gra\_Direction(width, height) = 0$
7: $Maxima(width, height) = 0$
8: **if** $(\sigma \neq 0)$ **then**
9: $[I\,()]$ = **Gaussian_Convolution**$(G_{\Delta x,\Delta y,\sigma}\,(), I\,(), \sigma, X_c, Y_r)$ (Function 2)
10: **end if**
11: $[Gra\_Magnitude\,(\,), Gra\_Direction\,(\,)]$ = **First_Derivative_Convolution**$(I\,(), width, height)$ (Function 3)
12: $[Maxima()]$ = **Non_Maxima_Suppression**$(Gra\_Magnitude\,(\,), Gra\_Direction\,(\,), width, height)$ (Function 4)
13: **Hysteresis_Thresholding**$(Maxima\,(\,), T2, T1, width, height)$ (Function 6)
14: **return** $Maxima()$

Figure 4.2: Block diagram of Identification of Edges in Pixel Domain

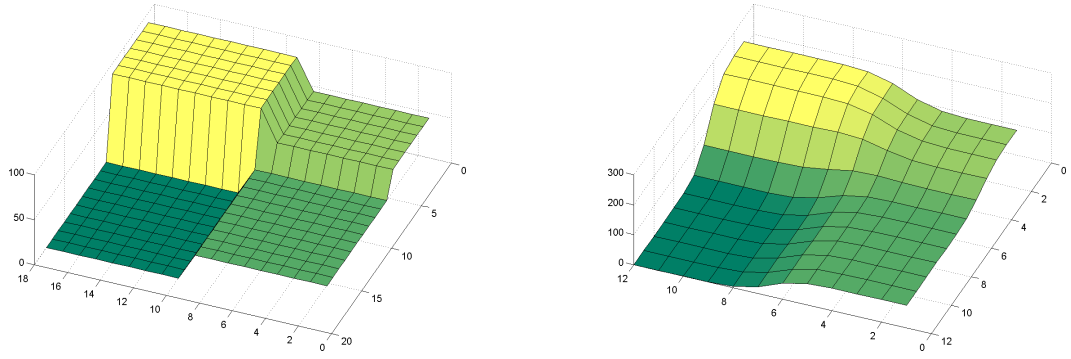


i. **Gaussian Convolution.**

Gaussian convolution is an operation that reduces high frequencies of the intensity function of an image *I()* by the use of a filter called the Gaussian operator. It is applied in order to reduce noise and fine detail from the image. Parameter σ of the gaussian operator $G_{\Delta x,\Delta y,\sigma}()$ determines the degree of smoothing to be applied to the image.

Figure 4.3-a shows an image seen as a scalar field. The image is made up of four intensity values (i.e. four colors). Each color represents a change in the intensity function. Figure 4.3-b shows the image after the gaussian convolution, notice

Figure 4.3: Gaussian Convolution



(a) Original Image in 3D

(b) Original Image after Gaussian Convolution in 3D

that large differences in the intensity function are smoothed by the convolution. Also notice that neighborhood of pixels of constant intensity values remain unchanged.
In lines 8 - 17, Function 2, the gaussian operator $G_{\Delta x,\Delta y,\sigma}()$ is applied to neighborhood of pixel *I(i,j)*, and the resulting value is stored in array *Smoothed_Image()*.
The input of the gaussian convolution is a TIFF file (Table 1.1) and the output is an array of a gaussian convolved image as shown in Figure 4.2.
Notice that If $\sigma = 0$ for $G_{\Delta x,\Delta y,\sigma}()$ then no gaussian convolution is applied to the image and the image goes directly to second step: First Derivative Convolution (see Function 1, line 8).
Also note that after applying Gaussian Convolution the smoothed image has a smaller size.
Size of Smoothed_Image = [Original number of columns $-2(3\sigma)$]$\times$ [Original number of rows $-2(3\sigma)$]

ii. **First Derivative Convolution.**

This process highlights regions whose first spatial derivatives are high (i.e. noise, big changes in intensity function of the image).
The procedure consists of two steps:

(a) Orthogonal differences calculation. The othogonal differences $\Delta_i()$ and $\Delta_j()$ of the current pixel are calculated and results are written to their respective $xComponent\,(col, row)$ and $yComponent\,(col, row)$ arrays as shown in lines 8 and 12, Function 3.

(b) Gradient magnitude and direction calculation. Based on the components

Function 2 Gaussian Convolution

$[Smoothed\_Image()]$ = **Gaussian_Convolution**($G_{\Delta x,\Delta y,\sigma}()$, $I()$, σ, X_c, Y_r)

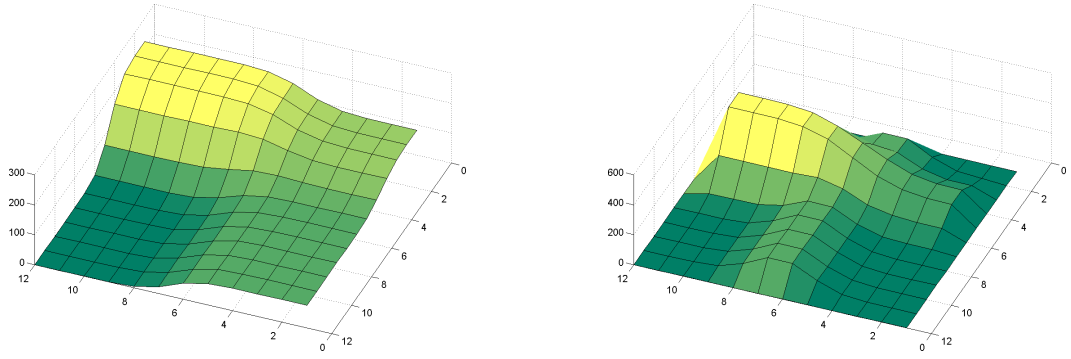
Input: $G_{\Delta x,\Delta y,\sigma}()$: Gaussian operator
$I()$: Image
σ: Standard deviation
X_c: Total number of columns
Y_r: Total number of rows

Output: $Smoothed\_Image()$: Image softened by the Gaussian convolution

Precondition: $\sigma \neq 0$

1: {$I()$ is scanned pixel by pixel in the two following loops}
2: **for** $row = 3\sigma$ to $Y_r - 3\sigma$ **do**
3: $y = 0$
4: **for** $col = 3\sigma$ to $X_c - 3\sigma$ **do**
5: $x = 0$
6: $l = 0$
7: {Convolution is done in the following two loops: $G_{\Delta x,\Delta y,\sigma}(k,l)$ is convolved with $I(i,j)$}
8: **for** $j = row - 3\sigma$ to $j \leq row + 3\sigma$ **do**
9: $k = 0$
10: **for** $i = col - 3\sigma$ to $i \leq col + 3\sigma$ **do**
11: $Smoothed\_Image(x+col, y+row) = Smoothed\_Image(x+col, y+row) + I(i,j)\, G_{\Delta x,\Delta y,\sigma}(k,l)$
12: $x = x + 1$
13: $y = y + 1$
14: $k = k + 1$
15: **end for**
16: $l = l + 1$
17: **end for**
18: **end for**
19: **end for**
20: **return** $Smoothed\_Image(\,)$

Figure 4.4: First Derivative Convolution



(a) Gaussian Convolved Image in 3D

(b) First Derivatives Convolution applied to a Gaussian Convolved Image in 3D

found in previous step, gradient magnitude ($\nabla()$) and discretized gradient direction ($\lfloor\theta()\rfloor$) are calculated as results of this procedure (lines 15 and 17 of Function 3).

The output of this procedure is:

(a) Gradient magnitude array. It is an array that holds the rate of increase or decrease of the intensity function per pixel.

(b) Discretized gradient direction array. It is an array that holds gradient direction per pixel. Edge direction corresponds to gradient direction - 90°.

iii. **Non-maxima Suppression.**

Let R_T be a *region* of pixels with a large image gradient ($\nabla I > T$). The goal of the Non-maxima Suppression Algorithm is to make R_T a thin region (a *path*), by suppressing from it all pixels whose gradient $\nabla I()$ is not a local maximum.
Gradient $\nabla I()$ is used to determine a directional derivative calculated along a line. The output of this step is a gradient maxima array.
The procedure to suppress non-maxima follows four steps:

(a) Two-neighbor finding. Two pixels in array *Gra_Magnitude()* in the 8-neighborhood of the current pixel (col,row) are found by calling the function $select-two-neighbors-along-theta()$(Line 4, Function 4). The found pixels must be along gradient direction of current pixel. Neighbor1 and neighbor2 are assigned with values of the gradient along gradient direction.

Function 3 First Derivative Convolution

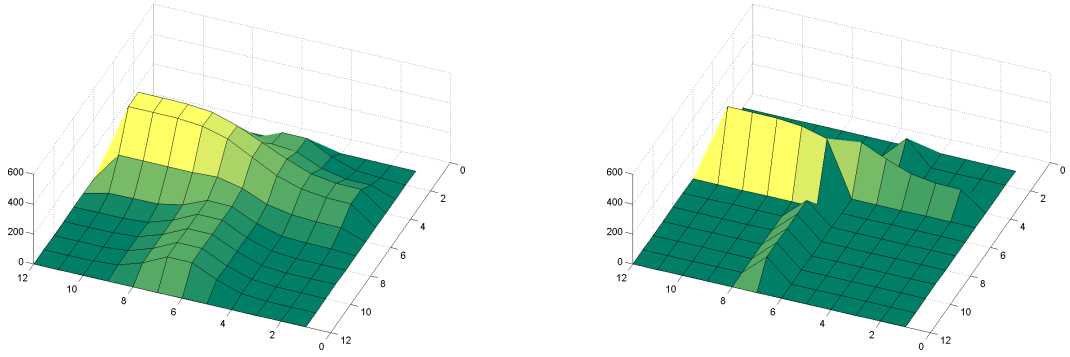
$[Gra\_Magnitude\,(\,),Gra\_Direction\,(\,)]$ = **First_Derivative_Convolution**($I()$, $width$, $height$)

Input: $I()$: Image
$width$: Total number of columns of image
$height$: Total number of rows of image

Output: $Gra\_Magnitude(\,)$ Array of gradient magnitude
$Gra\_Direction(\,)$ Array of gradient direction

1: $xComponent(width, height) = 0$
2: $yComponent(width, height) = 0$
3: {Scan $I\,()$ pixel by pixel:}
4: **for** $row = 1$ to $height - 1$ **do**
5: **for** $col = 1$ to $width - 1$ **do**
6: {Find horizontal component of directional derivative:}
7: **for** $j = row - 1$ to $j \leq row + 1$ **do**
8: $xComponent(col, row) = xComponent(col, row) + \Delta_i I\,(col, j)$
9: **end for**
10: {Find vertical component of directional derivative:}
11: **for** $i = col - 1$ to $i \leq col + 1$ **do**
12: $yComponent(col, row) = yComponent(col, row) + \Delta_j I\,(i, row)$
13: **end for**
14: {Find gradient magnitude $\nabla\,()$:}
15: $Gra\_Magnitude(col, row) = |\nabla\,(yComponent(col, row), xComponent(col, row))\,|$
16: {Find discretized gradient direction $\lfloor\theta\,()\rfloor$:}
17: $Gra\_Direction(col, row) = \lfloor\theta\,(yComponent(col, row), xComponent(col, row))\rfloor$
18: **end for**
19: **end for**
20: **return** [$Gra\_Magnitude(\,)$, $Gra\_Direction(\,)$]

Figure 4.5: Non-maxima Suppression



(a) First Derivatives Convolution applied to a Gaussian Convolved Image in 3D

(b) Non-maxima Suppression in 3D

(b) First suppression. Neighbor1 and neighbor2 should be **less** than the gradient of the current pixel (line 5, Function 4). Hence, the current pixel is taken into consideration as a maximum and is written to array $Maxima(col, row)$ (line 6, Function 4). Otherwise, the current pixel is not considered a maximum. Suppression consists on writing a zero in array $Maxima(col, row)$.

(c) Two-neighbor finding. Step a) is repeated here but instead of *Gra_Magnitude()*, *Maxima()* array is used. Two pixels in array *Maxima()* in the 8-neighborhood of current pixel (col, row) are found by calling the function $select - two - neighbors - along - theta()$(Line 12, Function 4). The found pixels must be along gradient direction of the current pixel. Neighbor1 and neighbor2 are assigned with values of the gradient along gradient direction.

(d) Second suppression. Neighbor1 and neighbor2 should be **less or equal** than the gradient of current pixel(line 13, Function 4). Hence, current pixel is considered a maximum. Otherwise current pixel is not considered a maximum and is suppressed by writing a zero in $Maxima\,(col, row)$ as shown in line 14, Function 4.

Results of this procedure are maxima values of the gradient stored in array $Maxima(\,)$. These values correspond to candidate edges. Zero values in $Maxima(\,)$ correspond to non-edge pixels.

iv. **Hysteresis Thresholding.**

Hysteresis thresholding is a process to avoid edge streaking. Two thresholds $T1$ and $T2$, with $T1 > T2$ are used for this process. Gradient values ($\nabla()$) above $T1$

Function 4 Non-maxima Suppression

$[Maxima()]$ = **Non_Maxima_Suppression**($Gra\_Magnitude(\ )$, $Gra\_Direction()$ $width$, $height$)

Input: $Gra\_Magnitude(\ )$: Array of gradient magnitude
$Gra\_Direction()$: Array of gradient direction
$width$: Total number of columns
$height$: Total number of rows

Output: $Maxima()$: Array of maxima vaues of gradient magnitude

1: $neighbor1, neighbor2$
2: **for** $row = 1$ to $height - 1$ **do**
3: **for** $col = 1$ to $width - 1$ **do**
4: $[neighbor1, neighbor2]$ =**select-two-neighbors-along-theta** $(row, col, Gra\_Magnitude(\ ), Gra\_Direction(\ ))$ (Function 5)
5: **if** $((Gra\_Magnitude(col, row) \geq neighbor1)$ **and** $(Gra\_Magnitude(col, row) \geq neighbor2))$ **then**
6: $Maxima(col, row) = Gra\_Magnitude(col, row)$
7: **end if**
8: **end for**
9: **end for**
10: **for** $row = 1$ to $height - 1$ **do**
11: **for** $col = 1$ to $width - 1$ **do**
12: $[neighbor1, neighbor2]$ =**select-two-neighbors-along-theta** $(row, col, Maxima(\ ), Gra\_Direction(\ ))$ (Function 5)
13: **if** $!((Maxima(col, row) > neighbor1)$ **and** $(Maxima(col, row) > neighbor2))$ **then**
14: $Maxima(col, row) = 0$
15: **end if**
16: **end for**
17: **end for**
18: **return** $[Maxima(\ )]$

Function 5 Select Two Neighbors Along Gradient Direction $\theta()$

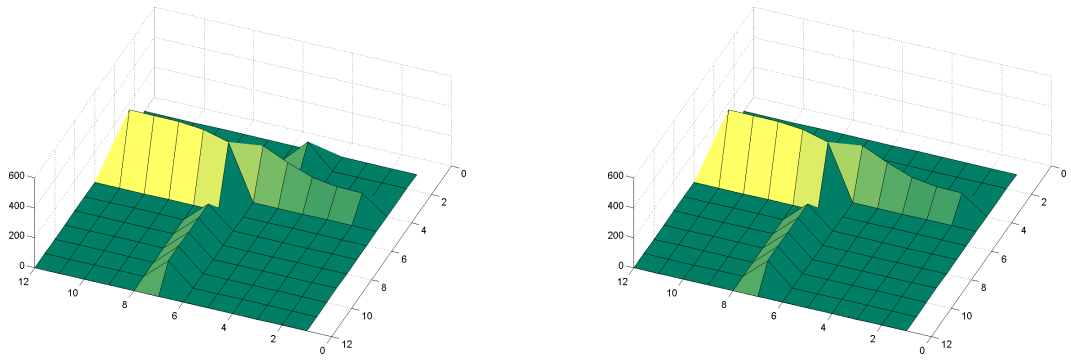
$[neighbor1, neighbor2]$ = **select-two-neighbors-along-theta**(row, col, $Values(\ )$, $Gra\_Direction(\ )$)

Input: row: Current row
col: Current column
$Values(\ )$: Array of gradient magnitude
$Gra\_Direction(\ )$: Array of gradient direction

Input-Output: $neighbor1$: Pixel along gradient direction of current pixel (col, row)
$neighbor2$: Pixel along gradient direction of current pixel (col, row)

1: **if** $(Gra\_Direction(col, row) == 0°)$ **then**
2: $neighbor1 = Values(col - 1, row)$
3: $neighbor2 = Values(col + 1, row)$
4: **else if** $(Gra\_Direction(col, row) == 45°)$ **then**
5: $neighbor1 = Values(col + 1, row - 1)$
6: $neighbor2 = Values(col - 1, row + 1)$
7: **else if** $(Gra\_Direction(col, row) == 90°)$ **then**
8: $neighbor1 = Values(col, row - 1)$
9: $neighbor2 = Values(col, row + 1)$
10: **else if** $(Gra\_Direction(col, row) == 135°)$ **then**
11: $neighbor1 = Values(col - 1, row - 1)$
12: $neighbor2 = Values(col + 1, row + 1)$
13: **else if** $(Gra\_Direction(col, row) == 180°)$ **then**
14: $neighbor1 = Values(col - 1, row)$
15: $neighbor2 = Values(col + 1, row)$
16: **else if** $(Gra\_Direction(col, row) == 225°)$ **then**
17: $neighbor1 = Values(col + 1, row - 1)$
18: $neighbor2 = Values(col - 1, row + 1)$
19: **else if** $(Gra\_Direction(col, row) == 270°)$ **then**
20: $neighbor1 = Values(col, row - 1)$
21: $neighbor2 = Values(col, row + 1)$
22: **else if** $(Gra\_Direction(col, row) == 315°)$ **then**
23: $neighbor1 = Values(col - 1, row - 1)$
24: $neighbor2 = Values(col + 1, row + 1)$
25: **end if**
26: **return** [$neighbor1$,$neighbor2$]

Figure 4.6: Hysteresis Thresholding



(a) Non-maxima Suppression in 3D

(b) Hysteresis Thresholding in 3D

are immediately accepted as generalized edge pixels ($E()$), gradient values below $T2$ are immediately rejected. Pixels whith gradient values between $T1$ and $T2$ must be evaluated in order to decide wether or not they belong to a generalized edge ($E()$). In order to be part of an $E()$ a pixel must be $C_8()$ to a pixel that is either connected -in the *path* of a pixel with gradient above $T1$- or $N_8()$ of a pixel with $\nabla()$ value above $T1$.
In Function 6, maxima values of the gradient are stored in array $Maxima(\ )$. Line 4, Function 6 shows the rejection of pixel $Maxima(col, row)$ because its value is below threshold $T2$. Rejection of a pixel is done by writing a zero to array $Maxima(\ )$.
Line 7, Function 6 is a condition to decide wether or not the current pixel is connected ($C_8()$) to a pixel with gradient ($\nabla()$) above threshold $T1$. Connectivity ($C_8()$) is known, by testing for non-zero values in the neighborhood of current pixel of array $Maxima(\ )$. If the pixel is not connected $C_8()$ then its value must be above $T1$ (line 9, Function 6) in order for the pixel to be considered an $E()$. Otherwise it is suppressed (line 10, Function 6).
Output of this step is an array of Generalized edges stored in array *maxima()* where Generalized edges are pixel values different from zero.

Vectorized Edge Synthesis

It is the transformation from a data structure representation, in the present case "raster", into vector representation. This process is used to reduce the amount of data that describes an image. Figure 4.7 shows how a Generalized Edge (*E()*) is transformed into three straight edges (*e( )*).
The Algorithm divides a Generalized Edge (*E()*) into two parts, until the parts fulfill a given condition. Considering a Generalized Edge consisting of a sequence of edge

Function 6 Hysteresis Thresholding

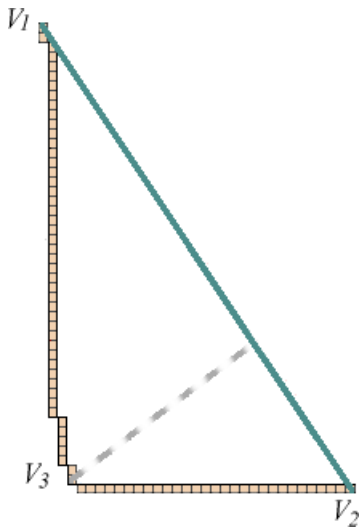
Hysteresis_Thresholding($Maxima(\ )$, $T2$, $T1$, $width$, $height$)

Input: $T2$: Lower threshold
$T1$: Higher threshold
$width$: Total number of columns
$height$: Total number of rows

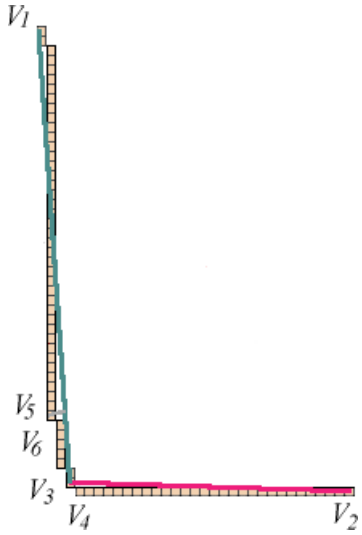
Input-Output: $Maxima(\ )$: Array of maxima values of gradient

1: **for** $row = 1$ to $height - 1$ **do**
2: **for** $col = 1$ to $width - 1$ **do**
3: {Pixels whose gradient is below $T2$ are rejected:}
4: **if** $Maxima(col, row) \leq T2$ **then**
5: $Maxima(col, row) = 0$
6: {Testing for non-concectivity ($C_8()$) of current pixel:}
7: **else if** $(Maxima(col - 1, row - 1) = 0)$ **and**
$(Maxima(col, row - 1) = 0)$ **and**
$(Maxima(col + 1, row - 1) = 0)$ **and**
$(Maxima(col - 1, row) = 0)$ **then**
8: {Testing the neighborhood ($N_8()$) for pixels with gradient below $T1$:}
9: **if** $(Maxima(col, row) < T1)$ **and**
$(Maxima(col + 1, row) < T1)$ **and**
$(Maxima(col - 1, row + 1) < T1)$ **and**
$(Maxima(col, row + 1) < T1)$ **and**
$(Maxima(col + 1, row + 1) < T1)$ **then**
10: $Maxima(col, row) = 0$
11: **end if**
12: **end if**
13: **end for**
14: **end for**

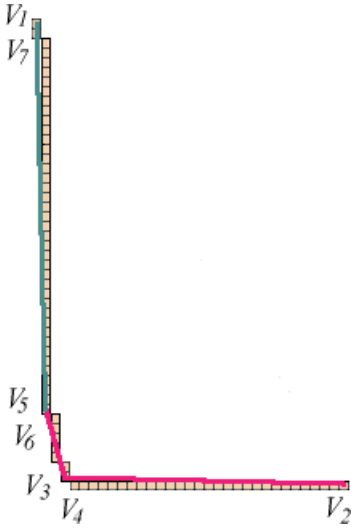
Figure 4.7: Approximation by straight lines



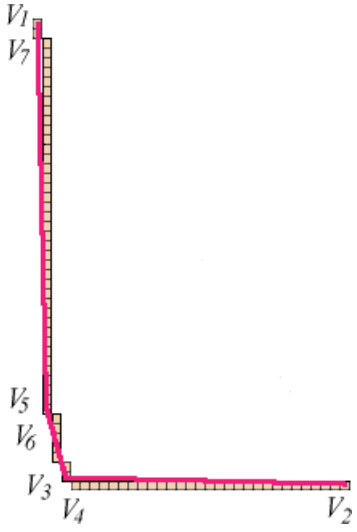
(a) Rough Approximation between v_1 and v_2



(b) Creation of a straight edge between v_2 and v_3



(c) Creation of a straight edge between v_3 and v_5



(d) Creation of a straight edge between v_5 and v_1

pixels $v_1, v_2, ..., v_n$, where v_1 and v_n are the endpoints. Pixels v_1 and v_n are joined by a straight line. For each pixel of the Generalized Ege, the distance to the straight line is calculated. If the maximum distance is larger than a given threshold the Generalized Edge is divided into two new Generalized Edges. The Generalized Edge is divided at the position where the maximum distance is found.

The process is composed of three steps:

i. Initial approximation. A straight line is drawn accros endponits of Generalized Edge ($E(v_1, v_2)$ in Figure 4.7-a).

ii. Distance vertex-line calculation. Euclidean distance is calculated for all pixels comprising the generalized edge. The longest distance calculated is retained as well as the pixel at which the longest distance is measured (lines 7-8, Function 7).

iii. Tolerance filtering. The longest distance, must be below threshold s_t. In case the longest distance is below this threshold there is no need for subdivision of the current generalized edge (line 13, Algorithm 7). Therefore endpoints v_0 and v_f are retained as endpoints of a new element: an edge that its described by its two vertices $e(v_o, v_f)$. The rest of the vertices are eliminated and the process stops. Otherwise, the generalized edge $E(v_o, v_f)$ is subdivided in two shorter generalized edges: $E(v_0, v)$ and $E(v, v_f)$ (lines 16, 17 Function 7).

The procedure is repeated recursively, until the approximation is good for all segments comprising the original Generalized Edge.
Figure 4.7 shows the process of Vectorized Edge Synthesis. In Figure 4.7-a the longest distance (dotted line) from the rough approximation (solid line) to $E(v_1, v_2)$ is $d(v_3, \overline{v_1 v_2})$ = 23.58. Since threshold $s_t = 1.0$, the Generalized Edge $E(v_1, v_2)$ is subdivided into two segments: $\overline{v_1, v_3}$ and $\overline{v_3, v_2}$. Then the longest distance is calculated for segment $\overline{v_3, v_2}$, $d(v_4, \overline{v_3 v_2}) = 0.96$, which is below the treshold s_t. Therefore, segment $\overline{v_3, v_2}$ is no further subdivided (Figure 4.7-b) and it is transformed into an edge: $e(v_3, v_2)$. The longest distance calculated at segment $\overline{v_1, v_3}$ happens at pixel v_5, since $d(v_5, \overline{v_1 v_3}) = 1.55$ is above s_t, then the segment $\overline{v_1, v_3}$ is subdivided into: $\overline{v_1, v_5}$ and $\overline{v_5, v_3}$ (Figure 4.7-c). The longest distance found for these pair of segments is below the threshold s_t. Therefore, the Algorithm stops, and two new edges are obtained: $e(v_1, v_5)$ and $e(v_5, v_3)$. Generalized Edge $E(v_1, v_2)$ is transformed into three straight edges: $e(v_1, v_5)$, $e(v_5, v_3)$ and $e(v_3, v_2)$ as shown in Figure 4.7-d.
For additional information on this Algorithm refer to [3].

4.1.2 Direct Corner Extraction From Image

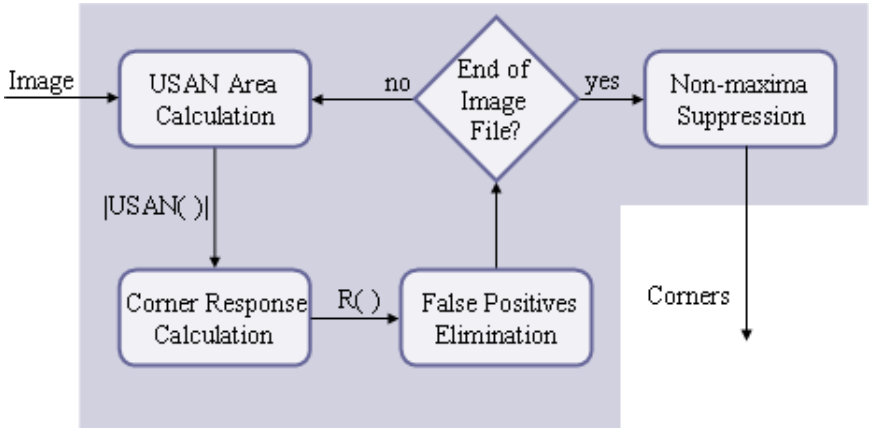
It is the identification of corner pixels directly from an image. The Direct Corner Extraction is seen in Figure 4.1 as a part of the methodology workflow. The corner detector implemented is based on S.U.S.A.N. corner detector ([12]).

Function 7 Vectorized Edge Synthesis

$[e()]$ = **Vectorized_Edge_Synthesis**(E(v_0,v_f))
Input: $E\,(v_0, v_f)$: Array of generalized edges
Output: $e()$: Array of edges where each row holds the 2D coordinates of the initial and final vertices of an edge

1: $maximum\_distance = 0$
2: **for** $v_i \in E\,(\,)$ **do**
3: {Euclidean distance from each pixel of $E()$ to the straight line is calculated: }
4: $distance = d\,(v, \overline{v_0\ v_f})$
5: {maximum distance is found: }
6: **if** $distance > maximum\_distance$ **then**
7: $maximum\_distance = distance$
8: $v = v_i$
9: **end if**
10: **end for**
11: {maximum distance is tested against the threshold s_t:}
12: **if** $maximum\_distance < s_t$ **then**
13: **return** $[e\,(v_0, v_f)]$
14: **else**
15: {$E()$ is divided into two parts:}
16: **return** $e\,() =$ **Vectorized_Edge_Synthesis**$(E(v_0, v))$
17: **return** $e\,() =$ **Vectorized_Edge_Synthesis**$(E(v, v_f))$
18: **end if**

Figure 4.8: Block diagram of Direct Corner Extraction from a given Image *I()*



Given a Circular Mask ($M()$) delimiting a circular region of the image, the Univalue Segment Assimilating Nucleus (USAN) area is the area of the Circular Mask made up of pixels similar in intensity to the intensity of the nucleus (v_0) of the mask. In a digital image, the USAN area will reach a minimum when the nucleus lies at a corner point. SUSAN is not sensitive to noise and is very fast for it only uses very simple operations. Required steps (see Figure 4.8) for this process are: (i) USAN Area Calculation, (ii) Corner Response Calculation, (iii) False Positives Elimination and (iv) Non-maxima Suppression. In the following paragraphs each one of these steps will be explained.

i. **USAN Area Calculation.**
Pixels whose intensity value is similar to the nucleus of the circular mask (M()) are found and counted in this step. The delta intensity function $\Delta_{v_0,I_t}(v_i)$ applies the threshold ΔI_t to the pixels of the mask in order to find the $USAN()$ area, as shown in line 3 Function 8.

ii. **Corner Response Calculation.**
The Corner Response ($R(v_0)$) is a measure of the cornerness of the nucleus of the mask. The bigger the value of the corner response the larger the possibility for the nucleus to be considered a corner. Corner response is determined by the threshold g_t and the USAN size (line 1, Function 9). If the USAN size is below this threshold (line 2, Function 9), it means that the nucleus is considered a candidate corner. Otherwise it is not considered a candidate corner and its corner response is set to zero (line 4, Algorithm 9).

iii. **False Positives Elimination.**
False Positives are points wrongly reported as corners. Cases like noise, lines across the circular mask ($M()$) may be reported as corners. Hence the aim of the

Function 8 USAN Area Calculation

$[usan\_size]$ = **USAN_Area_Calculation**($I()$, $M()$, v_0, ΔI_t)

Input: $I()$: Image
$M()$: Circular mask
v_0: nucleus of the mask $M()$
ΔI_t: Intensity difference threshold

Output: $usan\_size$: cardinality of $USAN()$

1: $usan\_size = 0$
2: **for** every v_i in $M()$ **do**
3: **if** $(\delta_{v_0,\Delta I_t}(v_i) == 1)$ **then**
4: $usan\_size = usan\_size + 1$
5: **end if**
6: **end for**
7: **return** $usan\_size$

Procedure 9 Corner Response Calculation

$[corner\_response]$ = **Corner_Response_Calculation**($usan\_size$, g_t)

Input: $usan\_size$: Cardinality of $USAN()$
g_t: Geometrical threshold

Output: $corner\_response$: Number ($R(v_0)$)

1: **if** $usan\_size < g_t$ **then**
2: $corner\_response = g_t - usan\_size$
3: **else**
4: $corner\_response = 0$
5: **end if**
6: **return** $corner\_response$

False Positives Elimination is to detect and eliminate such points.
In order to detect false positives, the center of mass of the USAN() is found (line 1, Procedure 10). Then, the Euclidean distance (D_E) is calculated based on the center of mass and the nucleus (v_0) (line 2, Procedure 10). If *distance* is below 1 then a false positive is reported (line 11, Procedure 10) by setting its corner response to zero.
If *distance* is above or equal to 1 (line 3, Procedure 10) and any of the pixels in line from v_0 to center of mass does not belong to the $USAN()$, then the nucleus is reported as a false positive and its corner response is set to zero (line 7, Procedure 10).
Note: Calculation of false positives approximates continuous values of the center of mass ($CM(USAN())$) to discrete values of pixel: $(0, 0.5)$ to zero, and $[0.5, 1.0)$ to one.

Procedure 10 False Positives Elimination

False_Positives_Elimination($USAN()$, v_0, $corner\_response$)
Input: $USAN()$: USAN area
v_0: Nucleus of mask $M()$
Input-Output: $corner\_response$: Number ($R(v_0)$)

1: $center\_of\_mass = CM(USAN())$
2: $distance = D_E\,(center\_of\_mass, v_0)$
3: **if** $distance \geq 1$ **then**
4: {Search pixel by pixel from nucleus to center of mass:}
5: **for** $v_i = v_0$ to $center\_of\_mass$ **do**
6: **if** $v_i \notin USAN()$ **then**
7: $corner\_response = 0$
8: **end if**
9: **end for**
10: **else**
11: $corner\_response = 0$
12: **end if**

iv. **Non-maxima Suppression.**
Non-maxima Suppression is a process to detect a corner response ($R()$) that is not a maximum in a subwindow of 5×5 pixels. In order to know if the current pixel (x, y) (line 5, 11) has a value of corner response that is not a maximum, the subwindow is scanned and tested for values of $R()$ bigger than that of the current pixel (Function 12). If a value of $R()$ is found complying with such condition, then the current pixel is not considered a maximum and its corner response is set to zero (line 7, Procedure 11).
Finally, candidate corners are those pixels whose corner response has a value

different from zero.
Note: in case the current pixel has the same highest value as any of its neighbors in the subwindow, it is not to be considered a maximum. Therefore its corner response is set to zero.

Procedure 11 Non-maxima Suppression

Non-maxima_Suppression($R()$)
Input-Output: $R()$: Array of corner response
Precondition: Corner response ($R()$) has been found for all pixels of the image

1: $Y_r = 480$ (Total number of rows)
2: $X_c = 640$ (Total number of columns)
3: **for** $x = 2$ to $X_c - 2$ **do**
4: **for** $y = 2$ to $Y_r - 2$ **do**
5: $maximum =$ **find_maximum**$(R(), x, y)$ (Function 12)
6: **if** $maximum == 0$ **then**
7: $R(x, y) = 0$
8: **end if**
9: **end for**
10: **end for**

4.1.3 Edge-corner Alignment

This process aims at putting together the detected corners and edges. The guiding principle is the relationship of colinearity between an edge and its close corners. In case a corner is aligned with an edge, one of the following three cases may occur:

- the corner falls inside the edge,
- the corner falls at an endpoint of the edge,
- the corner falls outside the edge.

Assumption: for the current problem, the first two cases are of no interest because they do not yield useful information. In contrast, the last case enables the contouring of the missing corners.
In order to find a colinear corner, the distance between the corner $c()$ and each of the vertices of the current edge $e()$ must be calculated (lines 1, 2 Procedure 13). This step helps to ensure that neighboring corners are chosen for the Edge-corner Alignment. Corners which are not farther than threshold ec_t (line 3, Procedure 13) are tested for colinearity (line 4, Procedure 13). If the corner is colinear to the edge, then the edge is extended to meet the corner (line 5, Procedure 13).

Function 12 Find Maximum

$[maximum]$ = **Find_Maximum**($R()$, x, y)

Input: $R()$: Array of corner response
x, y: Central pixel

Output: $maximum$: Boolean value to determine wether or not the central pixel is a maximum

```
1: maximum = 1
2: for i = x − 2 to i < x + 2 do
3:   for j = y − 2 to j < y + 2 do
4:     if (i ≠ x) and (j ≠ y) then
5:       if (R(x, y) < R(i, j)) then
6:         maximum = 0
7:         i = x + 2
8:         j = y + 2
9:       end if
10:    end if
11:  end for
12: end for
13: return maximum
```

The same procedure must be repeated for all edges $e()$ obtained in the Edge Vectorization process (4.1.1).
For additional information on this Algorithm refer to [5].

Procedure 13 Colinear corner-edge Attachment

Colinear_Corner_Edge_Attachment($corner()$, $e()$)

Input: $corner()$: Corner

Input-Output: $e()$: Edge

```
1: distance_v0 = D_E(corner(), e(v0, vf))
2: distance_vf = D_E(corner(), e(v0, vf))
3: if ((distance_vo ≤ ec_t) or (distance_vf ≤ ec_t)) then
4:   if (Alignment(corner(), e()) == OUTSIDE)(Function 14) then
5:     e() = extend(e(), corner())
6:   end if
7: end if
```

Function 14 Alignment

value = **Alignment**(*corner*(), *e*())
Input: *corner*(): Corner
e(): Edge
Output: *value*: String, it represents the position of a corner respect to an edge

1: $colinear = colinearity\,(e\,(),corner\,())$
2: **if** $(colinear == 0)$ **or** $(colinear == 1)$ **then**
3: $value = ENDPOINT$
4: **else if** $(colinear > 0)$ **or** $(colinear < 1)$ **then**
5: $value = INSIDE$
6: **else if** $(colinear < 0)$ **or** $(colinear > 1)$ **then**
7: $value = OUTSIDE$
8: **end if**
9: **return** *value*

4.2 Optimization

Results obtained in the Edge-corner Alignment process were insufficient. Many edges remained without a common corner after the Edge-corner Alignmet process. Besides, processing time was too long. Therefore, an Optimization was implemented. The Optimization consists of: (i) The use of 1D filters instead of 2D filters for the process of Identification of Edges in Pixel Domain and (ii) Indirect Corner Calculation based on existent edges.

i. The use of 1D filters instead of 2D filters for the process of Identification of Edges in Pixel Domain.
The order of magnitude for an algorithm based on 1D filter is:
$O_1(m * X_c * Y_r)$,
The order of magnitude for an algorithm based on 2D filter is:
$O_2(m^2 * X_c * Y_r)$, where
m = filter side,
X_c = image width,
Y_r = image height.

$\Rightarrow O_1(m * X_c * Y_r) < O_2(m^2 * X_c * Y_r)$

Advantages of 1D filter:

(a) More efficient.

Disadvantages of 1D filter:

(a) Fine detail remains.

(b) Picture orientation sensitive.

The optimized process of Edge Vectorization is divided into:

(a) Identification of Edges in Pixel Domain. It is the process already explained in section Edge Vectoriation (4.1.1). The only difference is that 1D filters are used instead of 2D filters for steps: Gaussian Convolution and First Derivative Convolution.

(b) Vectorized Edge Synthesis. It is the same process explained in section Edge Vectorization (4.1.1).

ii. Indirect Corner Calculation. Based on the fact that images being processed are made up of straight lines, corners are calculated by finding the intersection point ($p_i()$) between two non-parallel edges. For a given pair of edges to be related through a common corner they must be closer than a given threshold. Two edges being far away from each other are not sought to be related by a corner.

4.2.1 Identification of Edges in Pixel Domain

This process has already been explained in section 4.1.1. It is composed of four steps:

i. Gaussian Convolution

ii. First Derivative Convolution

iii. Non-maxima Suppression

iv. Hysteresis thresholding

The steps Gaussian Convolution and First Derivative Convolution are the only steps that will be explained further. In order to show the use of 1D filters in the convolution processes.

i. **Gaussian Convolution with 1D filter.**
Algorithm 15 uses a 1D Gaussian operator ($G_{\Delta x,\sigma}()$), therefore only one loop is required in order to perform the convolution as shown in lines 7 - 11, Algorithm 15. Comparing the 1D filter convolution loop and 2D filter convolution loop (lines 8 - 17, Algorithm 2), it can be seen that the convolution in Algorithm 2 performs two loops.

Note that a horizontal 1D filter was used in this step. Horizontal filters affect vertical features of the image.

Algorithm 15 Gaussian Convolution with 1D filter

$[Smoothed\_Image()]$ = **Gaussian_Convolution_1D**($G_{\Delta x,\sigma}()$, $I()$, σ, X_c, Y_r)

Input: $G_{\Delta x,\sigma}()$: 1D Gaussian operator
$I()$: Image
σ: Standard deviation
X_c: Total number of columns
Y_r: Total number of rows

Output: $Smoothed\_Image()$: Image softened by the Gaussian convolution

Precondition: $\sigma \neq 0$

1: {$I()$ is scanned pixel by pixel in the two following loops}
2: **for** $row = 3\sigma$ to $Y_r - 3\sigma$ **do**
3: **for** $col = 3\sigma$ to $X_c - 3\sigma$ **do**
4: $x = 0$
5: $k = 0$
6: {Convolution is done in following loop: $G_{\Delta x,\sigma}(k)$ is convolved with $I(i, row)$}
7: **for** $i = col - 3\sigma$ to $i \leq col + 3\sigma$ **do**
8: $Smoothed\_Image(x + col, row) = Smoothed\_Image(x + col, row) + I(i, row)\, G_{\Delta x,\sigma}(k)$
9: $k = k + 1$
10: $x = x + 1$
11: **end for**
12: **end for**
13: **end for**
14: **return** $Smoothed\_Image(\ )$

ii. **First Derivative Convolution with 1D filter.**
1D filters are employed in this step. In the present case the orthogonal diferences $\Delta_i I\,(col, row)$ and $\Delta_j I\,(col, row)$ are used in the convolution (lines 7, 9, Algorithm 16). These 1D directional derivatives become 2D by embedding them in a loop as shown in lines 7-9 and lines 11-13, Algorithm 3 in the First Derivative Convolution with 2D filter.

The horizontal directional derivative ($\Delta_i I\,(col, row)$) and the vertical directional derivative ($\Delta_j I\,(col, row)$) are the basic elements for gradient calculation.

Algorithm 16 First Derivative Convolution with 1D filter

$[Gra\_Magnitude\,(\,),Gra\_Direction\,(\,)]$ = **First_Derivative_Convolution_1D**($I()$, $width$, $height$)

Input: $I()$: Image
$width$: Total number of columns of image
$height$: Total number of rows of image

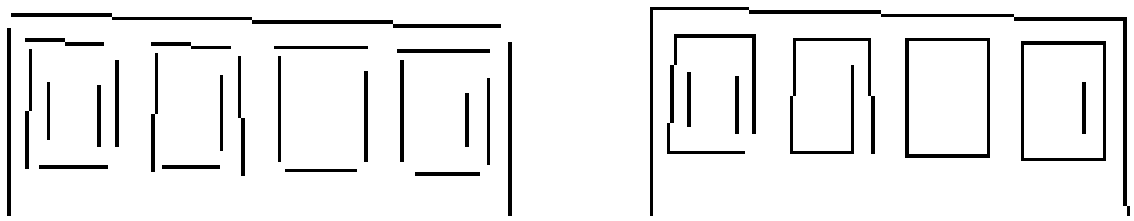
Output: $Gra\_Magnitude(\,)$ Array of gradient magnitude
$Gra\_Direction(\,)$ Array of gradient direction

1: $xComponent(width, height) = 0$
2: $yComponent(width, height) = 0$
3: {Scan $I\,()$ pixel by pixel:}
4: **for** $row = 1$ to $height - 1$ **do**
5: **for** $col = 1$ to $width - 1$ **do**
6: {Find horinzontal component of directional derivative:}
7: $xComponent(col, row) = \Delta_i I\,(col, row)$
8: {Find vertical component of directional derivative:}
9: $yComponent(col, row) = \Delta_j I\,(col, row)$
10: {Find gradient magnitude $\nabla\,()$:}
11: $Gra\_Magnitude(col, row) = |\nabla\,(yComponent(col, row), xComponent(col, row))\,|$
12: {Find discretized gradient direction $\lfloor\theta\,()\rfloor$:}
13: $Gra\_Direction(col, row) = \lfloor\theta\,(yComponent(col, row), xComponent(col, row))\rfloor$
14: **end for**
15: **end for**
16: **return** $[Gra\_Magnitude\,(\,),Gra\_Direction\,(\,)]$

4.2.2 Indirect Corner Calculation

Based on the fact that images being processed are made up of straight lines, *corners* are calculated by finding the point of intersection ($p_i\,()$) between two non-parallel *edges*.

Figure 4.9: Corner calculation based on existent edges



(a) Edges (b) Edges and calculated corners

In order for edges to qualify, distance between any pair of vertices of two edges must not exceed a given threshold.
In order to perform corner calculation there must be a set of vectorized edges ($e()$). Pairs of edges to be used for the corner calculation process have to comply with: (i)they must not be parallel and, (ii) the euclidean distance (D_E) between two vertices of the pair of edges must be below a user defined threshold. Line 3, Algorithm 17 shows this classification. Once a pair of edges fit into the classification, the point of intersection $p_i\,()$ is calculated (line 4, Algorithm 17). To end with, edges are extended to meet the calculated corner (lines 5, 6 Algorithm 17).
Detail of an image after the Edge Vectorization process is shown in Figure 4.9-a. The missing corners are noticeable. Figure 4.9-b shows the same image after the corner calculation process. It can be seen the well delineated corners after the process.
Assumption: The largest distance between a pair of vertices below the threshold is used to calculate the point of intersection of two edges. This means that if two pairs of vertices qualify, the longest distance will be chosen for the corner calculation.

Algorithm 17 Indirect Corner Calculation

Indirect_Corner_Calculation(T, $e_1()$, $e_2()$)
Input: T: Threshold
Input-Output: $e_1(), e_2()$: Edges to intersect

1: $dot\_product = e_1().e_2()$
2: {Non-parallel edges are intersected:}
3: **if** (($Under\_Threshold(T, e_1, e_2)$ (Function 18)) **and** (($dot\_product <> -1$) **or** ($dot\_product <> 1$))) **then**
4: $intersection\_point = p_i(e_1, e_2)$
5: $extend(e_1(), intersection\_point)$
6: $extend(e_2(), intersection\_point)$
7: **end if**

Function 18 Is Maximum Distance under Threshold

$[under]$ = **Under_Threshold**(T, $e_m(v_1, v_2)$, $e_n(v_3, v_4)$)

Input: T: Threshold
$e_m(v_1, v_2)$, $e_n(v_3, v_4)$: Edges to be tested

Output: $under$: Boolean, the distance between the edges is under the given threshold T or not

1: $under = false$
2: $max = 0$
3: $D_{13} = D_E(v_1, v_3)$
4: $D_{14} = D_E(v_1, v_4)$
5: $D_{23} = D_E(v_2, v_3)$
6: $D_{24} = D_E(v_2, v_4)$
7: **if** $(D_{13} < T)$ **then**
8: $max = D_{13}$
9: $under = true$
10: **end if**
11: **if** $(D_{14} < T$ **and** $max < D_{14})$ **then**
12: $max = D_{14}$
13: $under = true$
14: **end if**
15: **if** $(D_{23} < T$ **and** $max < D_{23})$ **then**
16: $max = D_{23}$
17: $under = true$
18: **end if**
19: **if** $(D_{24} < T$ **and** $max < D_{24})$ **then**
20: $max = D_{24}$
21: $under = true$
22: **end if**
23: **return** $under$

4.3 Parameters

This section is about the use of parameters in the implemented methodology. Processes are enhanced by the use of parameters. Parametrization enables a process to work with images of varying characteristics. The parameters are exposed according to the methods that use them:

i. Edge Vectorization
Parameters: Thresholds $T1$ and $T2$, σ, s_t, mask.

ii. Direct Corner Extraction
Parameters: ΔI_t, g_t.

iii. Indirect Corner Calculation
Parameters: Vertex-vertex distance.

Figure 4.10: Original image

Figure 4.10 is a 3D model view of a building of the University of Heidelberg. This image is going to be used to show the results obtained depending on the varying values of the parameters.

i. Edge Vectorization
Four cases with fixed and variable parameters will be presented in this section.

(a) Fixed parameters: T1 and T2: 400 - 300; s_t: 2; 2D masks
Variable parameters: σ

Figure 4.11: Edge Vectorization with Variable values of σ

(a) $\sigma = 0.0$

(b) $\sigma = 2.0$

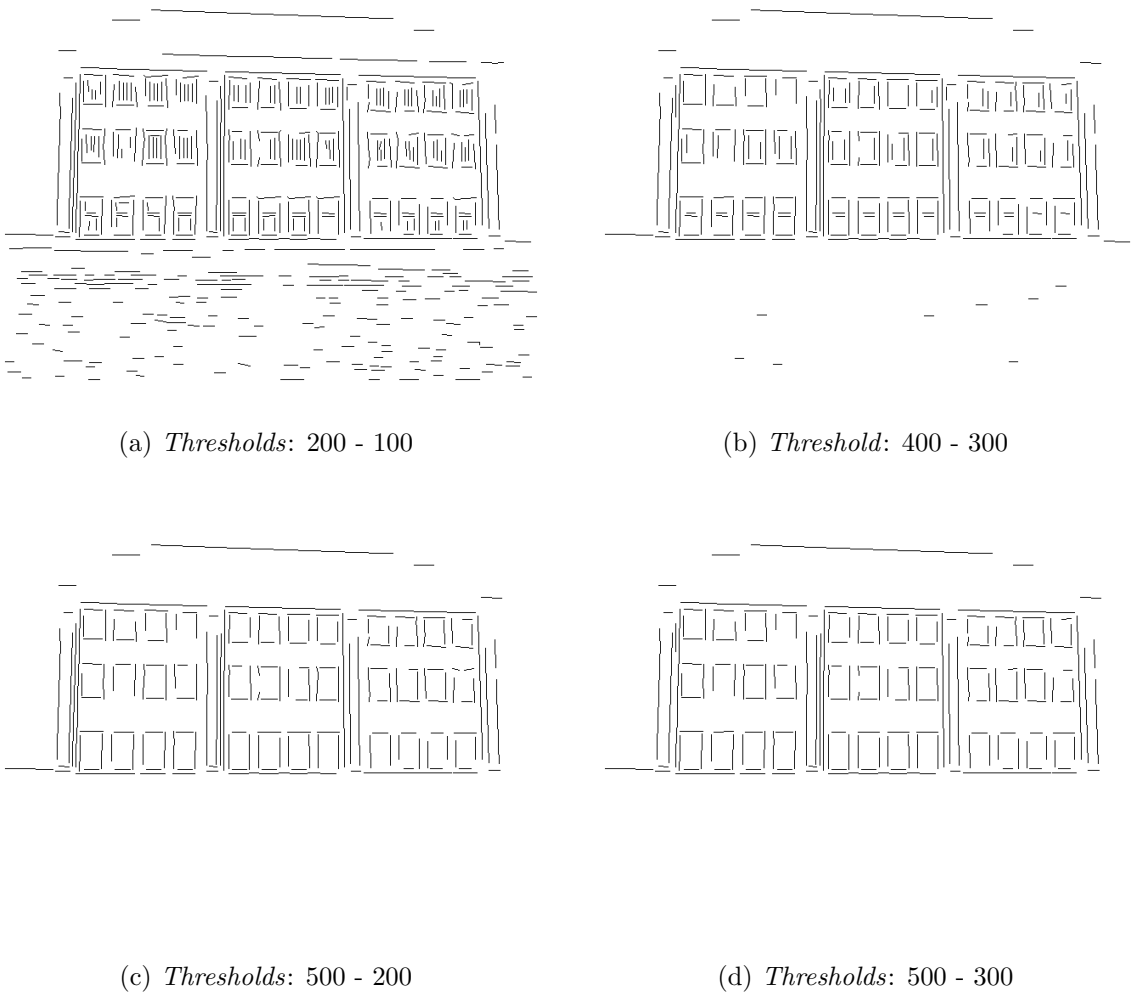
(c) $\sigma = 4.0$

(d) $\sigma = 6.0$

In Figure 4.11 edges obtained are affected by σ. Details tend to disappear for high values of σ (Figure 4.11-c,d). Though corners are suppressed in the Edge Identification process, the suppression is even stronger for high values of σ. For the purpose of this work edges must be devoid of fine detail, but on the other hand the smoothing must not be so much as to misshape the features, as in the case of Figure 4.11-d. A good value of σ for the present case is 2.0 as in (Figure 4.11-b).

(b) Fixed parameters: σ: 2.0; s_t: 2; 2D masks
Variable parameters: T1,T2

Figure 4.12: Edge Vectorization with variable values of thresholds T1 and T2



(a) *Thresholds*: 200 - 100

(b) *Threshold*: 400 - 300

(c) *Thresholds*: 500 - 200

(d) *Thresholds*: 500 - 300

The use of these thresholds aims at controling line streaking. To determine

the values of thresholds T1 and T2, it is important to start by giving a very high value to both thresholds (i.e 500). Once the edges obtained are as expected, the level of detail must be graduated by lowering T2, until line streaking is very little. Good values for thresholds in present case are: 400 - 300 (Figure 4.12-b) and 500 - 300 (Figure 4.12-d).
Note that the value of these thresholds is dependent on the values of the gradient. If 1D masks are used for the edge detection, the calculation of the gradient results in lower values. Therefore the thresholds must be set to an initial lower value, i.e 100.

(c) Fixed parameters: T1, T2: 400 - 300
Variable parameters: σ vs. s_t

The parameter s_t, which determines the length of the edges obtained in the Edge Vectorization process. The parameter σ determines the degre of detail in the edges obtained.
There must be an equilibrium in these two parameters: for low values of σ, low values of parameter s_t are required, as shown in Figure 4.13-a. Otherwise edges obtained are misshaped (Figure 4.13-b). For higher values of σ, the parameter s_t may have low or high values as shown in Figure 4.13-c, d. However, big values of the parameter s_t, should result in longer Vectorized Edges which is an advantage for the Matching Algorithm.

(d) Fixed parameters: σ: 2.0; s_t: 2
Variable parameters: T1, T2, mask

The parameters mask determine the use of 2D or 1D filters for the edge extraction process. 2D masks are bigger, hence the smothing is stronger. It means less detail is obtained in the final edges, as shown in Figure 4.14-a. Figure 4.14-b shows results of the edge detection with 1D masks.

ii. Direct Corner Extraction
Variable parameters: ΔI_t vs. g_t

Figure 4.15 shows corners obtained from the Direct Corner Extraction process. The value 15 for parameter g_t is a good value, because the most significative corners are detected (Figures 4.15-b, d). Low values of parameter ΔI_t are used to find corners in images where there is little contrast, hence mthe amount of detected corners increases (Figure 4.15-a, b). Table 4.1 shows the number of corners detected for each case of Figure 4.15.

iii. Direct Corner Extraction and Edge Vectorization
Fixed parameters: T1 and T2: 400 - 300, σ: 2, s_t: 10

Figure 4.13: Edge Vectorization with variable values of σ and s_t

(a) σ vs. s_t : 0.0 - 1

(b) σ vs. s_t : 0.0 - 10

(c) σ vs. s_t : 2.0 - 1

(d) σ vs. s_t : 2.0 - 10

Figure 4.14: Edge Vectorization using variable filters

(a) 2D mask, T1 = 400, T2 = 300

(b) 1D mask, T1 = 150, T2 = 120

Tabla 4.1: Corner Detection Thresholds and Number of Detected Corners

ΔI_t	g_t	Detected Corners
20	28	3913
20	15	1943
60	28	1163
60	15	286

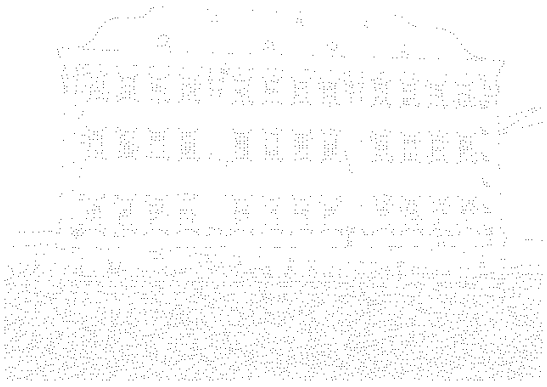
Variable parameters: ΔI_t vs. g_t : 20-15, 20-28, 60-15, 60-28

Corner and edges must be matched by the Edge-corner Alignment process. Therefore the parameters used in each process play an important role in the output result. It can be seen in Figure 4.16-a that edges become misshaped in the process of Corner Alignment because of the large amount of detected corners. Cases shown in Figures 4.16-c, d give the best possible results.

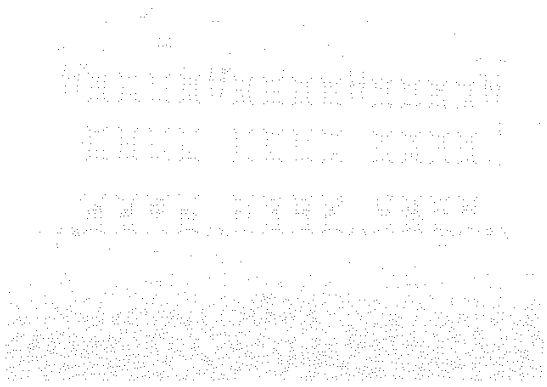
iv. Indirect Corner Calculation and Edge Vectorization based on 2D masks
Fixed parameters: T1 and T2: 400 - 300, σ: 2.0, s_t: 10, 2D mask
Variable parameters: *distance*: 15, 12, 8, 4

Figure 4.17 shows the corners and edges resulting from the Edge Vectorization and Corner Calculation processes. Parameter distance is variable. As shown in Figure 4.17-a large values for this parameter give corners at wrong locations. But

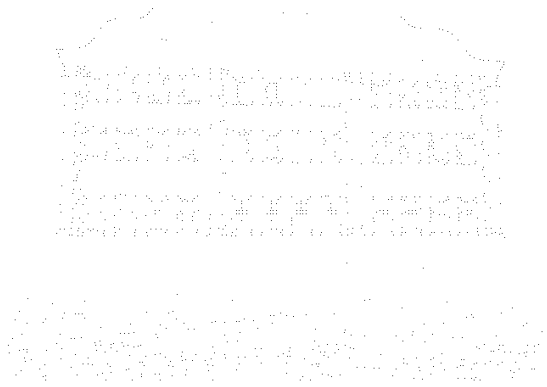
Figure 4.15: Corner detection with variable values of ΔI_t and g_t



(a) ΔI_t vs. g_t: 20-28



(b) ΔI_t vs. g_t: 20-15



(c) ΔI_t vs. g_t:60-28

(d) ΔI_t vs. g_t: 60-15

Figure 4.16: Corner-edge matching with variable values of ΔI_t and g_t

(a) ΔI_t vs. g_t: 20-28

(b) ΔI_t vs. g_t: 20-15

(c) ΔI_t vs. g_t:60-28

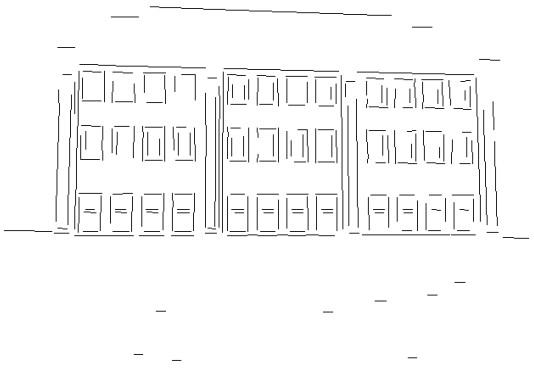
(d) ΔI_t vs. g_t: 60-15

Figure 4.17: Edge Vectorization using 2D filters and calculated corners with variable distance

(a) distance = 15

(b) distance = 12

(c) distance = 8



(d) distance = 2

low values for this parameter (Figure 4.17 -d) give little amount of corners in the output image. Appropiate values for this parameter are around 8 (as shown in Figure 4.17-c).

4.4 Experimental Setup

In this section the input parameters will be presented for a case of study which is developed in the next chapter.

The methodology described above will be applied to an image. The model was implemented in Java version jdk1.3.1_09. The prototype works off-line, it does not have connection to GPS, orientation tracker, video camera.

4.4.1 Case of study

This case of study shows results of the Edge Vectorization and Corner Extraction processes. The case presents results of the two methods implemented: the Initial Approach and the Optimization.

i. Initial Approach

   (a) Input data: \Data\Photo18.tif

   (b) Edge Vectorization parameters

   - $\sigma = 2.0$
   - $T1 = 400$
   - $T2 = 300$
   - $s_t = 2$
   - $mask = 2D$

   (c) Direct Corner Detection parameters

   - $\Delta I_t = 40$
   - $g_t = 15$

   (d) Function: Direct Corner Detection + Edge Vectorization

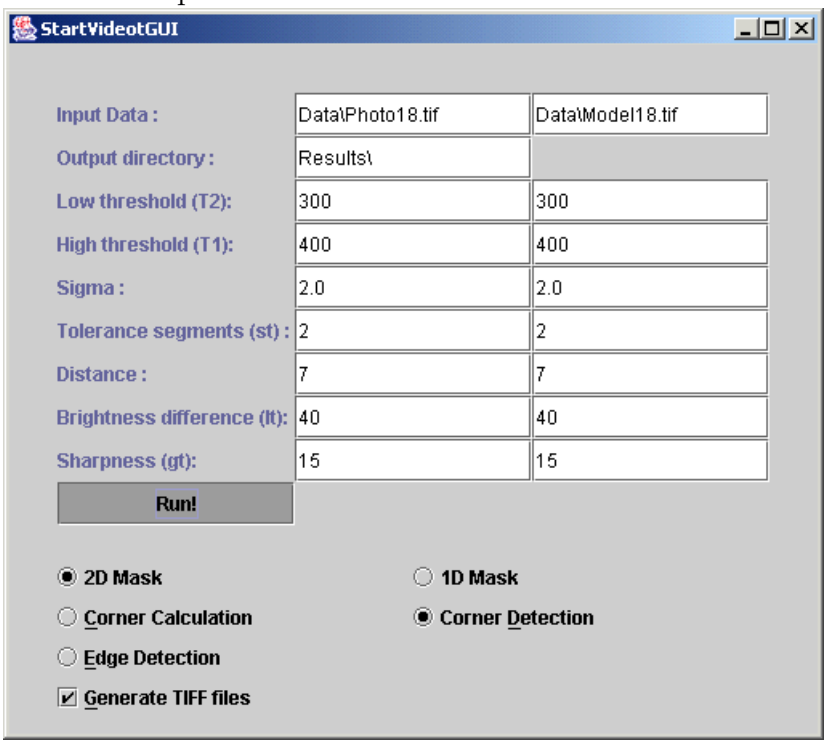
   (e) Classes used: StartVideotGUI.class (Graphical User Interface shown in Figure 4.18).

   (f) Output data: TIFF files (\Results\corPhoto18.tif, \Results\mgPhoto18.tif)

ii. Optimization based on 1D filters

   (a) Input data: \Data\Photo18.tif

Figure 4.18: Graphical User Interface of Class StartVideotGUI.class



(b) Edge Vectorization parameters

- $\sigma = 0$
- $T1 = 80$
- $T2 = 60$
- $s_t = 2$
- $mask = 1D$

(c) Function: Edge Vectorization

(d) Classes used:StartVideotGUI.class (Graphical User Interface shown in Figure 4.18)

(e) Output data: TIFF files (\Results\segPhoto18.tif)

iii. Optimization based on 2D filters

(a) Input data: \Data\Photo18.tif

(b) Edge Vectorization parameters

- $\sigma = 2.0$
- $T1 = 400$

- $T2 = 300$
- $s_t = 2$
- $mask = 2D$

(c) Indirect Corner Calculation parameters

- $distance = 7$

(d) Function: Corner Calculation + Edge Vectorization

(e) Classes used:StartVideotGUI.class (Graphical User Interface shown in Figure 4.18)

(f) Output data: TIFF files (\Results\longPhoto18.tif)

Chapter 5

Demostration application

Results of the implemented algorithms will be shown in this section. They are based on the photo shown in Figure 5.1. Results of the Initial Approximation algorithms will be presented in numeral (i) and results of the Optimization algorithms in numeral (ii) and (iii). At the end of this chapter there is a table comparing results of the two implemented methods.

Figure 5.1: Original Photo

i. Initial approximation
   Figure 5.2 presents detected corners result of the Direct Corner Extraction process applied to Figure 5.1. These results are shown superimposed on original image in figure 5.3-c. Even though corner localization is good, missing corners are noticeable. Many of the missing corners can be obtained by lowering the value of the parameter ΔI_t and raising the value of parameter g_t. But since these results are going to be used for the Edge-corner Alignment process, the setting of the parameters to the current values (i.e. $g_t = 15$ and $\Delta I_t = 40$) gives the best results for such process.

   Details of Figure 5.3-c are shown in Figures 5.3-a and 5.3-b. Arrows in the detailed figures point to the detected corners.

Figure 5.2: Initial Approach: Direct Corner Extraction

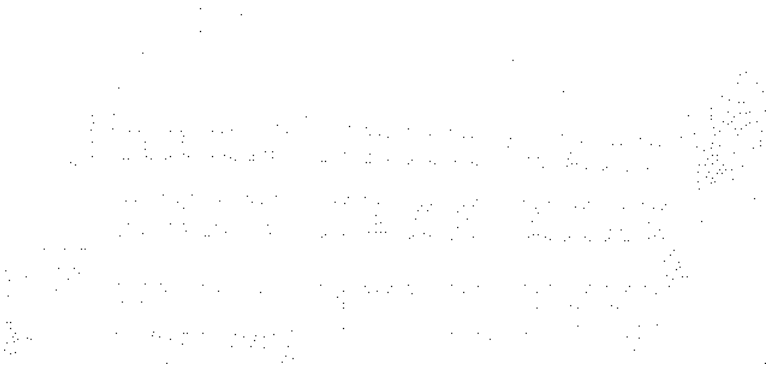


Figure 5.4-a shows edges result of the Edge Vectorization process. Notice that the most important features of the building are represented by the vectorized edges. Figure 5.4-b shows edges and aligned corners. Notice that the difference is not very significative with respect to the edges obtined in Figure 5.4-a.

Figure 5.5 shows vectorized edges and aligned corners superimposed on the original image. Notice the good localization of edges.

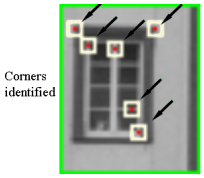
Figure 5.5-a shows a detail of figure 5.5-c. The detail shows a corner aligned to an edge (to the right of the window) and the edges delineating the window.

Figure 5.5-b shows a detail of the tower in Figure 5.5-c. The detected corners are pointed out by the arrows. These corners are lost in the final output (see Figure 5.5-c) because there are no vectorized edges in that region. Therefore, the process of Edge-corner Alignment produced no results in the region.

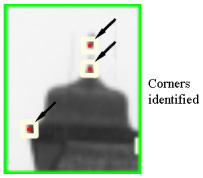
ii. Optimization based on 1D filters
Figure 5.6 shows results of the Edge Vectorization process based on 1D filters. Figure 5.6-a shows results of edge vectorization process with parameter $\sigma = 0$. In this figure there is *no* corner calculation: plain edge vectorization was applied to the image. Notice that most of the edges do have corners. The same can be seen in Figure 5.6-b, this figure was processed with parameter $\sigma = 5$. In this figure the corners begin to disappear because of the higher value of the parameter σ.

iii. Optimization based on 2D filters
Figure 5.7-a shows results of the Edge Vectorization process. Figure 5.7-b shows vectorized edges after the Corner Calculation process. In contrast with results shown in the numeral (i) Initial Approximation, the corner calculation improved the edge vectorization results to a great extent. Figure 5.8-a shows the same window seen in Figure 5.5-a.

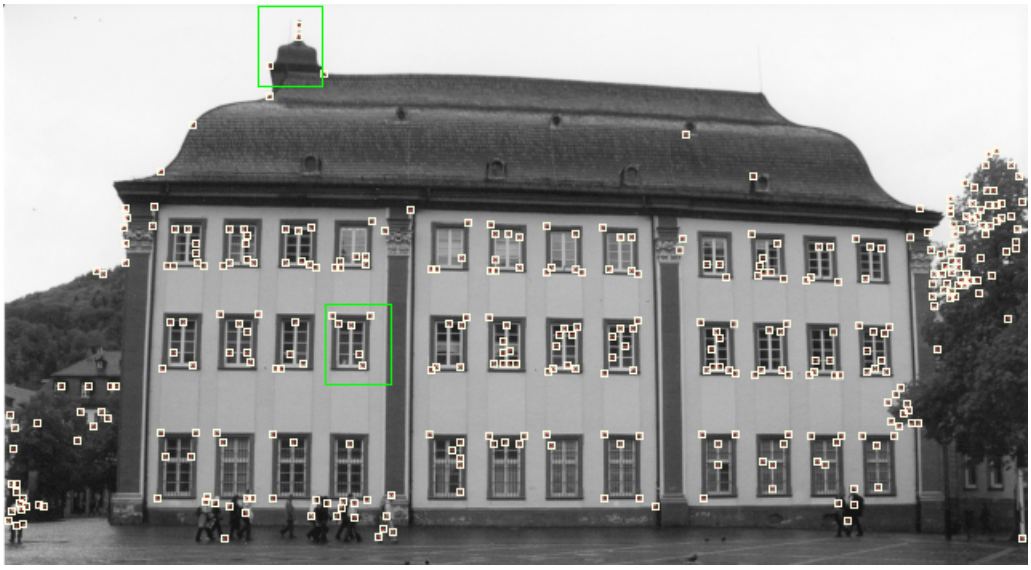
Figure 5.3: Initial Approach: Results of Direct Corner Extraction superimposed on Original Image (Figure 5.1)



(a) Detail of Figure 5.3-c with arrows pointing to detected corners



(b) Detail of image 5.3-c with arrows pointing to detected corners



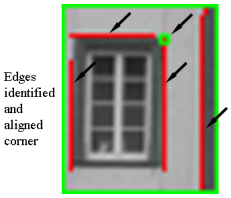
(c) Original image with superimposed corners

Figure 5.4: Initial Approach: Edge Vectorization and Edge-corner Alignment

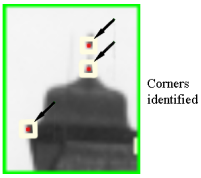
(a) Vectorized Edges

(b) Vectorized Edges and Aligned Corners

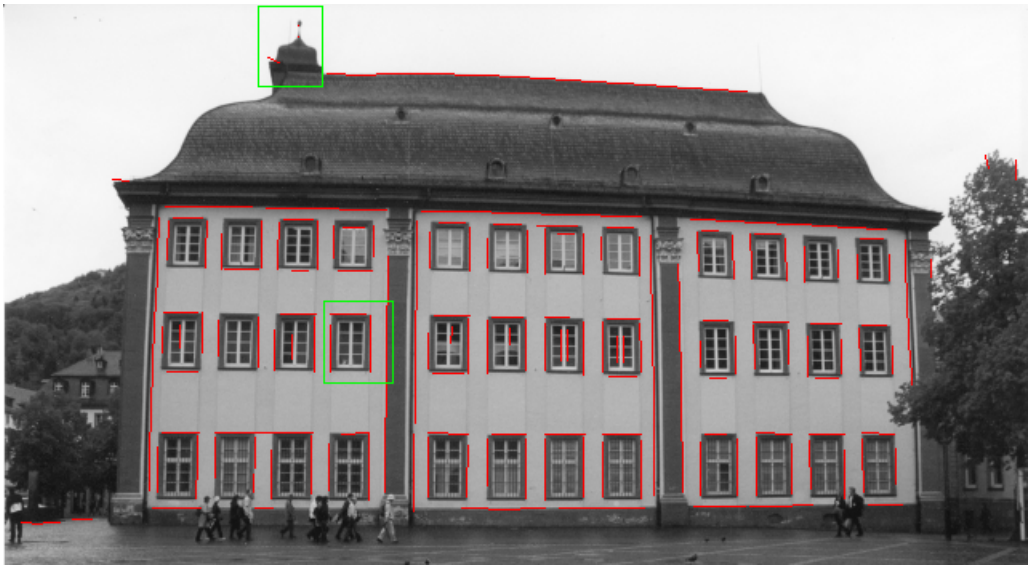
Figure 5.5: Initial Approach: Edge Vectorization and Edge-corner Alignment superimposed on Original Image (Figure 5.1)



(a) Detail of 5.5-c with arrows pointing to edges and corners



(b) Detail of image 5.5-c with arrows pointing to corners



(c) Original image with superimposed edges and aligned corners

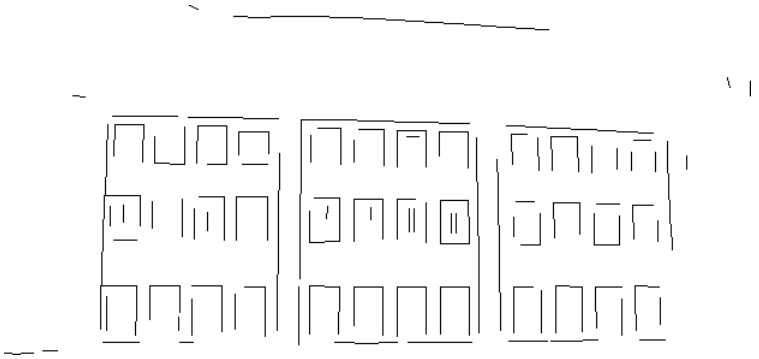
Figure 5.6: Optimization: Vectorized Edges using 1D filters with *no* Corner Calculation

(a) Vectorized Edges for $\sigma = 0$

(b) Vectorized Edges for $\sigma = 5$

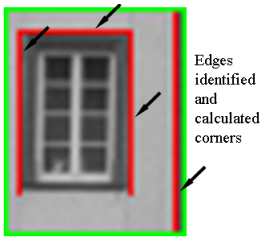
Figure 5.7: Optimization: Vectorized Edges using 2D filters and Calculated Corners

(a) Vectorized Edges



(b) Vectorized Edges and Calculated Corners

Figure 5.8: Optimization: Results of Edge Vectorization and Corner Calculation superimposed on Original Image (Figure 5.1)



(a) Detail of 5.8-b with arrows pointing to highlighted edges

(b) Original image with superimposed edges and calculated corners

Tabla 5.1: Corners Generated for Initial Approach vs. Optimization

	No. corners
Initial Approach Corner Detection	5
Optimization Corner Calculation	61

Tabla 5.2: Execution Times for Initial Approach vs. Optimization

	Time(s)
Initial Approach Edge Vectorization 2D + Corner Detection	23.2
Optimization Edge Vectorization 2D + Corner Calculation	8.1
Optimization Edge Vectorization 1D (requires no corner calculation)	4.5

In table 5.1 it is seen that corner calculation outperforms corner detection at generating corners for the existent edges. Processing time is greatly improved by the Optimization as shown in table 5.2.

Chapter 6

Conclusions

i. The algorithms developed are effective in generating Edges and Corners if the input image consists mainly of straight edges. Fortunaltely, the objects of interest in the application area"urban environment" are basically made of straight lines.

ii. The 3D model view and the video-frame may be images differing in contrast, illumination, intensity, etc. Therefore, the parameters to obtain Edges and Corners from such images must be given according to the characteristics of each image. Otherwise, the Matching Algorithm will not be able to identify the matching points in both images.

iii. In order to improve performance, 1D operators were applied to the Gaussian and First Derivatives steps of the process Identification of Edges in Pixel Domain, instead of 2D operators. The results achieved were satisfactory. As the processing time with 1D is 1/3rd of the processing time with 2D operators.

iv. Better performance was obtained by Indirect Corner Calculation from detected edges instead of Direct Corner Extraction from the image. In this case not only the processing time dropped to 40%, but also the number of corners generated was doubled.

v. However as the image grows the processing time does so. For large images a Divide and Conquer approach would allow to attack smaller images. Therefore having a good performance.

vi. The performance of the implemented algorithms is sensitive to the numerical parameters of them. In a future, an additional module should be written to make the setting of such parameters automatic.

vii. The Edge Vectorization based on 1D filters gives very good features. Somehow, fine detail is present in the resulting edges. This is not a recommended input for

the Matching Algorithm, which does not work very well in the presence of fine detail.

viii. In order to overcome the problem of the vanishing corners the Laplacian of Gaussian filter was tried instead of the Gaussian filter. But results were not as good as expected. Much better results were obtained with the use of 1D filters in the process of Edge Identification.

ix. Corner calculation could be improved by finding a neighboring corner resulting from the corner detection process to a calculated corner. In this way more accurate corners would be obtained.

Apendix A

Prototype of user-defined class *BC_Mesh2D*

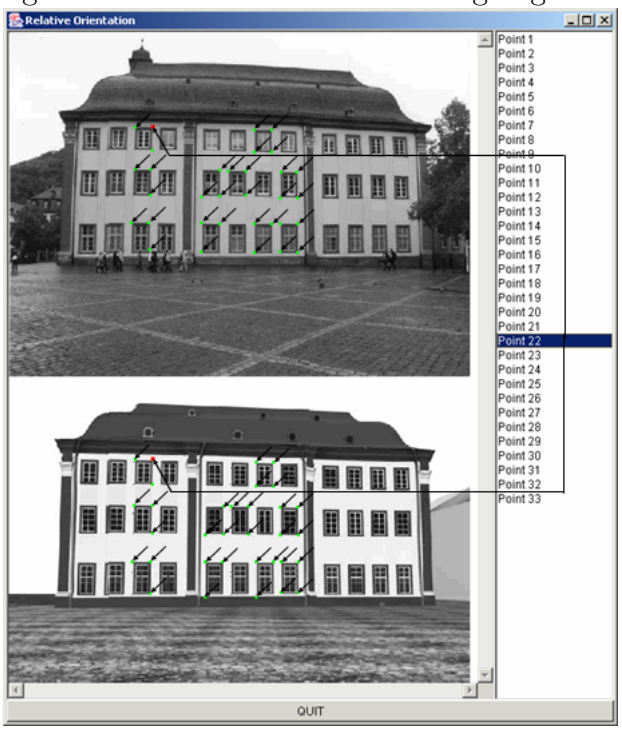
```
package de.fhg.igd.progis.geist.imagepreparation;

public class BC_Mesh2D{
  Vector vertices;
  Vector edges;
  int lpVertices;
  int lpEdges;

  public BC_Mesh2D();
  private BC_Vertex2D getVertex(int _id);
  public BC_Vertex2D getVertexNear(double _x, double _y,
                  double _xTolerance, double _yTolerance);
  public void setVertices(Vector _vertices);
  public void addEdge(int _a, int _b);
  public void addEdge(BC_Vertex2D _a, BC_Vertex2D _b);
  public void addEdge(double _x1, double _y1, double _x2,
                   double _y2, int _id1, int _id2);
  public int countVertices();
  public int countEdges();
  public BC_Edge2D findEdge(BC_Vertex2D _v1, BC_Vertex2D _v2);
  public BC_LinePairVector getLinePairs();
  public Enumeration getEdges();
  public Enumeration getVertices();
}
```

Apendix B

Results of the Matching Algorithm

Figure B.1 shows results of the Matching Algorithm. The Edges and Corners obtained with the implemented methodology were used as input for the process of matching. Image on the upper part is a photograph of a building of the University of Heidelberg and image on the lower part of the figure is the 3D model view of the same building. The arrows superimposed on the images are the detected matching points. In the figure, matching point 22 is poined out by the long arrow.

Figure B.1: Results of the Matching Algorithm



BIBLIOGRAPHY

[1] Adobe Developers Association. Tiff revision 6.0. Internet Publication. Available from http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf.

[2] J.F. CANNY. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Inteligence*, 8(6):679–698, nov 1986.

[3] D. DOUGLAS and T. PEUCKER. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, 1973.

[4] Steven FEINER, Blair MACINTYRE, Tobias. HOELLERER, and Anthony WEBSTER. A touring machine: prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Proceedings of the International Symposium on Wearable Computing*, 1997.

[5] J.D. FOLEY, A. V. DAM, and S. K. FEINER. *Introduction to Computer Graphics*. Addison-Wesley Publishing Co, 2 edition, 1994.

[6] Anil K. JAIN. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[7] Bolan JIANG and Ulrich NEUMANN. Extendible tracking by line autocalibration. In *Proceedings of IEEE and ACM International Symposium on Augmented Reality*, oct 2001.

[8] Gudrun KLINKER, Oliver CREIGHTON, Allen H. DUTOIT, Rafael KOBYLINSKI, Christoph VILSMEIER, and Bernd BRUEGGE. Augmented maintenance of powerplants: A prototyping case study of a mobile ar system. In *Proceedings of IEEE and ACM International Symposium on Augmented Reality*, page 24, oct 2001.

[9] Joseph NEWMANN, David INGRAM, and Andy HOPPER. Augmented reality in a wide area sentient environment. In *Proceedings of IEEE and ACM International Symposium on Augmented Reality*, oct 2001.

[10] Wolfram Research. Eric Weisstein's world of mathematics. Internet Publication. Available from http://mathworld.wolfram.com/.

[11] Kiyohide SATOH, Mahoro ANABUKI, Hiroyuki YAMAMOTO, and Hideyuki TAMURA. A hybrid registration method for outdoor augmented reality. In *Proceedings of IEEE and ACM International Symposium on Augmented Reality*, oct 2001.

[12] S. M. SMITH and J. M. BRADY. Susan - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[13] Milan SONKA, Vaclav HLAVAC, and Roger BOYLE. *Image Processing, Analysis and Machine Vision*. Brooks/Cole Publishing Co, Pacific Grove, CA, 1999.