Die approbierte Originalversion dieser Dissertation ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DISSERTATION

A vision-based sensing system for sentient building models

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors
der technischen Wissenschaften unter der Leitung von

Univ. Prof. Dipl.-Ing. Dr. techn. Ardeshir Mahdavi

E 259.3
Abteilung für Bauphysik und Bauökologie
Institut für Architekturwissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Architektur und Raumplanung

von

Oğuz İçoğlu

Matrikelnummer: 0326183
Abteilung für Bauphysik und Bauökologie, TU-Wien
Karlsplatz 13 (259.3), A1040 Wien

Wien, Mai 2006.

ACKNOWLEDGEMENT

This dissertation is carried out during my employment at the Department of Building Physics and Building Ecology in Vienna University of Technology. First and foremost, I would like to thank my supervisor, Univ. Prof. Ardeshir Mahdavi, for his generous support and guidance throughout the studies starting with the building physics lectures and ending up in the preparation of this dissertation.

I would also like to thank Ao. Univ. Prof. Markus Vincze for his valuable review and examination.

The research described in this dissertation is undertaken as a part of a project, which is funded by Austrian Science Foundation, project number P15998-N07. The project team, whom I am indebted to, includes Univ. Prof. Ardeshir Mahdavi (director), Ao. Univ. Prof. Georg Suter, Mag. Klaus Brunner, Dipl.-Ing. Bojana Spasojević and Josef Lechleitner. I would like to state my acknowledgement particularly to Georg Suter for his guidance throughout the project, and to Klaus Brunner for his help during the technology review given in chapter 3.

A great deal of thanks goes to my family for their endless support; to my mother who encouraged me through all stages of my academic pursuit, and to my father who put the first appetite of being an engineer in my mind.

Finally, my deepest gratitude is directed to my beloved fiancée, whose patiance and support provided the amount of time and effort needed to make this thesis happen possible.

Oğuz İÇOĞLU

March 2006

ABSTRACT

The quality and cost effectiveness of services in the building industry possess high potential for improvement. One significant approach in bringing out this potential is to conceive buildings as sentient entities that continuously adapt to changes in the environment. A *sentient building* possesses a multi-faceted internal representation of its own context, structure, components, systems, and processes (Mahdavi 2003a, 2003b). This representation enables the self-regulatory determination of such a building's indoor-environmental status in accordance with the needs of its occupants. However, towards the realization of the sentient buildings, already acquired scientific foundations (theories, methods, and tools) must be transformed into a technically mature and industrially promising level.

Specifically, such transformation must occur in three critical areas. Firstly, the representational core of sentient buildings must integrate rather static building component class hierarchies (product models) with process-oriented systems controller hierarchies (process models). Secondly, to achieve real-time building operation support and to avoid bottleneck situations resulting from manual model input and updating activities, the underlying product-process model must possess the capability to autonomously update itself. Finally, given the specific features and challenges of the building systems control domain (e.g. multiple domains/systems, multiple levels of spatial hierarchy, contingencies of outdoor climate and occupancy behavior), proper control semantics (methods, rules, algorithms) must allow for scalable implementation schemes.

To provide a proof of concept for the feasibility of this transformation towards the realization of sentient buildings, a *lighting control system* is developed within the scope of a FWF project (Mahdavi & Suter 2002). The aim of the project is, concisely, to provide and maintain the most desirable lighting conditions in an office space. The research described in this thesis focuses on the second challenging item of the transformation within the project context. The second item implies that the associated representations must be *self-updating*, if they are to be applied effectively in the course of building operation and maintenance activities. This requires capabilities in the areas of contextual and indoor-environmental monitoring.

In the lighting control system, the required monitoring capabilities arise in three major fields. Objects in the space must be identified, their locations must be sensed and occupancy information must be obtained. In addition to these monitoring activities, the prospective solution must comply with the building-specific requirements, where *low-cost, low-maintenance*, and *scalability* are crucial. In this dissertation, the study towards the realization of these capabilities is described. Prior to the implementation of a solution, available technologies are reviewed. With respect to the requirements, vision-based approaches were found to be preferable in terms of being software supported and system customizable. In our efforts for realizing such a solution, a Vision-based Object Location and Occupancy Sensing system (*VIOLAS*) is developed (İçoğlu & Mahdavi 2005).

VIOLAS extracts context information from the environment using *image processing* methods applied to the scenes captured by the cameras. VIOLAS utilizes *network cameras* for this purpose. These new technology cameras are feasible for buildings. They make use of the existing network installation without requiring additional infrastructure. They act like regular network devices, and convey camera images with standard Internet protocols. Through the same communication channel, they also enable the control of third-party devices like pan-tilt units that effectively increase the monitoring ranges.

In addition to its primary objectives, the software implementation of VIOLAS must fulfill the aforementioned building-specific requirements. Towards this end, the research proposes a common model that integrates hardware and software whereby the components are tied together via Internet. Network cameras constitute the hardware part of the system, and fit in this structure by conveying video images like as distributed network devices on Internet. Image Processing Units (IPUs) form the distributed software components. They are the programs that perform vision-based sensing and extract the context information by applying optimized image-processing and computer-vision methods on the images captured from the cameras. IPUs, implemented on different computers scattered across the facility, convey the context information to a central Application Server, where the parallel incoming results are combined, displayed to the operator, and concurrently conveyed to the lighting control system. In addition to enabling scalability and incremental growth, the distributed structure of the model enhances performance resulting from the parallel operations.

Additional function of the Application Server is to control the status of the components and dynamically assign active network cameras to active IPUs in such a manner that the workload is constantly balanced within the system. This arrangement provides a kind of self-organizing capability, and minimizes operator overhead. The resulting flexible and adaptive structure is highly suited to the requirements of control applications for sentient buildings.

INHALTSANGABE

Ein visuell-basiertes Sensorsystem für sentiente Gebäudemodelle

Die Qualität und Kosteneffektivität von Dienstleistungen in der Bauindustrie haben ein grosses Verbesserungspotenzial. Eine mögliche Strategie zur Umsetzung dieses Potenzials besteht darin, Gebäude als sentiente Einheiten zu verstehen, welche sich kontinuierlich an Veränderungen in der Umgebung anpassen. Um *sentiente Gebäude* (sentient buildings) zu realisieren, müssen umfassende Modelle der physischen und Verhaltensaspekte von Gebäuden über den Lebenszyklus generiert werden. Eine händische Wartung führt jedoch zu Engpässen, welche das Erreichen von Echtzeit-Unterstützung des Gebäudebetriebs verunmöglichen (Mahdavi 2003a, 2003b). Deshalb sollte ein Modell die Fähigkeit haben, sich selber zu aktualisieren. Das bedingt einen vielseitigen Sensormechanismus, welcher Echtzeit-Information über den Zustand eines Gebäudes liefert (Mahdavi 2001a, 2001b, Pal & Mahdavi 1999).

Deshalb hat sich die Abteilung für Bauphysik und Bauökologie an der Technischen Universität Wien darauf konzentriert, einen Prototyp eines *selbstaktualisierenden* Raummodells für den Gebäudebetrieb zu entwickeln. Als Demonstrationsprototyp wurde ein *Lichtregelungssystem* umgesetzt im Rahmen eines FWF (Fonds zur Förderung der wissenschaftlichen Forschung) Projekts (Mahdavi & Suter 2002). Das Ziel des Projekts bestand darin, die bestmöglichen Lichtverhältnisse in einem Büroraum zu erzeugen. Für die Generation eines umfassenden, selbst-aktualisierenden Modells ist es notwendig, dass der Systemprototyp Objekte identifizieren, ihre Position bestimmen, sowie die Anwesenheit von Personen im Raum (occupancy) erkennen kann. Zusätzlich zu diesen Erfassungsaktivitäten soll eine mögliche Lösung gebäudespezifische Anforderungen erfüllen, d.h. es soll kostengünstig, wartungsarm und skalierbar sein. In dieser Dissertation wird die Umsetzung von diesen Anforderungen beschrieben. Vor der Beschreibung der Umsetzung werden bestehende Technologien beurteilt. Mit Blick auf die Anforderungen wurden computer-vision-basierte Lösungsansätze als vielversprechend eingeschätzt bezüglich der vorhandenen Software und Anpassbarkeit. Im Rahmen unserer Arbeit mit dem Ziel der Realisierung einer

solchen Lösung wurde ein Vision-based Object Location and Occupancy Sensing system (VIOLAS) entwickelt (İçoğlu & Mahdavi 2005).

VIOLAS funktioniert nach dem bekannten Barcode-Leseprinzip und setzt eine Kombination von visuellen *Markierungen* (Tags) und *Videokameras* ein. Anders als bei einem Barcode ermöglicht die Struktur der Tags zusätzlich zur Identifikationsnummer die Extraktion von Positionsinformation. Durch *bildverarbeitende* Methoden erfasst das System in Echtzeit Identifikation und Position eines markierten Objekts. Durch den Einsatz von Videokameras kann es ebenfalls die Anwesenheit von Personen basierend auf Bewegungsanalyse erkennen.

Zusätzlich zu diesen primären Zielen soll die Softwareumsetzung von VIOLAS die erwähnten gebäudespezifischen Anforderungen erfüllen. Deshalb wird in dieser Dissertation eine verteilte Systemarchitektur vorgeschlagen, in welcher die Hardware (Kameras) und die Software (bildverarbeitende Programme) über das Internet miteinander verküpft werden. Neben der Skalierbarkeit und inkrementellen Erweiterbarkeit verbessert die verteilte Systemarchitektur die Performance durch parallele Operationen. Die resultierende flexible und anpassbare Struktur ist sehr geeignet für die Anforderungen von Steuerungsandwendungen in sentienten Gebäuden.

ÖZET

Duyarlı Bina Modelleri için Görüntü Tabanlı Bir Algılama Sistemi

Kontrol ve otomasyon sistemlerinin; binalardaki işletme ve bakım maliyetlerini düşürerek, çevresel şartları iyileştirerek ya da konfor ve güvenlik seviyelerini yükselterek yapı performansını arttırması beklenmektedir. Ancak, bu amacın gerçekleştirilmesine yönelik veri toplama ve izleme faaliyetleri mevcut bina otomasyon sistemlerinde sınırlıdır: genellikle bu işlemler sadece asansör ve birkaç benzeri bina ekipmanını kapsar. Binaların yaşam döngüsü boyunca yürütülen kapsamlı bir durum izlemesine yönelik sistematik ve ölçeklenebilir yaklaşımlarda hala eksiklikler bulunmaktadır. Daha yüksek seviyede bir başarım elde etmek için önemli yaklaşımlardan biri, binaları duyarlı birer varlık olarak tanımlamak, ve onları sürekli değişen çevresel ve beşeri şartlara cevap verecek şekilde tasarlamaktır (Mahdavi 2003a, 2003b). *Duyarlı binalar*da (sentient buildings) düzenli olarak toplanması gereken veriler sadece asansörlere değil; iç yüzeylere, mobilyalara, kapılara, pencerelere, içerde bulunan insanlara ve benzer diğer statik veya dinamik ögelere ait olmalıdır.

Ancak duyarlı binaların gerçeklenmesi henüz tartışmalı bir konudur. Böyle bir gerçekleme, herşeyden önce binaların fiziksel ve (insan etkileşimi sonucu oluşan) sosyolojik özelliklerine ait kapsamlı modellerinin üretilmesine ihtiyaç duyar. Bu modellerin manuel olarak üretilmesi bir darboğaz yaratmakta ve bina otomasyon sistemlerinin *gerçek-zamanlı* desteklenmesini önlemektedir. Bundan dolayı gerçek-zamanlı bina durum bilgisini üretebilecek çok yönlü algılama sistemlerine ihtiyaç duyulmaktadır (Mahdavi 2001a, 2001b, Pal & Mahdavi 1999).

Viyana Teknik Üniversitesi, Yapı Fiziği ve Bina Ekolojisi bölümü, bina operasyonları için prototip bir *kendini güncelleyen* bina modelinin geliştirilmesi üzerinde çalışmaktadır (Mahdavi 2001b). Konseptin somutlaştırılması için, bir FWF (Avusturya Bilim Kurulu) projesi kapsamında, örnek bir *aydınlatma kontrol sistemi* geliştirilmektedir (Mahdavi & Suter 2002). Bu sistemin amacı, kısaca, çalışma ofisini temsil eden bir test alanında en uygun aydınlatma koşullarını sağlamaktır. Aydınlatma kontrolü için gerekli algılama işlemleri üç temel konu üzerinde

yoğunlaşmıştır: alan içindeki nesneler tanınmalı, yerleri tespit edilmeli ve alandaki doluluk (occupancy) bilgisi sezilmelidir. Algılama özelliklerinin yanında, olası bir çözüm binaya özgü gereksinimlere de uyum sağlamalıdır. Dolayısıyla az bakım gerektirmeli, ölçeklenebilir ve düşük maliyetli olmalıdır. Bu tezde, sisteme bu yeteneklerin kazandırılmasına yönelik çalışmalar anlatılmaktadır. Herhangi bir çözümün uyarlanmasından önce mevcut teknolojiler incelenmiştir. Bu doğrultuda, gerek yazılım destekli, gerekse konfigüre edilebilir olmasından ötürü görsel tabanlı yaklaşımlar üzerinde odaklanılmıştır. Böyle bir çözümün gerçeklenmesi çabaları sonucunda, görüntü tabanlı bir algılama sistemi olan VIOLAS (vision-based object location and occupancy sensing system) geliştirilmiştir (İçoğlu & Mahdavi 2005).

VIOLAS, "barkod okuyucu" prensibine benzer bir şekilde çalışmaktadır. Sistem, nesneler üzerine yapıştırılan görsel *etiketler* (tag) ve onları tarayan *kameralar*dan faydalanır. Etiketlerdeki özel yapı sayesinde, *görüntü işleme* (image-processing) metodları kullanılarak, nesneler tanınır ve yerleri belirlenir. Aynı zamanda, kameralar aracılığıyla, alandaki doluluk bilgisi, görüntülerdeki hareketin algılanmasıyla ortaya çıkarılır.

VIOLAS'ın yazılım uyarlaması, ana amacın gerçekleştirilmesine ek olarak, yukarıda bahsedilen binaya özgü gereksinimleri de karşılamalıdır. Bu amaçla, tez çalışmasında, sistemin donanım (kameralar) ve yazılım (görüntü-işleme programları) bileşenlerini Internet üzerinden bağlayarak entegre eden dağıtık bir mimari sunulmuştur. Bu mimari, *ölçeklenebilirlik* ve *aşamalı büyüme*ye ek olarak, paralel işlemler sayesinde yüksek bir başarım da sağlamaktadır. VIOLAS, döner-taban ünitesine (pan-tilt) bağlı bir kamera aracılığı ile 50 m^2'lik bir alanı tarayabilmektedir. Bu tarama ile alan içindeki doluluk algılanmakta, nesneler tanımlanmakta ve yerleri ortalama olarak 0.18 cm pozisyon ve 4.2 derece oryantasyon hatası ile tespit edilmektedir. Gerek elde edilen algılama başarımı, gerekse meydana gelen esnek ve adaptif yapı, görüntü tabanlı uygulamaların duyarlı binaların gereksinimlerine uygun olduğunu göstermektedir.

List of Contents

ABSTRACT v
INHALTSANGABE viii
ÖZET x
1 INTRODUCTION 21
2 REQUIREMENTS FOR A SELF-UPDATING MODEL 26
3 TECHNOLOGY REVIEW 27
  3.1 Object Identification and Location Sensing 27
    3.1.1 Electromagnetic and Radio Frequency 27
    3.1.2 Ultrasound 28
    3.1.3 Optical / Vision-Based 28
    3.1.4 Technology Evaluation 30
  3.2 Occupancy Sensing 31
4 SENSOR EVALUATION FOR VISON-BASED TECHNOLOGIES 32
  4.1 Web Cameras 32
  4.2 Digital Photo Cameras 33
  4.3 CCTV Cameras 33
  4.4 Network Cameras 34
  4.5 Technology Evaluation 35
  4.6 Pan-Tilt Units 37
    4.6.1 Communication 37
      4.6.1.1 Physical Layer 37
      4.6.1.2 Data Link Layer 38
    4.6.2 Position Feedback 39
5 OBJECT IDENTIFICATION AND LOCATION SENSING 40
  5.1 Location Sensing 40
  5.2 Object Identification 41
  5.3 Original Method 42
    5.3.1 Target Recognition 43
      5.3.1.1 Binarization 43
      5.3.1.2 Edge Detection 43
      5.3.1.3 Edge Filtering 44
      5.3.1.4 Ellipse Fitting 45
      5.3.1.5 Concentricity Test 45
      5.3.1.6 Code Deciphering 46
    5.3.2 Pose Extraction 48
      5.3.2.1 Extracting the Plane Orientation 48

5.3.2.2 Extracting the Tag Position 54
5.3.2.3 Extracting the Rotation Angle 56
5.3.2.4 Breaking the Ambiguity 60
6 ENHANCED LOCATION SENSING 65
6.1 Adaptive Sharpening 66
6.2 Edge-Adaptive Zooming 68
6.3 Results of Enhanced Location Sensing 74
7 OCCUPANCY SENSING 77
7.1 Homomorphic Filtering 78
7.2 Illumination-invariant Change Detection 81
8 VIOLAS CONCEPTUAL DESIGN 83
8.1 Hardware Interface 83
8.2 Sensing Core 85
8.3 Coordinate Transformation 85
8.4 Data Fusion 89
8.4.1 Tag-Level Fusion 89
8.4.2 Object-Level Fusion 90
8.5 Communication Interface 91
8.6 User Interface 91
9 IMPLEMENTATION 96
9.1 Image Processing Unit 97
9.1.1 IPU Control 98
9.1.2 Image Acquisition 99
9.1.3 Image Processing 106
9.2 Application Server 107
9.2.1 Resource Management 108
9.2.2 Data Integration 110
9.3 User Interface Server 111
9.3.1 Common Development Scheme for CGI Applications 111
9.3.2 Implementation in VIOLAS 113
9.4 Database Server 116
9.4.1 Remote Data Module 116
9.4.2 Communication with the Database Server 118
9.4.3 Implementation in VIOLAS 119
10 AUXILIARY PROGRAMS IN VIOLAS 121
10.1 Camera Calibration 121
10.2 Image-Processing Tester 124
11 A DEMONSTRATIVE TEST 126

11.1 Organization of the Test Platform 126

11.2 VIOLAS in Operation 131

12 CONCLUSIONS 139

REFERENCES 141

Appendix A 143

Appendix B 145

Appendix C 148

Appendix D 153

Appendix E 156

CURRICULUM VITAE 159

List of Figures

Figure 1. Flow diagram of the lighting control system. 22

Figure 2. Generation of self-updating space model........ 23

Figure 3. Layered view of the system architecture (Brunner 2006)........ 25

Figure 4. Web camera connection scheme........ 33

Figure 5. CCTV camera connection scheme........ 34

Figure 6. Network camera connection scheme........ 35

Figure 7. Pan-Tilt unit installation scheme. 38

Figure 8. “Pose from circle” algorithm. 41

Figure 9. Tag structure is illustrated with a sample tag........ 42

Figure 10. Original method used in TRIP system. 42

Figure 11. (a) Original image captured from the network camera. (b) Result of the binarization after being applied to the input image........ 43

Figure 12. (a) Connected edge points for a prospective ellipse. (b) Connected edge points that will be eliminated after the filtering process........ 44

Figure 13. (a) Result of edge detection. (b) Result of edge filtering. 44

Figure 14. Geometric parameters of an ellipse........ 45

Figure 15. A tag possesses at least two concentric ellipses in the edge-detected image. With the addition of even-parity bit, the number of concentric ellipses may be three. ... 46

Figure 16. Code ring is scanned, first for designating the start bits, then for extracting the ID number. The dot-trio on the code ring represents the pixel samples for each sector.47

Figure 17. The geometry of pinhole camera model. 49

Figure 18. Camera intrinsic parameters. 50

Figure 19. 3D rotation of the camera frame to the new (X',Y',Z') frame, after R_1 is applied.53

Figure 20. 3D rotation of the camera frame to the new (X'',Y'',Z'') frame after R_2 is applied.55

Figure 21. The unit vectors on the target plane........ 57

Figure 22. Circle projection ambiguity. 61

Figure 23. Modified tag structure designed to break the ambiguity. Benchmark points, $P_t = (e_{t_1}, e_{t_2}, e_{t_3}, e_{t_4})$, are depicted on the target-centered coordinate system. 63

Figure 24. Extracting the location of projected benchmark points, $P = (e_1, e_2, e_3, e_4)$, in the binary image. 63

Figure 25. Enhanced object identification and location sensing. 66

Figure 26. The results of low-pass filtering. 67

Figure 27. (a) Unsharp-masked image. The sharpening process also augments the power of the noise. (b) Adaptively sharpened image. 69

Figure 28. The expansion of the original image, $f(n_1,n_2)$, to the zoomed image, $z(n_1,n_2)$. ... 70
Figure 29. Results obtained with different zooming algorithms. 72
Figure 30. Results obtained for a sample scene in the test-bed.. 75
Figure 31. An example for the common motion detection problems.................................. 78
Figure 32. Decomposition of the images into illumination and reflectance components. 80
Figure 33. Homomorphic filter for multiplied signals. 81
Figure 34. Change detection in sequential reflectance components...................................... 81
Figure 35. The results of the change detector.. 82
Figure 36. Hardware interface.. 84
Figure 37. Sensing core.. 85
Figure 38. Coordinate transformation. 86
Figure 39. Parameters considered in the coordinate transformations.................................. 86
Figure 40. Data fusion.. 89
Figure 41. Tag-level fusion. 90
Figure 42. Object-level fusion.. 90
Figure 43. Communication interface.. 91
Figure 44. User interface.. 92
Figure 45. Object and tag hierarchy in the object inventory. 93
Figure 46. Object and tag reference frames. 94
Figure 47. Distributed structure of VIOLAS... 97
Figure 48. Structure of the image processing unit... 98
Figure 49. Serial D9 port of an IQeye3 camera.. 101
Figure 50. (a) Visca DCP-24 controller (GNT 2006). (b) RS232 connection layout between the camera and the controller. 102
Figure 51. Structure of the application server. 109
Figure 52. Common gateway interface. 111
Figure 53. Structure of the user interface server. 114
Figure 54. User table given as an example to show the structure of a XML file. 117
Figure 55. Structure of the database server. 118
Figure 56. Calibration image.. 124
Figure 57. Image-processing tester. 125
Figure 58. Database server program.. 126
Figure 59. User interface server, and the camera employed in the test............................. 127
Figure 60. (a) System parameters setup. (b) Camera parameters setup. 128
Figure 61. Object setup. 129
Figure 62. A snapshot of the object list.. 129
Figure 63. Some of the tagged objects in the test-bed.. 130

Figure 64. 2D sketch of the test-bed.. 130
Figure 65. Application server.. 132
Figure 66. Image processing unit 133
Figure 67. Graphical representation of the test-bed generated by the user interface server. 136
Figure 68. Converting RGB images to gray scale.. 144
Figure 69. Translation in 3D. 149
Figure 70. Rotation in 3D.. 150
Figure 71. Extracting the incidence angle.. 153
Figure 72. A simple potentiometer.. 156

List of Tables

Table 1. Qualitative overview of location-sensing technologies... 30
Table 2. Qualitative comparison of camera technologies. 36
Table 3. Ground-truth data of the objects in the test-bed. 131
Table 4. Sensed location values of the objects in the test-bed recorded by VIOLAS. 134
Table 5. Position and orientation errors of the objects with respective camera distances and incidence angles. 134
Table 6. The average and maximum position and orientation errors for different camera-tag distance bins... 135
Table 7. The average and maximum position and orientation errors for different incidence angle bins. 135
Table 8. Identification results of the test implemented without enhancement methods....... 137
Table 9. Location values of the identified objects sensed without enhancement methods. . 138

Glossary

2D	Two dimensional
3D	Three dimensional
A/D	Analogue to Digital
API	Application Programming Interface
CAM	Camera
CCD	Charge Coupled Device
CCTV	Close Circuit Television
COM	Component Object Model
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTS/RTS	Clear-to-Send / Ready-to-Send
DCOM	Distributed Component Object Model
FOV	Field of View
FTP	File Transfer Protocol
FWF	Fonds zur Förderung der wissenschaftlichen Forschung
GUID	Global Unique Identifier
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HW	Hardware
ID	Identity, Identification, Identifier
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPU	Image Processing Unit

JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LED	Light Emitting Diode
MS	Microsoft
NETCAM	Network Camera
NTSC	National Television Systems Committee
P/T	Pan-Tilt Unit
PC	Personal Computer
PCI	Peripheral Component Interconnect
PIR	Passive Infrared
RFID	Radio Frequency Identification
RGB	Red-Green-Blue
SNR	Signal to Noise Ratio
STDOUT	Standard Output
SW	Software
TCP/IP	Transfer Control Protocol / Internet Protocol
TRIP	Target Recognition using Image Processing
USB	Universal Serial Bus
VIOLAS	Vision-based Object Location and Occupancy Sensing System
WEBCAM	Web Camera
XML	Extensible Markup Language
XON	Transmit on
XOFF	Transmit off

1 INTRODUCTION

Building automation is expected to improve building performance by reducing the operation and maintenance costs of buildings (e.g. for heating, cooling, and lighting), improving environmental performance, augmenting human comfort, and providing higher safety levels. However, data collection and monitoring activities in current building automation systems are rather limited: the focus is mostly on service systems such as elevators and office equipment. There is a lack of systematic and scalable approaches to comprehensive facility state monitoring throughout buildings' life cycle. One significant approach in achieving a higher level of building automation technology is to assume that buildings are sentient entities, constantly changing in response to changes in the environment and occupancy (Mahdavi 2003a, 2003b). In *sentient buildings*, collected data must cover not only the state of systems such as elevators, but also the state of room enclosure surfaces, furniture, operable windows, occupants, and other static or dynamically changing building entities.

However, the realization of a sentient building is still a controversial subject. Towards such a realization, the role of the *simulation-based control* mechanisms is important (Mahdavi 2001b). These mechanisms control specific building operations (like heating, lighting or security) through the analysis of the building model with an embedded simulator component. As a result of this analysis, the current conditions can be observed, and consequently, the new prospective conditions that will take place with the actuation of the control devices can be foreseen. These features enable the development of an effective control system. However, such a development primarily entails for generating comprehensive and *self-updating* models of the physical and behavioral aspects of facilities over their life cycle (Mahdavi 2001a, 2001b, 2003a, 2003b, Pal & Mahdavi 1999). Thereby, Vienna University of Technology, Department of Building Physics and Building Ecology has focused on developing a prototype sensor-supported self-updating building model for simulation-based building operation support (Mahdavi 2001b). To deliver a proof of the concept for feasibility, a *lighting control system* is designed and implemented within the scope of a FWF project (Mahdavi & Suter 2002).

The aim of the system is, concisely, to provide and maintain the most convenient lighting conditions in a test space intended to represent an office room. Figure 1 briefly demonstrates the flow in the system. First, a data collection unit collects the *environmental data.* A weather station located on the tower of the university building is used for sensing the outdoor conditions. Weather station is an integrated unit that tracks the values of temperature, humidity, global irradiance and global illuminance. Within the context of environmental conditions, data collection unit also tracks the status of the *control devices*, i.e., positions of the remotely controllable window blinds and the illumination levels of the room uplights.

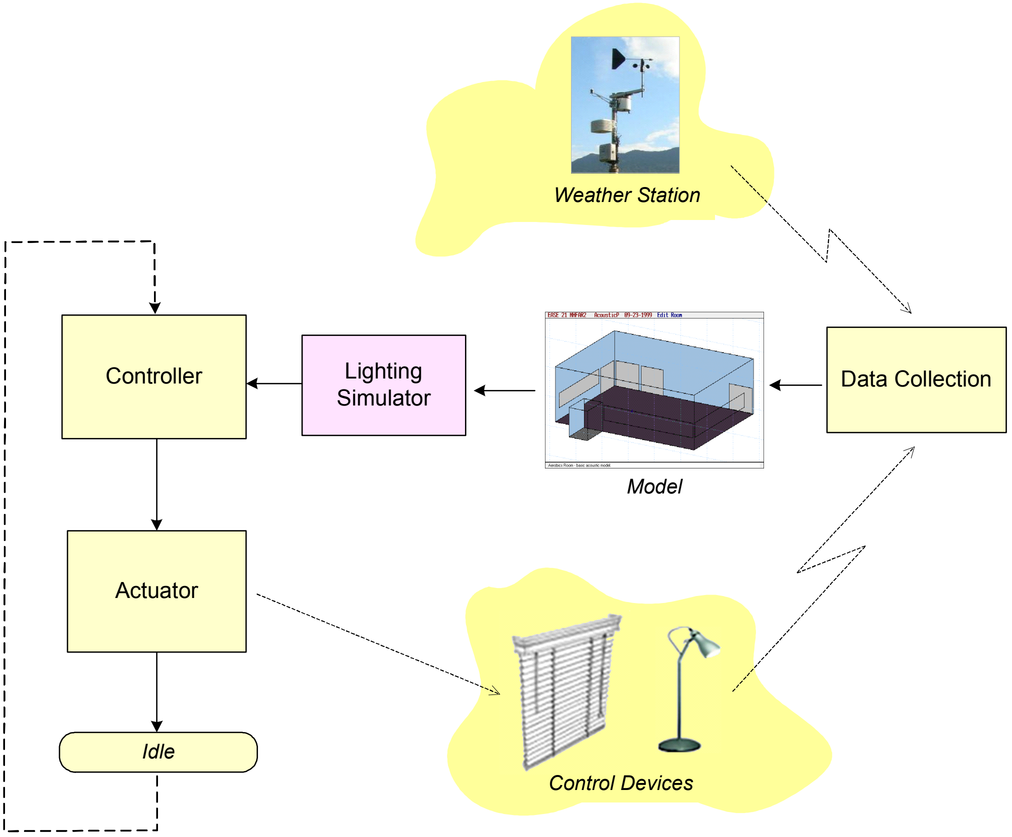


Figure 1. Flow diagram of the lighting control system.

As stated above, unlike the feedback control systems that cannot effectively support sentient buildings, a simulation-based control approach is implemented in the project. Based on the collected environmental data, the model of the subject space is generated. A controller unit analyzes the lighting status of the space through a lighting simulator program applied on the generated model. Based on the returning results, the controller runs a decision making process, and consequently provides the solution that generates the most appropriate lighting. For the realization of the outcome, the controller subsequently applies this solution to the control devices through an actuator unit. This flow is reiterated in order to maintain the most convenient lighting against changing conditions in the environment.

As mentioned above, the simulator application requires a precise model of the associated space, and this model must be constantly updated with respect to the changes in the building. The data collection unit, however, is not sufficient to provide such an updated model of the space. Therefore, the generation of the self-updating space model is undertaken with a model generator unit supported by a digitally derived sky luminance mapping and an object sensing system (Figure 2).

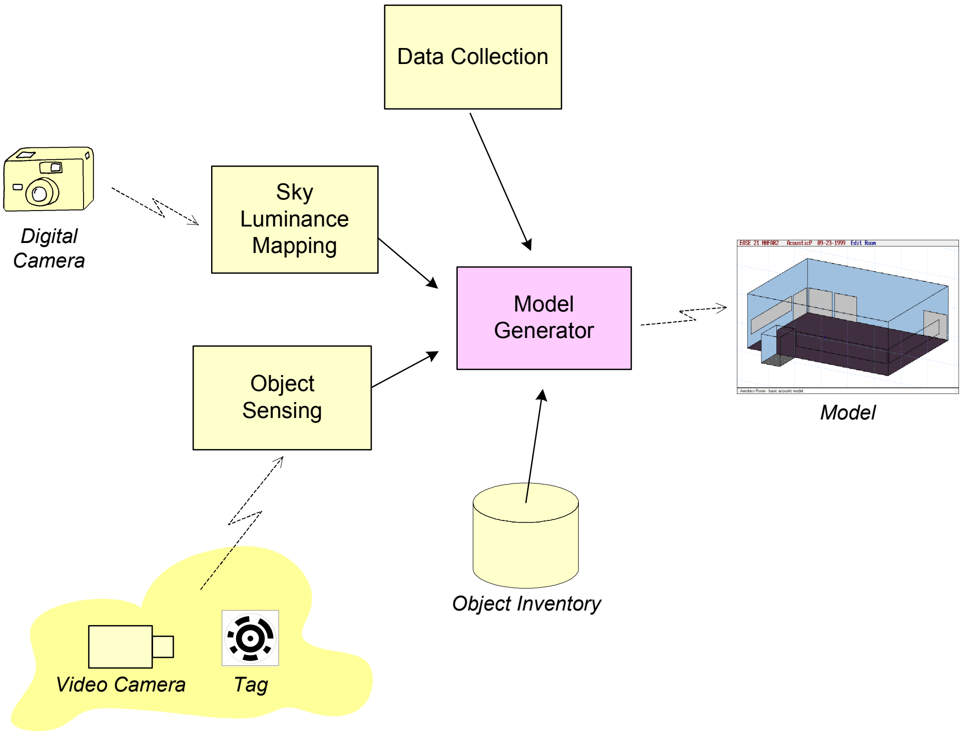


Figure 2. Generation of self-updating space model.

Towards the realization of the first supporting unit, i.e., sky luminance mapping, a digital camera equipped with a fisheye converter and pointing toward the zenith is placed next to the weather station on the tower of Vienna University of Technology. Digital images of the sky are continuously taken, analyzed and calibrated to construct the sky model (sky luminance distribution pattern) for the simulation application (Spasojević & Mahdavi 2005).

This thesis describes the implementation of the second supporting unit of the model generator: object sensing system. As depicted in Figure 2, the information (definition and dimensions) about the objects in the test space is stored in an object inventory. Object sensing, briefly, identifies these objects, extracts their location, and detects the occupancies in the space. Thus, in addition to the environmental data and sky luminance map, the model generator combines these sensed object information with the known object data (stored in the inventory), and constructs a comprehensive and up-to-date space model.

Before the implementation of an object sensing solution, available technologies are reviewed, and vision-based methods are found to be preferable. Towards this end, a vision-based object location and occupancy sensing system, VIOLAS, is implemented. This system concisely works like the well-known “barcode reader” principle, using a combination of visual markers (tags) and video cameras. Unlike the barcode, the special structure of the tags enables the extraction of location information together with a coded identity number. Thus, with utilizing image-processing methods, the system obtains in real-time the identification and location of the tagged objects; and by exploiting the video cameras, also detects the occupancy based on the motion salience in the space. The details of the design and implementation of the system are described throughout the thesis.

For the project, the implementation of the lighting control system requires the concurrent execution of the distinct units described above. The lighting controller, model generator and simulator must run in parallel and in accordance with each other. In addition to these units, the object inventory is managed, the generated model is visualized and the system performance is continuously monitored within the system. The data flow and hardware integration (handling the data collection from sensors, and driving the actuators) must also be established in a way to enable the harmony and consistency.

Towards this end, a system architecture is designed, where a layered view is shown in Figure 3. At the lowest level, the physical layer comprises the building and its environment as such. Observing and controlling the state of these entities are handled with sensors and actuators, as shown on the next system level. Sensors can be hardware devices like the weather station, or hardware-software integrated units like sky mapping and object sensing. Actuators are utilized to drive blinds (motorized shading), and luminaries (dimmers). To gather and distribute data between these devices and computer programs using them, a communications layer is needed. This may include combinations of various specialized (LonWorks, BACnet) and general-purpose (Ethernet, TCP/IP, MQ) communication technologies.

The key element of the system is in the next level, the model service layer. Instead of letting applications –such as lighting controller or inventory management– communicate directly with the communications layer, it offers an additional level of abstraction that isolates applications from the details of communications and sensor hardware. The model service represents the current state of the building and its environment in the form of a live building product model. Applications are thus freed from dealing with specific communication, sensor and actuator systems, but instead, communicate with objects whose properties, methods and relations with other objects provide a high-level interface to the physical world. The details of the software architecture and implementation of the lighting control system are described in Brunner 2006.

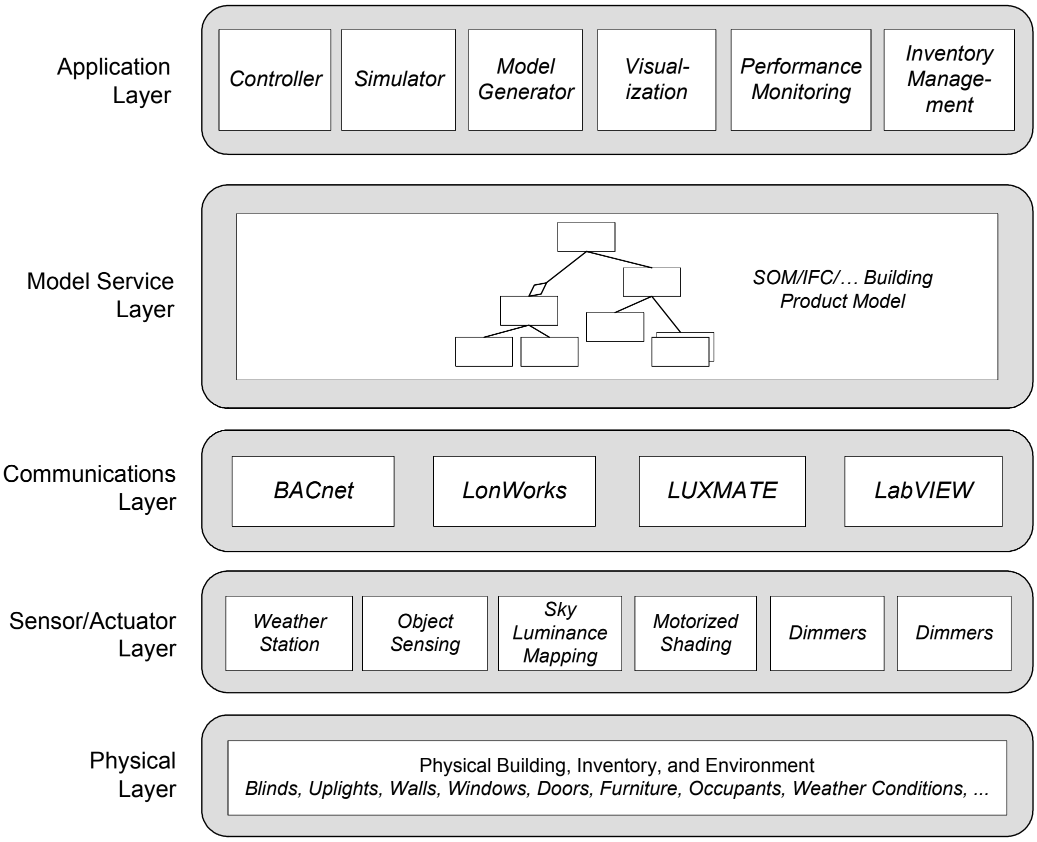


Figure 3. Layered view of the system architecture (Brunner 2006).

The present thesis, hereafter, describes the requirements for generation of self-updating models within the object-sensing scope in chapter 2, and review of the available sensing technologies towards the meeting of these requirements in chapter 3. Chapter 4 explains the feasibility of vision-based solutions and the evaluation of different type of visual sensors. Chapter 5 and 6 describes the details concerning object identification and location sensing. The methods used in occupancy detection are given in chapter 7. Chapter 8, 9 and 10 describe the design and implementation phases of *VIOLAS*, the vision-based object location and occupancy sensing system, based on the outlines and methods given in previous chapters. Chapter 11 defines the layout of a demonstrative test implemented for the performance evaluation of VIOLAS. The conclusions derived from the research are, finally, given in chapter 12.

2 REQUIREMENTS FOR A SELF-UPDATING MODEL

The generation of a space model is possible with the known dimensions of the objects in the space. However, enabling a self-updating capability requires the continuous collection of additional prerequisite information. Firstly, the objects in the space must be identified, and their locations must be sensed. The location information must be comprised of both *position* and *orientation* data for a precise model.

Furthermore, the solution intended for model generation should also comply with the requirements specific to building environments. Therefore, the prospective solution should require minimum maintenance, and be scalable to adapt itself to changes in a facility. Other important evaluation criteria are *accuracy* (relatively small systematic variation in measurements), *unobtrusiveness* (minimal installation and maintenance necessary, no inconvenience or health hazards for occupants), *cost* (per square meter, per object), *scalability* (dozens to hundreds of items per room, thousands per building), and *reliability* (accurate location information for long time intervals, under adverse conditions, in cluttered, changing indoor environments).

In addition to these major requirements, the lighting simulator program utilized within the project particularly needs to be aware of the occupancy information in the space. In the following chapter, the review of available technologies is given from these requirements' perspective.

3 TECHNOLOGY REVIEW

Prior to the implementation of a solution, the available technologies are reviewed from the building automation perspective. First, the object identification and location sensing systems are examined in view of primary requirements. Then, the possible occupancy detection methods are investigated.

3.1 Object Identification and Location Sensing

Most currently available location systems use *tag*s, small items affixed to the actual objects to be tracked. Location information is obtained by signal exchange between these tags and a sensor infrastructure *(sensor*s, *reader*s). Even more so than in other ubiquitous computing applications, building model applications call for rather small, long-lived tags that require no batteries or any other maintenance. Moreover, systems based on devices that obtain or calculate position information internally (called localized location computation) are not meaningful in building model applications, unless the location information is fed back to the overall system.

3.1.1 Electromagnetic and Radio Frequency

These include technologies based on the measurement of electromagnetic or radio frequency signals' field strengths, distortion, time-of-flight or frequency. Their main advantage is that they can usually operate through obstacles, without requiring a line-of-sight between tags and sensors. However, the presence of metal objects and thick walls can have significant influence on operating range and location accuracy. Electromagnetic systems (such as Polhemus FASTRAK (Polhemus 2004)) achieve very high accuracy and precision (mm range), but can only operate in relatively small, closed environments. They are also very expensive and sensitive to metallic objects, and often require cable connections between tags and sensors. A number of research prototypes and products are available for using existing RF infrastructure (such as Bluetooth or 802.11 networks) to calculate position information, for instance Ekahau Positioning Engine (Ekahau 2004). All these products are based on localized location computation, making them currently less suitable for building model applications.

Systems based on RFID (radio frequency identification) tags are particularly interesting, but currently no mature commercial system with acceptable accuracy is available. SpotON (Hightower et al. 2000), a research project, claims accuracy in the one meter range using off-the-shelf active tags. However, the available data can only support an accuracy of three meters. LANDMARC (Ni et al. 2003), a similar research system, aims to improve accuracy by installing a grid of reference tags throughout the area of interest. The location accuracy of the system is approximately the same as the granularity of the grid, which means that to achieve one meter accuracy, active (battery-powered) reference tags have to be placed in a grid with a unit length of one meter.

The commercial product PinPoint (Werb & Lanzl 1998) uses active tags, communicating with transponders in the microwave frequency range. In indoor environments, the system requires considerable installation overhead. The system can achieve a resolution of 3 meters at best, and for further resolutions, is limited to generating only the existence information. A competing product, WhereNet (WhereNet 2004), achieves similar performance.

3.1.2 Ultrasound

Ultrasound-based systems typically consist of battery-powered tags or “badges” and a set of transponder stations communicating with them; position information is obtained by measuring time-of-flight of acoustic signals. Research prototypes include Bat (Ward 1998) and Cricket (Priyantha 2000). A commercial product is available from Sonitor Technologies (Sonitor 2004). Sonitor’s system can operate in two modes: room-based (containment) and 3D. In 3D mode, it requires eight receiver devices to be fixed in every room; for positioning, four of these must be in direct line-of-sight to the tag. A maximum of four tags can be tracked per room, with a claimed resolution of 2–3 centimeters. Although this resolution is sufficient for building model purposes, the poor scalability and the strict line-of-sight restriction make this technology impractical for use in real-world applications.

3.1.3 Optical / Vision-Based

Sensors stimulated by optical attributes are also used for location awareness. The main advantage of vision-based location awareness systems is that they do not require high-cost tags that need continuous maintenance; sometimes they do not require a tag at all. The non-tagged technologies utilize visual attributes of the objects, and are based on computer vision methods that exploit the relationship of the brightness at a point (x,y) in the image with the depth (z) information of the surface under certain lighting or camera conditions. A prototypical realization of the methods mentioned above is EasyLiving (Krumm et al. 2000), where the location of the occupants inside a room is detected together with their personal identification. EasyLiving successfully combines multi-sensor data. However, the system is limited to person location and identification. Moreover, the identification is based on color features only, which makes it difficult to distinguish people wearing clothes with similar colors.

Another approach is the use of laser sensors to determine depth information. These systems comprise a transmitter unit where a laser beam is generated and emitted, and a receiver unit where the reflecting laser beam is captured. Depth information is extracted from the travel time of the laser beam. A more complex version of these sensors is the laser camera where the above process takes place for each picture element and consequently, forms a range map of the scene. CityScanner (GeoData 2004) performs a combination of this technology with digital cameras. This system is, however, designed primarily for outdoor use and generally too slow for building model applications.

The laser sensor systems mentioned above do not directly accomplish object identification, but require additional processing methods to recognize the objects from 3D scenery. Other vision-based systems use passive tags and utilize their visual attributes rather than the visual attributes of the objects themselves. These technologies usually work in variations of the well-known ”barcode reader” principle, scanning scenes for distinctive optical markers. Just as other optical active-tagged or non-tagged systems, they have the disadvantage of requiring line-of-sight between objects and sensors (cameras, scanners). However, when compared with non-tagged solutions, the main benefit of using tags is that they can be coded with an ID number that makes the identification of the individual objects possible.

One example is Phoenix Technologies’ Visualeyez system that uses LED (light emitting diode) markers fixed to the tracked object. It achieves sub-millimeter accuracy, and is able to track thousands of tags simultaneously. Its main disadvantages are high cost and the power consumption of the LED tags, limiting its usefulness for realistic office situations. There are also systems using simpler visual tags rather than power consuming LEDs. Shape features are used for identifying most of these tags, where the area and number of holes, lines and plain regions in the tag determine its main shape features. A Mitsubishi Electric Research Laboratories research prototype (Mitsubishi 2004) focuses on identification of trademark logos by using the shape definitions. In addition to shape-based methods, the contours of the visual tag are extracted and length and curvature features are also processed for identification. TRIP (Target Recognition using Image Processing (Lòpez et al. 2002)) is a particular example of the contour-based method. It works with circular black-and-white TRIPcode tags that can be generated with an ordinary laser printer. The contours of the circular tags are taken from camera images, and then used to calculate the location and identification with a number of image processing techniques.

3.1.4 Technology Evaluation

Table 1 provides a qualitative comparison of the main location-sensing technologies considered above.

Table 1. Qualitative overview of location-sensing technologies.

	RFID	ultrasound	vision (tagged)
Precision	3 m	Few cm	Few cm
Obtrusiveness	Medium	Medium	High
Cost	High	High	Low
Scalibility	High	Low	High
Reliability	High	Low	Low
Identification	Yes	Yes	Yes

In short, the findings are:

– RF-based tagged systems—while promising due to their high scalability and reliability— are not accurate enough and require considerable infrastructure, as well as fairly expensive tags.

– Ultrasound systems provide sufficient accuracy, but have serious shortcomings in scalability and reliability, as only a few tags per room can be tracked, and accurate positioning requires clear line-of-sight between tags and receivers. Cost, both in terms of infrastructure and tags, is comparable to RFID solutions.

– Tagged vision-based systems require relatively simple and cheap infrastructure (cameras) and very cheap tags (paper printouts). For real-world applications, full camera supervision of office spaces raises privacy concerns. However, the discernible sensors of visual-based technologies generate less anxiety among privacy advocates than RF-based systems because of their stealthy nature. Vision-based systems require a clear line-of-sight that reduces their reliability. For experimental applications, though, such systems provide a useful solution and possess potential for adaptation to the real-world.

It can be concluded that there is no perfect location system for self-updating building models today. Vision based methods appear as the most appropriate solutions that can form a basic infrastructure to the requirements because of being software-supported and open for modifications and improvements. The latest developments in distributed programming, software agents and high power processors, also make the vision-based solutions more promising. Based on this technical review, such a system is adopted as described in the following chapters.

3.2 Occupancy Sensing

Occupancy sensors operate based on detection of motion, assuming that occupancy creates movement. There are two types of off-the-shelf occupancy sensors, passive infrared (PIR), and ultrasonic.

PIR sensors sense infrared heat radiated from the human body (10 micron wave lengths). Because there can be other sources of heat at the same temperature, the sensors respond to changes in position of the source of heat.

Ultrasonic sensors emit an inaudible high frequency tone. Like sonar, the tone bounces off the objects in the room and returns to the sensor. If there is motion, the acoustical response changes and occupancy is sensed. When occupancy is sensed (by either type of sensor), the occupancy flag is kept risen until no motion is detected for approximately 15 minutes. These sensors have a limited sensing range. They can detect slight hand motion up to 3 meters and full body motion up to 10 meters. Ultrasonic sensors offer better detection than PIR sensors. In rooms where it is critical to sense occupancy accurately, dual technology (PIR and ultrasonic) sensors can be used.

On the other hand, it is also possible to integrate a vision-based solution for occupancy sensing by using the same infrastructure developed for location. Motion detection from video image sequences is a method currently being developed and used for surveillance purposes (Wildes 1998, Toth et al. 2000). This method also provides a robust detection in adjustable sensitivity levels.

4 SENSOR EVALUATION FOR VISON-BASED TECHNOLOGIES

Based on the technical review described in previous chapter, vision-based methods appear to be the most feasible solution towards the requirements of a self-updating model generation. Subsequently, different camera technologies are examined in the course of selecting an appropriate sensor. During the review process, four major camera technologies are evaluated in accordance with the system requirements defined in chapter 2.

4.1 Web Cameras

Web cameras (webcam) are digital cameras that enable image transfer on the Internet. However, web cameras need a separate computer designated for the image communication. These cameras utilize USB data communication standard for connection to the computers. When connecting multiple cameras to a computer, USB extension cards must be used for additional cameras. USB hubs can also be used for multiplexing USB ports, but this critically drops the data transfer speed (for USB 1.1 standard, data speed is 12 Mbits/sec). Utilizing cameras that support USB 2.0 standard (480 Mbits/sec) may reduce the number of required capture cards. USB standard has a cable length limit of 5 meters. In order to extend this limit, USB hubs are used as cable extenders.

Image acquisition from web cameras can be accomplished with DirectShow API's provided by Microsoft company. DirectShow is a software development kit that enables hardware-independent software design for communication with various multimedia devices. Web cameras are among these devices, which are supported by DirectShow compatibility. Figure 4 demonstrates a brief layout for the utilization of web cameras in an office space.

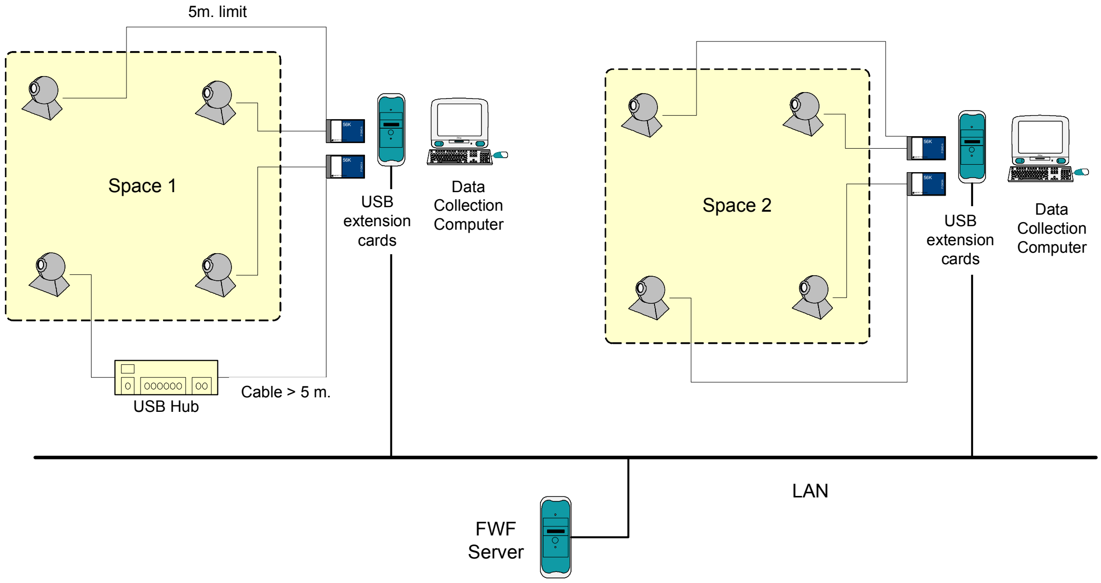


Figure 4. Web camera connection scheme.

Most of the web cameras come with a fixed lens, and provide low image quality in order to cope with the connection speed. However, there are some web camera solutions that provide relatively higher image quality, and changeable lenses (C/CS mount).

4.2 Digital Photo Cameras

Digital photo cameras shoot high resolution and high quality images, but acquiring them in a software application is not as straightforward as web cameras. Most of the digital photo camera manufacturers also use USB standard for computer connection, and develop software libraries to allow image transfer to external applications. However, utilizing such a method results in an undesired hardware-dependent solution. As the connection structure is very similar to the above USB system, additional information is not given specifically for these cameras.

4.3 CCTV Cameras

CCTV (close circuit television) cameras are conventional type of television cameras that generate analog video output. The structure of CCTV camera connection is given in Figure 5.

Since CCTV cameras generate analog video input, video capture cards must be utilized to digitize the images. These cards capture frames from analog video input with a specific frame rate, and subsequently encode them to a predefined image or video format. Most of these cards work on PCI bus. Even though some capture cards comply with DirectShow standard, there is no common support in video streaming like web cameras. Therefore, data acquisition from such cameras requires PCI bus programming. Video capture cards enable multiple analog camera inputs, however, they still cost a substantial additional expense to the cameras because of the complexity of hardware components they involve.

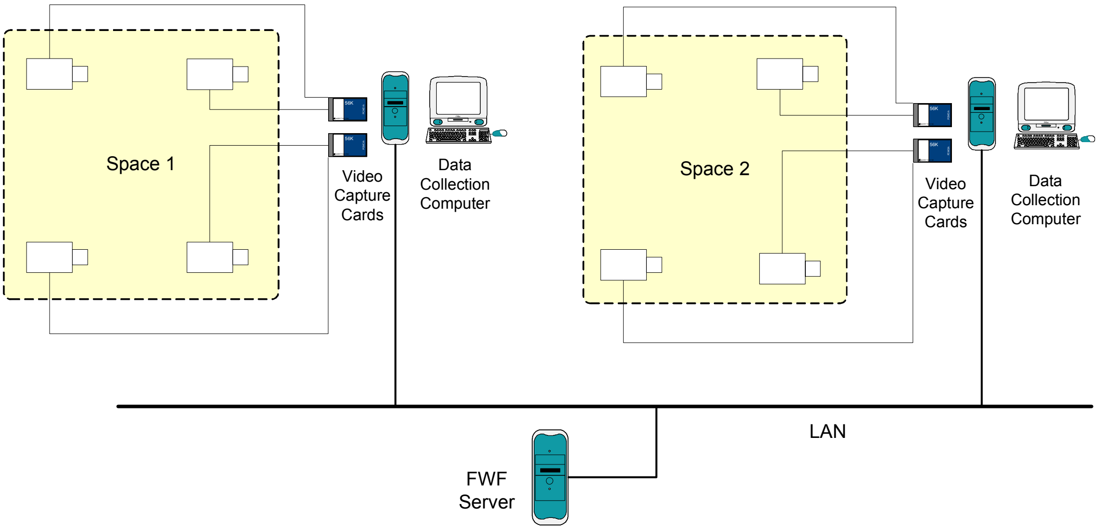


Figure 5. CCTV camera connection scheme.

CCTV cameras are widely used for various purposes. Most of them provide higher image quality and larger resolution when compared with webcams, since the frame rate and connection speed is not a matter of concern for these cameras.

4.4 Network Cameras

Recent developments in embedded computing led to the integration of sensors with processors. This reduced the costs, as dedicated computers were not necessary to enable data communication. Thus, data could be efficiently conveyed over large-scale networks. In vision sensing domain, such developments gave rise to network cameras (netcams). Netcams have embedded computing power that enables image relay over Internet via standard protocols. They also enable the control of third-party devices like pan-tilt units (P/T) through the same communication channel.

Figure 6 demonstrates the installation scheme of netcams in an office space. A network camera can be described as a camera and a computer combined. It has built-in software for a web server and FTP server, and it is connected directly to the network as any other network device. However, like other network devices, netcams are designed to relay data over network as fast as possible, therefore, they apply compression prior to data transfer. This results in the generation of relatively low-quality images, low resolution and blurred frames having lost the sharp details.

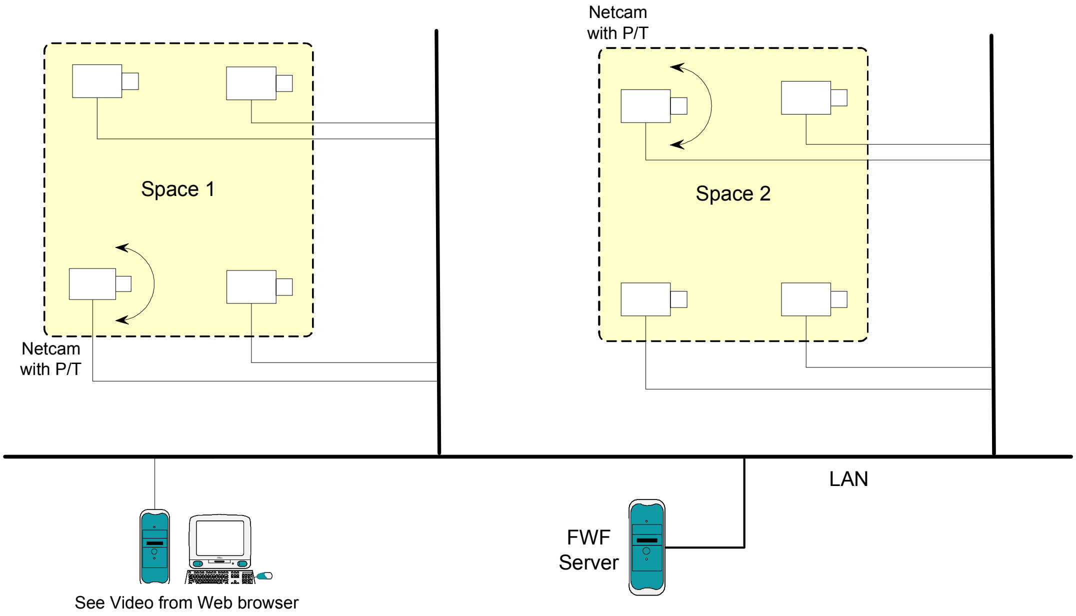


Figure 6. Network camera connection scheme.

Network cameras broadcast their images on the Internet through HTTP protocol. Frames can be seen from the web browsers. Some cameras require the installation of a plug-in or activex component for decoding the compression, where the camera's compression format is different from the ones that web browsers can read. These components can also be utilized in external applications for data acquisition. If such a specific compression is not a matter of concern, any application programmed to act as a web client can easily control the netcams and acquire images.

4.5 Technology Evaluation

Table 2 provides a qualitative comparison of the four main camera technologies considered above.

The web camera, standalone, is the most efficient technology regarding cost. However, expanding the number of web cameras is not feasible in a building environment, since dedicated computers are needed for transmitting the images over the network. The processing power of these computers can be utilized in the overall system, but this design will eventually prevent the configurability, and the system's processing capabilities will be bound down to the web cameras.

Table 2. Qualitative comparison of camera technologies.

	Webcam	Digital	CCTV	Netcam
Image Quality	Medium	High	High	Medium
Resolution	Low	High	Medium	Medium
Changeable Lens	No	Yes	Yes	Yes
Standard Data Comm.	Yes	No	Partly	Yes
Pan-Tilt Control	No	No	No	Yes
Cost (camera only)	Low	High	High	High
Cost (additional)	High	High	High	No cost

Even though they provide a more effective image quality, digital photo cameras and CCTV systems also posses the same drawbacks with the web cameras. These devices also require a sophisticated and hardware-dependent software development for image transfer.

Netcams relatively posses lower image quality when compared with CCTV and digital photo cameras. However, if a building is equipped with a network, the required infrastructure is already in place for netcams. In the typical office, computers are most likely connected via an ethernet network, e.g. a local area network (LAN). Network cameras comply with this standardized structure, and use either conventional ethernet protocol (IEEE 802.3, 100 Mbps) or wireless ethernet protocol (IEEE 802.11, 11 Mbps). Each device in a LAN must have a unique address, the IP address, to be able to connect directly to the Internet. This standardization enables the identification of cameras individually in a global environment. Today's computers and network devices have a high capacity to simultaneously communicate with several different units. A high-end network camera can send images to ten or more computers simultaneously. Therefore, they can easily be adapted to a prospective vision-based solution.

Towards this end, network cameras are selected as the proper sensor for the project. They also comply with requirements of a built environment, since they make use of the existing network installation without requiring additional infrastructure, and enable the control of third-party devices like pan-tilt units that effectively increase the monitoring ranges.

4.6 Pan-Tilt Units

Pan-tilt units are motorized mechanisms that provide rotation capability to cameras in vertical and horizontal directions. The employment of such units increases the covered space, which is otherwise achieved by either adding new cameras or increasing the field-of-views (FOV) of current ones. Increasing the FOV seems like the more efficient solution, however, high FOV (using low focal length) has some drawbacks like lower image detail and visible barrel-effect.

Pan-tilt mechanisms are comprised of two main parts: head and controller. P/T head is the part where the camera or the camera housing is mounted. It involves two motors for executing the pan and tilt functions. P/T controller (a.k.a. telemetry device) is an electronic interface that enables the control of P/T head by an external device. It receives control signals from the external device (mostly through the serial communication port) and converts them to proper electrical voltages that drive the P/T head. The external device can either be a keyboard-controller used widely in surveillance systems, or a PC. In our case, the external devices are the network cameras each of which substantially possesses the functionalities of a PC.

4.6.1 Communication

4.6.1.1 Physical Layer

The physical layer determines the cable connections and electrical voltages that correspond to 0s and 1s. RS232, RS485 and RS422 are some of the most common protocols used in this layer. Network cameras support serial communication connections like RS232 and RS485. Figure 7 illustrates the connection scheme of a pan-tilt unit mounted on the ceiling. Analogous to camera control and image acquisition commands, netcams receive pan-tilt control commands over the network, and convey them to the P/T controller through the serial port.

Even though network cameras are designed for a standardized solution, a contradictory situation arises at this point. Some network cameras accept every serial data string, and directly convey it to the P/T controller for execution. On the other hand, some network cameras accept predefined P/T control commands, and convey them to the P/T controller after converting to proper serial signals that the P/T controller can understand. Such cameras achieve this conversion with the guidance of an internal look-up table filled by the netcam user. A prospective solution must take both of these situations into consideration in order to comply with every network camera model. However, in either case, serial communication support of the netcams gives the possibility to handle pan-tilt functions simply like any other camera control.

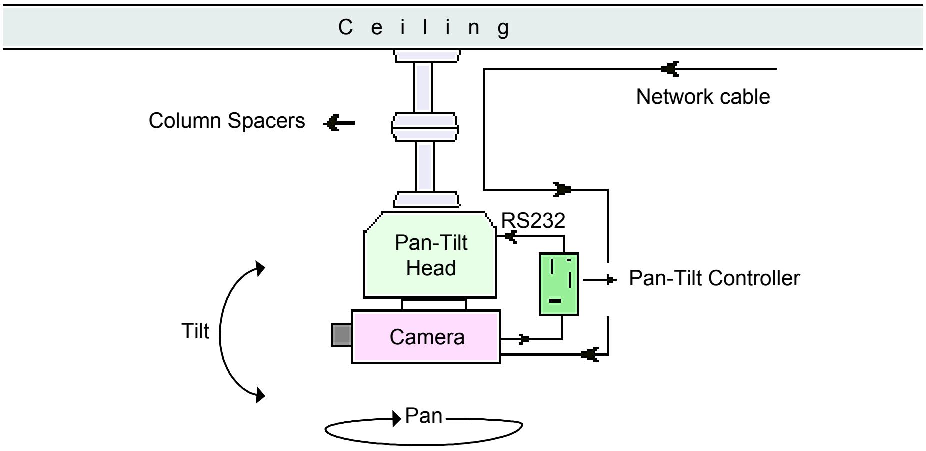


Figure 7. Pan-Tilt unit installation scheme.

4.6.1.2 Data Link Layer

The physical communication between the P/T controller and external device is performed generally by serial protocols like RS232, RS485, or RS422 as mentioned above. Data link layer resides at the top of the physical layer, and these two layers together define how the communication is set between the P/T controller and camera.

The data link layer defines the data flow control, error handling and handshake between the two nodes. CRS (cyclic-redundancy-check), checksum are some of the famous error-handling algorithms, and CTS/RTS (clear-to-send/ready-to-send), XON/XOFF are some of the popular flow-control methods. The protocols of this layer combine these methods, and provide an error-free communication. In contradiction to physical layer, there are no standard protocols for data link layer among P/T controllers. Main pan-tilt manufacturers develop their own protocols for communication. Among these proprietary protocols, some do not provide error handling, whereas some make use of CRS or checksum. Handshake between the nodes is usually not applied. Each protocol uses its own flow control, which is usually a simple quasi stop-and-wait method. These P/T controller protocols also define the set of commands (like pan-left, tilt-right...etc) used to control the P/T heads.

4.6.2 Position Feedback

Pan and tilt position recall is achieved by factory fitting inside the P/T head for both pan and tilt functions with high-grade servo potentiometers. These potentiometers are used to feedback positional information to the P/T controller. The P/T controller must also support this functionality to convey position information back to the camera, and eventually back to the vision-based sensing system. Controllers exploit non-volatile memories to store a fixed number of preset positions defined by the user, and rotate the P/T heads to these preset pan and tilt angles. Additionally, some controllers move the P/T head directly to a given absolute or relative pan-tilt angle. This functionality is defined in the command set of the controller protocol.

Position feedback is crucial for the system, since the exact viewing location of the camera must be known to convert the identified object locations to the real-world coordinates.

5 OBJECT IDENTIFICATION AND LOCATION SENSING

As explained in chapter 3, vision-based methods are preferred for an appropriate solution towards the generation of self-updating building models. Therefore, the object identification and location sensing system is designed on a similar approach that uses a combination of visual markers (*tags*) and cameras. The selection of a proper camera technology is described in the previous chapter. In this chapter, the selection of a feasible location sensing technology, and its adaptation to the system requirements are given.

Among the reviewed vision-based methods, the algorithm proposed in TRIP (Lòpez 2002, Lòpez et al. 2002) offers a suitable solution for location sensing in building environments. The TRIP algorithm uses optimized image processing methods, and obtains in real-time the identification and location (both position and orientation) of an object to which the visual tag is attached. Furthermore, the proposed method utilizes low-cost tags that do not require constant maintenance, and power supply.

5.1 Location Sensing

For location sensing, TRIP uses “pose from circle” algorithm (Forsyth et al. 1991, Trucco & Verri 1998), estimating the pose of a circle in space from a single image. The idea behind the algorithm is illustrated in Figure 8. A circle on the target plane generates an ellipse on the image plane of the camera. From the known parameters of the ellipse, it can be back-projected to the original circle, enabling the extraction of the orientation and the position of the target plane with respect to the camera origin.

Towards this end, if tags with *reference circles* are attached on the objects, their location can be estimated by the agency of the algorithm with respect to the viewing cameras. The details of the algorithm and the overall location sensing process are given in the following sections.

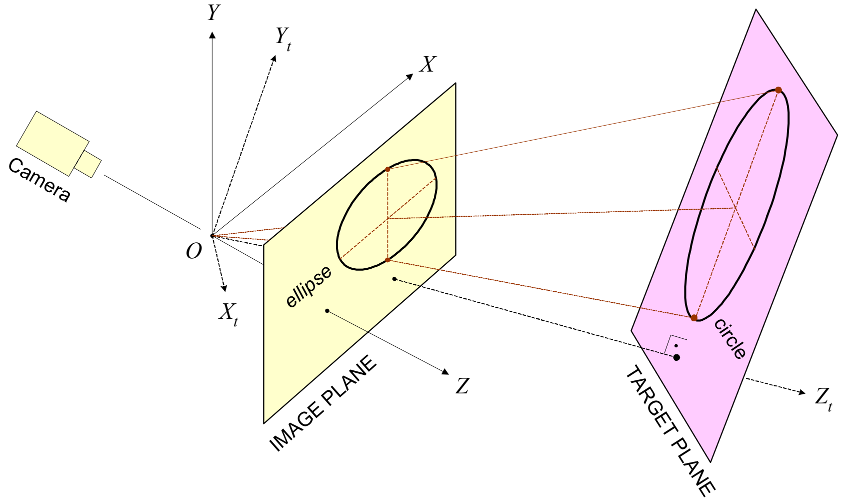


Figure 8. "Pose from circle" algorithm.
(X, Y, Z) denotes the coordinate system of the image plane, whereas (X_t, Y_t, Z_t) denotes the coordinate system of the target plane. The outcome of the algorithm is the parameters of the transformation between two coordinate systems. This transformation also defines the location of the target plane with respect to the image-plane-coordinate-system.

5.2 Object Identification

In TRIP system, object identification is also performed with a method similar to the barcode principle: special marks are placed around the reference circle used for location sensing. Particularly, two *code ring*s are utilized, one surrounding the other. The reference circle lies concentrically in the center of these code rings. This arrangement enables for ternary coding (each code ring corresponds to one binary digit), where the fourth combination is reserved as a special sign to designate the starting point.

Unlike the TRIP system, binary coding is preferred for a more convenient structure. Even though binary coding reduces the number of uniquely identifiable objects, it enables the production of smaller tags that would be less discernible with one code ring. For the tag generation, code ring around the reference circle is divided into 16 equal sectors (Figure 9), each one resembling a pie slice. The presence or absence of the "black" mark on the sector denotes the "1" or "0" coding respectively. The pattern of "0111" code sequence defines the start bits, and is never repeated elsewhere in the rest of the data string.

The identification number is encoded in the remaining 12 sectors. Finally, an even-parity bit is added at the center of the reference circle for the verification of the decoded data string in the end of the identification phase. This coding structure enables the definition of 2031 distinct tagged objects. The tags consume 12×12 cm^2 area, and they can be printed using regular black-and-white printers. This is one of the main benefits of the system, since the tags are low-cost, low-maintenance, and require no power input.

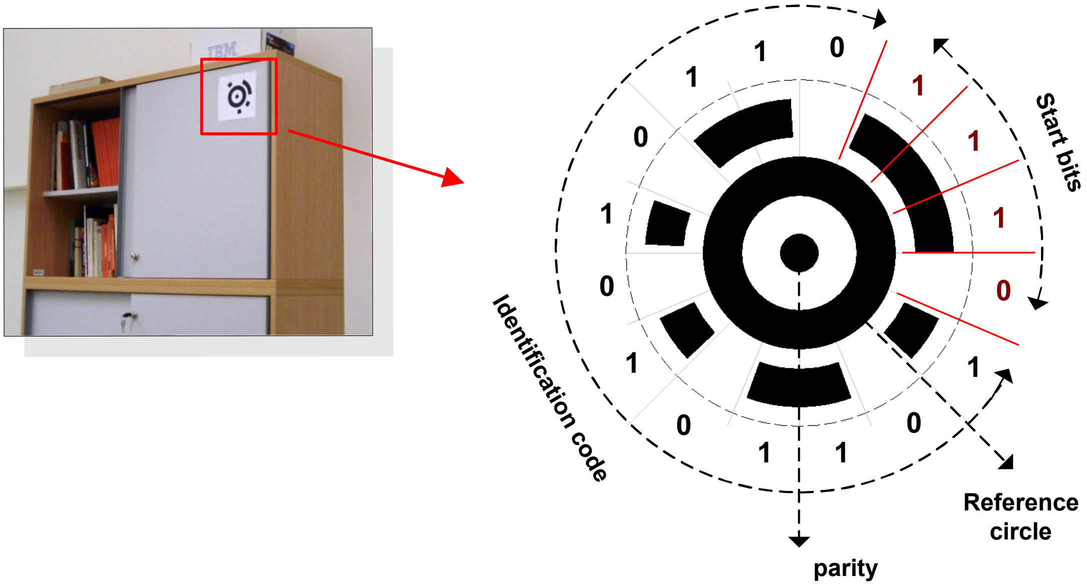


Figure 9. Tag structure is illustrated with a sample tag
coded with 0111-011010101101 data string (even-parity = 1). Identification number
corresponds to 1709 in decimals.

5.3 Original Method

The TRIP system divides the object identification and location sensing procedure into two phases (Figure 10). First is the "target recognition" phase, where the tags are detected, parameters of the *reference ellipses* (projection of the reference circle on the camera image) are extracted, and the identification numbers are decoded. Second is the "pose extraction", where the locations of the tags are computed from the outputs of the first phase (Lòpez et al. 2002).

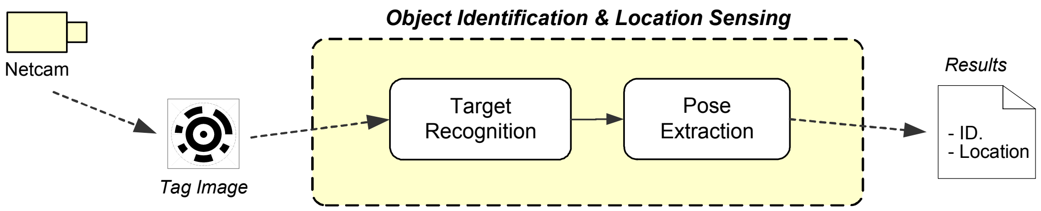


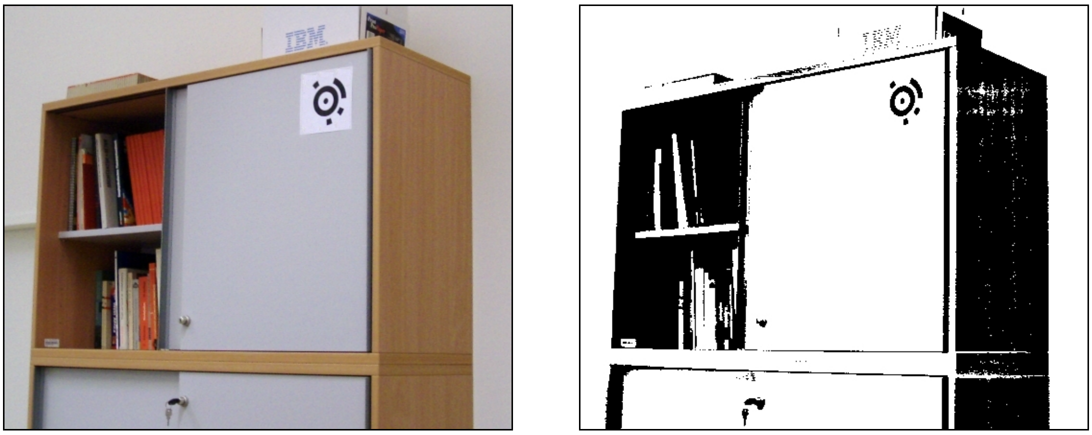
Figure 10. Original method used in TRIP system.

5.3.1 Target Recognition

Target recognition is the first phase that determines the geometric properties of the projections of tags in the images. By using the geometric properties, this procedure, subsequently, extracts the identification number encoded in the tags. In the course of accomplishing these actions, six sequential procedures are applied.

5.3.1.1 Binarization

Images are, firstly, binarized with an adaptive thresholding function, where a variable threshold value is employed taking the background illumination of each pixel into consideration (Wellner 1993). This procedure provides a robust image analysis furthermore, against variable lighting conditions. Figure 11 exemplifies the result of the binarization procedure. This procedure is applied to gray scale images, however, the original images captured by the cameras are defined in RGB format (Figure 11a). Prior to the implementation of the binarization process, the images are converted to gray scale. The details of the conversion are given in Appendix A.



(a) (b)

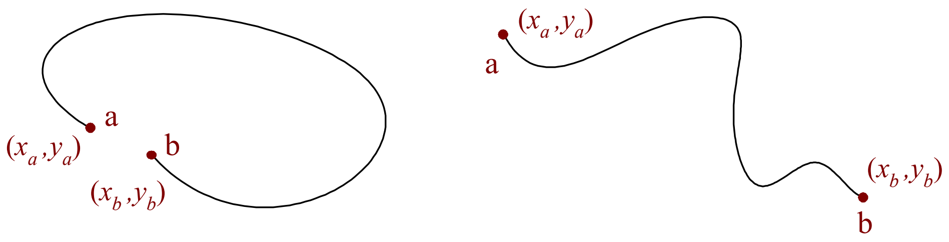
Figure 11. (a) Original image captured from the network camera. (b) Result of the binarization after being applied to the input image.

5.3.1.2 Edge Detection

Following binarization, one-pixel width edges in the binary image are extracted by applying an optimized binary edge detection process. This process is applied in TRIP system as follows: The “black” pixels that have a perpendicularly adjacent (four-connected) neighbor pixel with “white” intensity value and a diagonally adjacent (eight-connected) pixel with “black” intensity value are designated as edge points. Figure 13a illustrates the results of edge detection applied on the binary image.

5.3.1.3 Edge Filtering

Since the projection of a circle in an image generates an ellipse, tags' reference circles are observed as elliptical. The connected chains of edge points, located in the edge detection phase, are tracked in clockwise direction. The edge points, whose shape defines an ellipse are extracted by a filtering process. Filtering is performed with analyzing the ratio between the Euclidean distance of the extreme points of the edge and its pixel length. The details of the analysis are given in Figure 12 and Eq1.

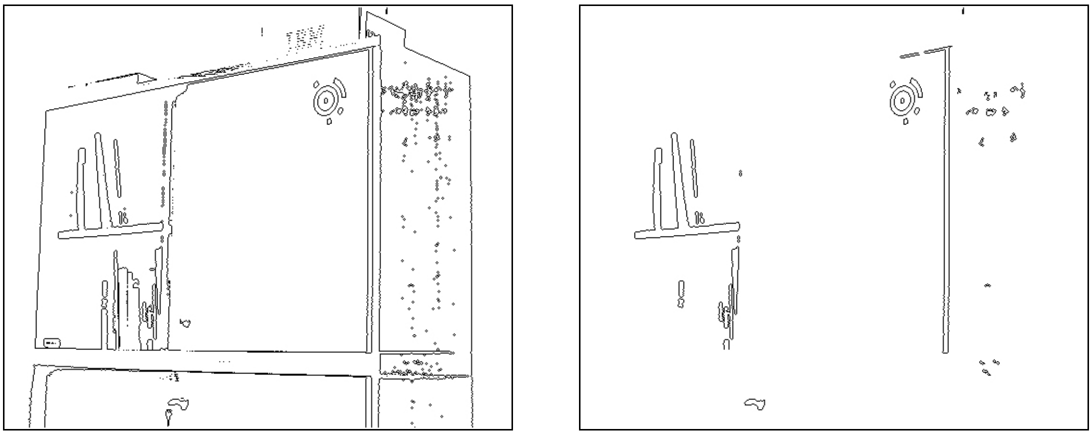


(a) (b)

Figure 12. (a) Connected edge points for a prospective ellipse. (b) Connected edge points that will be eliminated after the filtering process.

$$\begin{array}{l} L = \textit{connected edge points} \\ (a,b) = \textit{extreme points of } L\textit{, if} \\ T = \textit{threshold value} \end{array} \begin{cases} \dfrac{\sqrt{(x_a-x_b)^2+(y_a-y_b)^2}}{\textit{pixel length}\,(L)} < T\,, & L = \textit{ellipse} \\ \textit{else,} & L \neq \textit{ellipse} \end{cases} \qquad \text{Eq.1}$$

Eventually, the filtered connected edge points correspond to candidate tag's circular borders (Figure 13b).



(a) (b)

Figure 13. (a) Result of edge detection. (b) Result of edge filtering.

5.3.1.4 Ellipse Fitting

The purpose of this process is to obtain the ellipse parameters that best approximate the edge points. Therefore, "Direct Least-squares Ellipse Fitting" method (Pilu et al. 1996) is applied to each elliptical edge candidate encountered in the previous step. This procedure extracts the conic function (a second order polynomial given in Eq2) that represents the ellipse.

$$ax^2 + bxy + cy^2 + dx + ey + f = 0 \qquad \text{Eq.2}$$

An ellipse can also be expressed in parametric form as formulated in Eq3, and illustrated in Figure 14, where (x_0, y_0) is the centre point, a, b are the axes and θ is the orientation with respect to x-axis in clockwise direction. The stroll of t-parameter from 0 to 2π defines the associated ellipse. This parametric equation can be calculated from the cubic function extracted by the ellipse fitting process.

$$\begin{aligned} x &= x_0 + a.\cos(\theta).\cos(t) + b.\sin(\theta).\sin(t) \\ y &= y_0 - a.\sin(\theta).\cos(t) + b.\cos(\theta).\sin(t) \end{aligned} \qquad \text{Eq.3}$$

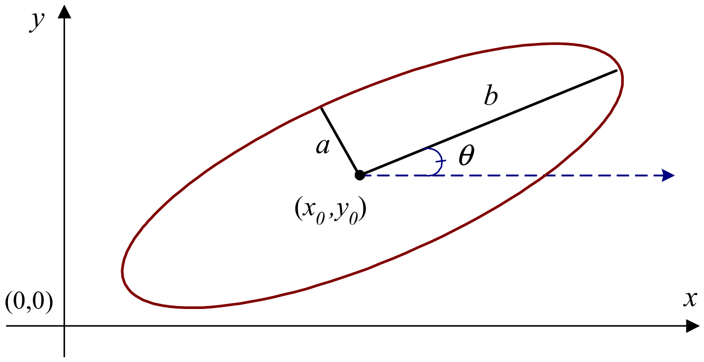


Figure 14. Geometric parameters of an ellipse.

"Direct Least-squares Ellipse Fitting" method requires minimum six values on the connected edge points in order to fit an ellipse. Therefore, the edge points must be longer than six pixels in the image. With the application of the method, it is possible to acquire always *one and only one* elliptical solution for each chain. The method is designed for specifically fitting ellipses in the least-squares sense. Further details about the method are given in Appendix B.

5.3.1.5 Concentricity Test

After the fitting process, ellipses are examined for concentricity. As seen in Figure 13b, all of the ellipses do not belong to the tag, but edges of circular objects still remain in the image. In order to eliminate these false ellipses, the results are undergone the concentricity test. As illustrated in Figure 15, a tag possesses at least two concentric ellipses after edge detection. Therefore, two or more ellipses sharing the same center and orientation are designated as tag circles. Among these, the outermost ellipse is used as base trajectory for the next step, code deciphering.

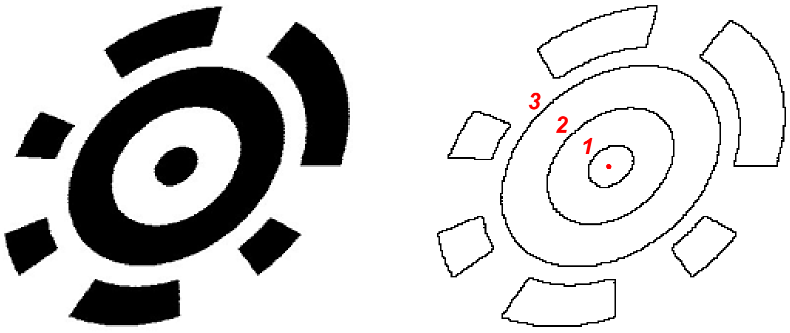


Figure 15. A tag possesses at least two concentric ellipses in the edge-detected image. With the addition of even-parity bit, the number of concentric ellipses may be three.

5.3.1.6 Code Deciphering

Among the ellipses that pass the concentricity test, the outermost ones are used as base trajectory for code deciphering. The outermost ellipses are marked as *reference ellipses*, and their parameters are exploited for pixel sampling procedure applied on the binary image for ID number extraction.

First, each reference ellipse is transformed to *unit circle* by a rotation process applied with θ degrees in counterclockwise direction, and subsequently, by a stretching process applied with $\frac{1}{a}$ and $\frac{1}{b}$ factors to x and y axis respectively.

Thus, the parametric equation of the ellipse given in Eq3 is transformed into the general equation of the unit circle (Eq4). The details of such geometric transformations are explained in Appendix C.

$$\begin{aligned} x &= x_o + \cos(t) \\ y &= y_o + \sin(t) \end{aligned} \qquad \text{Eq.4}$$

Since the distance between the reference circle and the code ring is fixed in the tag design, the code ring's sector points are easily determined with respect to the unit circle. These point locations are transformed back to the corresponding image pixel locations using the inverse transformation of the one employed to convert the reference ellipse to the unit circle. The intensity values of the pixels in the binary image determine the values on the code ring.

In a detailed explanation, it can be demonstrated that the back-transformation applied on the unit circle gives the parametric equation of the code ring (Eq5) on the image.

$$\begin{aligned} x &= x_0 + a'.\cos(\theta').\cos(t) + b'.\sin(\theta').\sin(t) \\ y &= y_0 - a.'.\sin(\theta').\cos(t) + b'.\cos(\theta').\sin(t) \end{aligned} \qquad \text{Eq.5}$$

Since the code ring's projection on the image is an outer ellipse surrounding the reference ellipse (Eq3), the constants in Eq5 are expected to prove the following; $a' > a$, $b' > b$, and $\theta' \approx \theta$.

First, the code ring is scanned for determining the position of the start bits (Figure 16). In order to perform the scan process, t-parameter, defined for the code ring (Eq5), is strolled from θ in $\frac{2\pi}{16}$ intervals. In each step, three pixel samples are taken. The correspondence of any of these pixels to "black" defines the "1" coding, and the contrary situation defines the "0" coding. The scan continues until the starting pattern is designated, or the rotation reaches the end, $t = \theta + 2\pi$. During the scanning, the outermost point that corresponds to the transition from "0" to "111" in the start-bits pattern is marked as *synchronization point* to be used further for pose extraction (depicted as a star in Figure 16).

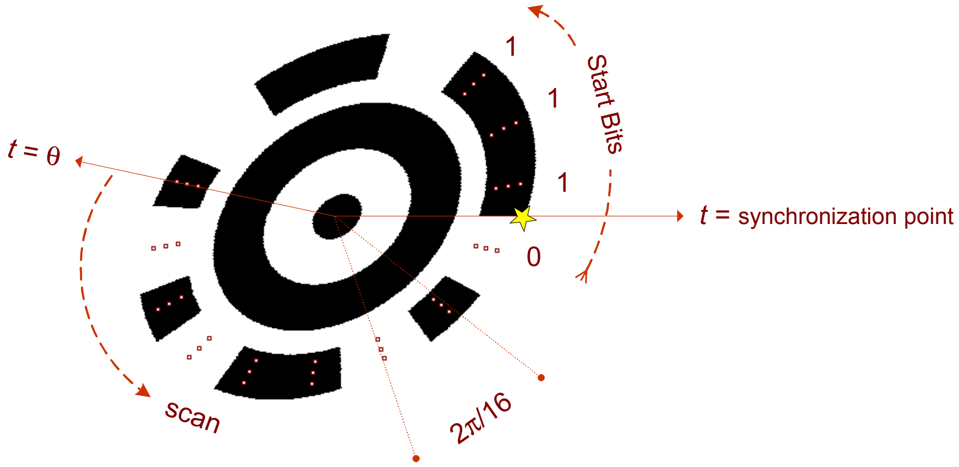


Figure 16. Code ring is scanned, first for designating the start bits, then for extracting the ID number. The dot-trio on the code ring represents the pixel samples for each sector.

After the designation of the start bits, the code ring is scanned one more time to extract the entire ID number. Finally, the ID number is validated with the even-parity check bit. In cases where the starting bits cannot be found or the ID number is not validated by the parity check, the tag is marked as *spurious*.

5.3.2 Pose Extraction

As introduced in section 5.1, TRIP utilizes “pose from circle” algorithm (Forsyth et al. 1991, Trucco & Verri 1998) that estimates the pose of a circle in space from a single image. However, this algorithm alone is not sufficient to give out the exact location of the objects. The complete “pose extraction” method (Lòpez 2002) used for location sensing is described in this section.

Pose extraction is applied to the outcomes of the target recognition process described in section 5.3.1. Pose extraction method takes as input the following parameters: (1) the conic function representing the reference ellipse (Eq2), i.e., the outermost border used as base trajectory for code deciphering, (2) coordinates of the synchronization point (Figure 16), (3) radius of the reference circle, a fixed value in the sighted target. Reference circles are printed on the tags with this fixed radius value.

The method involves four major phases. Initial two phases are comprised of the “pose from circle” algorithm. First phase enables the extraction of the target plane (tag plane) orientation by using the reference ellipse conic function. The tag position is computed in the second phase, where trigonometric rules are applied on the fixed tag radius. As mentioned above, “pose from circle” algorithm possesses a meager capability, and extracts only the plane orientation that lacks the plane rotation value. The rotation angle is computed in the third phase with the utilization of the synchronization point. In the final phase, the ambiguity arisen by the nature of the “pose from circle” algorithm (explained in section 5.3.2.1) is removed. As a result of four sequential procedures, the translation vector, $\vec{T} = (T_x, T_y, T_z)$, and rotation angles, (α, β, γ), that define the rigid body transformation between the camera coordinate system, (X, Y, Z), and the target-centered coordinate system, (X_t, Y_t, Z_t), are returned (Figure 8). This rigid coordinate transformation is a means of defining the location of the tags with respect to the viewing camera. Details of the each phase, mentioned above, are described in the following sections. Preliminary information about such 3D transformations like translation and rotation are described in Appendix C.

5.3.2.1 Extracting the Plane Orientation

The first phase of pose extraction establishes a *homography*, i.e., a projective transformation from one plane to another, that back-projects the reference ellipse on the image plane into its actual circular form on the target plane (Figure 17). As a result of this transformation, the orientation of the target plane is extracted with respect to the camera coordinate system.

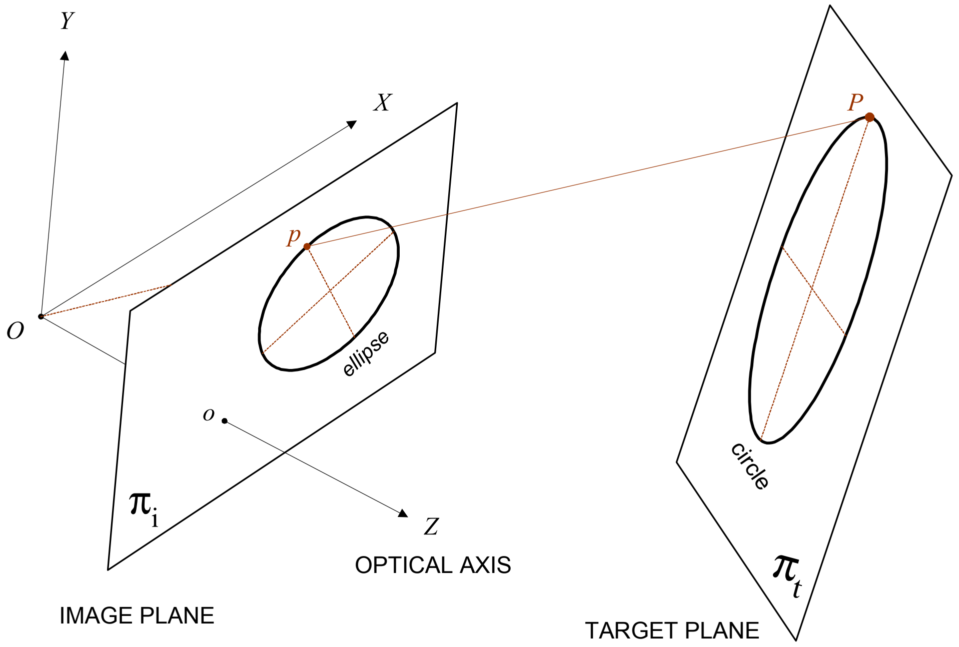


Figure 17. The geometry of pinhole camera model.

5.3.2.1.1 Camera Calibration

An important assumption of this procedure is that the subject camera is calibrated. On the left side of Figure 17, the most common mathematical model of an intensity camera, *the pinhole camera model*, is illustrated. The pinhole camera model consists of image plane, π_i, and a 3D point, O, the *focus* or *center of projection*. The distance between π_i and O is the *focal length*, f. The line through O and perpendicular to π_i is the optical axis, and o, the intersection between π_i and the optical axis, is the *image center* or *principal point*. Based on this model, a camera undertakes a perspective projection from the 3D projective space to the 2D projective plane. This projection is performed with an optical ray (light beam) reflected from a scene point. A sample scene point, P, is illustrated in Figure 17. The optical ray reflected from P passes through the center of projection and the image plane at the point p. The camera image is formed with the projection of such infinite 3D scene points to the finite 2D image points (pixels) on the π_i. The 3D reference frame in which O is the origin and the plane, π_i, is orthogonal to the z–axis is called the *camera frame* or *camera coordinate system*. The point locations in the camera coordinate system are expressed in metric units. The 2D reference frame defined on the plane, π_i, in which o is the center is called the *image coordinate system*. The point locations are expressed in pixel units.

In target recognition, all of the mentioned image processing methods are performed on the image coordinate system. Therefore, no camera calibration is required. On the other hand, the pose extraction method determines the location values in 3D space with respect to the camera coordinate system. However, one of the main inputs of the method, extracted conic function representing the reference ellipse (Eq2), is defined in the image coordinate system. The camera calibration is utilized at this point, since it enables the transformation of the pixel locations from the image coordinate system to the camera coordinate system.

In order to perform the camera calibration, the *intrinsic parameters* of the camera must be acquired. The pixel coordinates in the image do not correspond to the physical coordinates in the image plane. The relation between both depends on the perspective projection, the size and shape of the pixels, and the position of the light-sensor chip (CCD) in the camera. To find the transformation between the camera and pixel coordinates, the coordinates of an image point, (u, v), in pixel units must be linked with the coordinates of the same point, (x_c, y_c, z_c), in the camera reference frame as shown in Figure 18. Moreover, there may be skew (non-orthogonality) between the pixel axes, as illustrated with α in the same figure. For standard cameras, it is represented by a factor, k_α, with a value close to zero.

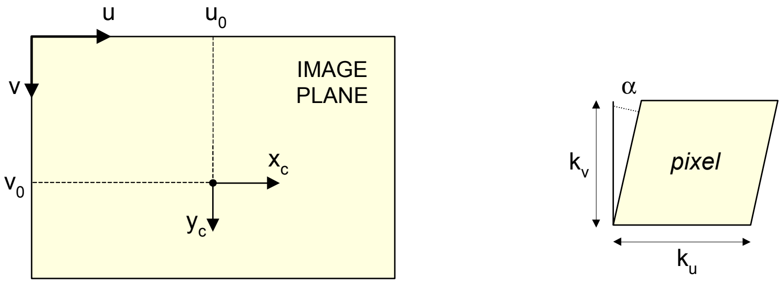


Figure 18. Camera intrinsic parameters.
The point is lying on the image plane, therefore $z_c = f$ in this example.

Therefore, the intrinsic parameters of a camera are defined as (1) the focal length, f, (2) the location of the image centre in pixel coordinates, (u_0, v_0), (3) the effective pixel size in the horizontal and vertical directions, (k_u, k_v), and (4) the skew k_α. For most CCD cameras, the pixels are almost-perfectly rectangular, thus $k_\alpha \approx 0$.

To perform the transformation between the image and camera coordinates, a mapping, M, between the pixel coordinates $w = [u \quad v]^T$ and the camera coordinates $p = [x_c \quad y_c \quad z_c]^T$ is defined in homogenous form (Eq6).

$$\begin{bmatrix} u' \\ v' \\ s \end{bmatrix} = M.\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}, \quad \text{where} \quad M = \begin{bmatrix} f/k_u & k_\alpha & u_0 \\ 0 & f/k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.6}$$

The Cartesian pixel coordinates, (u,v), are obtained by the normalization of $u`$ and $v`$ values with $s = z_c$. So, u and v are computed by: $u = u`/s$, and $v = v'/s$. This also normalizes the depth coordinate of the pixel frame, i.e., the focal length, to 1. Normalization is actually the loss of the depth data, which is an evident result of mapping from 3D to 2D. Thus, in a back-transformation from image to camera coordinate system, the depth component, z_c, cannot be reconstructed, instead, every point is defined in terms of the focal length, namely in focal-length units.

For a camera with fixed optics, these parameters are identical for all the images within the camera. For cameras with zooming and focusing capabilities, the focal length and the principal point location in the image can vary. Nevertheless, for auto-focusing cameras, the intrinsic parameters vary slightly, and they can reasonably be considered fixed. For auto-zooming cameras, the intrinsic parameters are usually calculated for a small set of zooming factors, and considered fixed only within those zooming intervals. The network cameras support the utilization of such adjustable focus and zoom lenses. However, in the project, the cameras are fixed to a constant focal length value. Therefore, the cameras are calibrated once, before being employed in the system. The computation of the required camera intrinsic parameters is described in section 10.1. The explanation of “pose from circle” algorithm, henceforth, assumes that the subject camera is already calibrated, i.e., the camera intrinsic parameters are known.

5.3.2.1.2 The Algorithm

As a general notation, the ellipse curve is defined in 3D as a cross-section of a cone:

$$ax^2 + bxy + cy^2 + dxz + eyz + fz^2 = P^T \cdot C \cdot P = 0 \qquad \text{Eq.7}$$

where ^{T} represents the matrix transpose, $P = [x \quad y \quad z]^T$ is a point on the curve, and C is the real symmetric matrix of a cone:

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \qquad \text{Eq.8}$$

As mentioned before, the edge image of the outermost border (Figure 15) is selected as the reference ellipse, and is used as the base trajectory for code deciphering. The same ellipse is also utilized for pose extraction. However, the reference ellipse parameters are extracted in the image plane as defined in Eq2. “Pose from circle” algorithm assumes that the reference ellipse is expressed in the camera coordinate system rather than in the image-plane pixel coordinates as returned by the target recognition process. Therefore, the image origin must be at the principal point, and the distances must be transformed into focal-length units. To achieve this, the real symmetric matrix of cone, C, (expressed in pixel coordinates) is normalized by using the matrix M (Eq6) of intrinsic camera parameters:

$$C_n = M^T \cdot C \cdot M \qquad \text{Eq.9}$$

The orientation of the circle's plane, π_t, is found by rotating the camera so that the intersection of the cone with the image plane becomes a circle, which happens when the image plane, π_i, is parallel to the target plane (Figure 17). This rotation, R_C, is estimated as the composition of two successive rotations, namely R_1 and R_2:

$$R_C = R_1 \cdot R_2 \qquad \text{Eq.10}$$

The first rotation, R_1, is determined by diagonalizing C_n, i.e., removing the coefficients with terms in xy, xz and yz. This 3D rotation transforms the ellipse matrix C_n into C'.

If $\lambda_1, \lambda_2, \lambda_3$ are the eigenvalues of C_n, (arranged in the order of $\lambda_1 < \lambda_2 < \lambda_3$), and $\vec{e}_1, \vec{e}_2, \vec{e}_3$, are the corresponding eigenvectors, then:

$$R_1 = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 & \vec{e}_3 \end{bmatrix} \qquad \text{Eq.11}$$

When applied on a single point, the rotation R_1, transforms the point, $P = [x \quad y \quad z]^T$, residing on the ellipse C_n, to the point, P', residing on the ellipse C' (Eq12).

$$\vec{P'} = R_1^T \cdot \vec{P} \qquad \text{Eq.12}$$

When applied on a cone matrix, R_1 transforms ellipse matrix, C_n, into C' (Eq13).

$$C' = R_1^T \cdot C_n \cdot R_1 = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \qquad \text{Eq.13}$$

The image plane, π_i, can also be transformed into a new plane, π_i', by using the rotation, R_1. If the camera frame is reconstructed with taking π_i' as the new image plane, it can be seen that the transformation puts the new Z'–axis through the center of the target (Figure 19). Projection of the reference circle on the new image plane generates the ellipse, C', where a and b axes become aligned with the new X', and Y' axes, the eventual result of removing the coefficients with terms in xy, xz and yz.

The arrangement of eigenvalues in the order of $\lambda_1 < \lambda_2 < \lambda_3$ means that the shorter ellipse axis is aligned with X' and the longer ellipse axis is aligned with Y'.

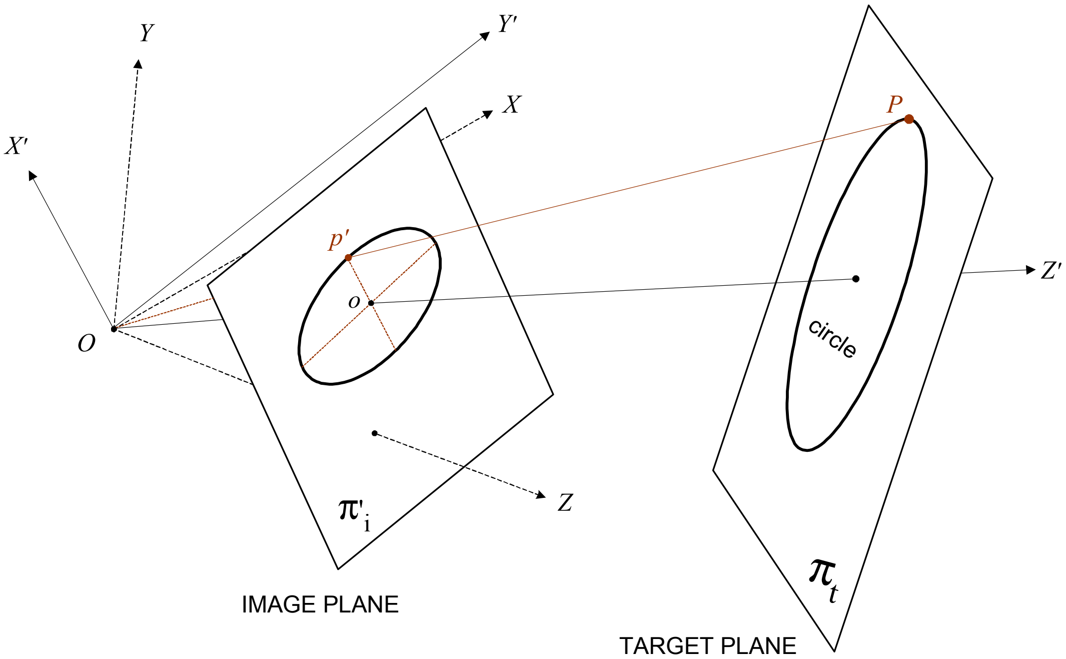


Figure 19. 3D rotation of the camera frame to the new (X', Y', Z') frame, after R_1 is applied. Please note that Z'–axis intersects the target plane with a generic angle.

The second rotation, R_2, must transform the image plane, π_i', into a new plane, π_i'', in such a way that the new plane becomes parallel with the target plane, π_t. Thereby, the projection of the reference circle on the new image plane generates a circle.

Thus, the second rotation, R_2, imposes the equality of the coefficients of x^2 and y^2 in C' in order to transform it into a circle. R_2 is achieved with a rotation around the Y'-axis by an angle θ (Eq14).

$$R_2 = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad \text{Eq.14}$$

This rotation sends a point P' to P'' (Eq15),

$$\vec{P''} = R_2^{\,T} \cdot \vec{P'} = R_2^{\,T} \cdot R_1^{\,T} \cdot \vec{P} \qquad \text{Eq.15}$$

and transforms the ellipse, C', into the circle, C'' (Eq16).

$$C'' = R_2^{\,T} \cdot C' \cdot R_2 = \begin{bmatrix} \sin^2\theta.\lambda_3 + \cos^2\theta.\lambda_1 & 0 & \sin\theta.\cos\theta.(\lambda_1 - \lambda_3) \\ 0 & \lambda_2 & 0 \\ \sin\theta.\cos\theta.(\lambda_1 - \lambda_3) & 0 & \sin^2\theta.\lambda_1 + \cos^2\theta.\lambda_3 \end{bmatrix} \qquad \text{Eq.16}$$

The value of the rotation angle, θ, is obtained by imposing the equality for the coefficients of x^2 and y^2 in Eq16. Towards this end, the matrix element, C''_{11}, that defines the x^2 coefficient is set to the y^2 coefficient, λ_2 (Eq17).

$$\sin^2\theta.\lambda_3 + \cos^2\theta.\lambda_1 = \lambda_2 \qquad \text{Eq.17}$$

$$\sin^2\theta.\lambda_3 + (1-\sin^2\theta).\lambda_1 = \lambda_2 \;\Rightarrow\; \sin^2\theta = \frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_1}$$

$$(1-\cos^2\theta).\lambda_3 + \cos^2\theta.\lambda_1 = \lambda_2 \;\Rightarrow\; \cos^2\theta = \frac{\lambda_3 - \lambda_2}{\lambda_3 - \lambda_1} \quad \text{thus,}$$

$$\theta = \pm\arctan\sqrt{\frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_2}} \qquad \text{Eq.18}$$

The composite rotation, R_C, which is the result of multiplying R_1 and R_2, transforms the image plane, π_i, so that it becomes parallel to the target plane, π_t (Figure 20). Consequently, a vector normal to the target plane can be obtained by applying Eq19. Vector $\vec{n}$ represents the orientation of the plane π_t expressed in the camera coordinate system (-1 in Eq19 allows for the right-handed coordinate system).

$$\vec{n} = R_C \cdot [0 \quad 0 \quad -1]^T = -\left[R_{C_{13}} \quad R_{C_{23}} \quad R_{C_{33}}\right]^T \qquad \text{Eq.19}$$

As a result of Eq18, there is a two-fold ambiguity in the recovered orientation depending on which sign of θ is chosen. Section 5.3.2.4 explains the geometric implications of this result, and the way proposed to break this ambiguity.

5.3.2.2 Extracting the Tag Position

C'' represents the real symmetric matrix of a circle in the image plane of radius, r_0, and center, $(x_0, 0, 1)$, in terms of the coordinate system, (X'', Y'', Z''), as illustrated in Figure 20. Eq16 can, therefore, be simplified making the terms x^2 and y^2 equal to λ_2, and then dividing by λ_2, as depicted in Eq20:

$$C'' = \begin{bmatrix} \lambda_2 & 0 & \Delta \\ 0 & \lambda_2 & 0 \\ \Delta & 0 & \Phi \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta/\lambda_2 \\ 0 & 1 & 0 \\ \Delta/\lambda_2 & 0 & \Phi/\lambda_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & 0 \\ -x_0 & 0 & x_0^{\,2} - r_0^{\,2} \end{bmatrix} \qquad \text{Eq.20}$$

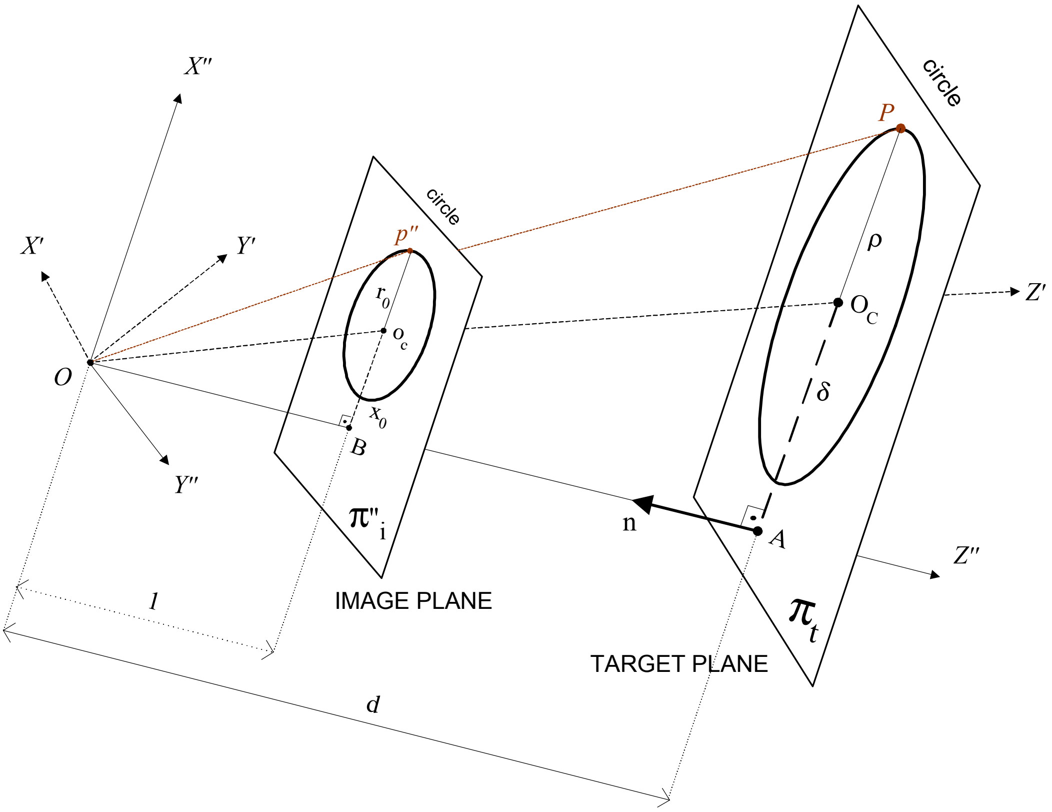


Figure 20. 3D rotation of the camera frame to the new (X'', Y'', Z'') frame after R_2 is applied. The figure also gives the geometric relations between planes π_i'' and π_t, where $|O_C A| = \delta$, $|PO_C| = \rho$, $|o_c B| = x_0$, and $|p'' o_c| = r_0$.

Equating correspondences between Eq16, Eq17, and the first equality in Eq20, it can be derived that:

$$\Phi = \sin^2 \theta.\lambda_1 + \cos^2 \theta.\lambda_3 = \lambda_1 - \lambda_2 + \lambda_3 \quad \text{Eq.21}$$

$$\Delta^2 = (\sin\theta.\cos\theta.(\lambda_1 - \lambda_3))^2 = (\lambda_2 - \lambda_1).(\lambda_3 - \lambda_2) \quad \text{Eq.22}$$

If the same process is now applied to the correspondences in the last two equalities of Eq20, the x_0 value for the coordinate X'', and the r_0 radius of the imaged circle, C'', are obtained:

$$x_0^2 = \frac{\Delta^2}{\lambda^2} = \frac{(\lambda_2 - \lambda_1).(\lambda_3 - \lambda_2)}{\lambda_2^2} \quad \text{Eq.23}$$

$$r_0^2 = x_0^2 - \frac{\Phi}{\lambda_2} = \frac{-\lambda_3.\lambda_1}{\lambda_2^2} \quad \text{Eq.24}$$

As mentioned within camera calibration (section 5.3.2.1.1), all distances are defined in focal-length units, therefore, the distance, $|OB|$, between the focus and image plane in Figure 20 equals to 1. When the equations, Eq23 and Eq24, are applied on the triangle-similarity constructed by the triangles OO_CP and Oo_cp'', an expression for δ in terms of the eigenvalues of C_n and the known radius of the reference circle, ρ, can be derived (Eq25). The fixed radius of the reference circle is defined in metric units. This reveals the depth information, and enables the acquisition of δ also in metric units.

$$\frac{\rho}{r_0} = \frac{\delta}{x_0} \Rightarrow \delta = \sqrt{\frac{-(\lambda_2 - \lambda_1).(\lambda_3 - \lambda_2)}{\lambda_1.\lambda_3}}.\rho \qquad \text{Eq.25}$$

The distance between the camera and the target plane, denoted by d, can be obtained by applying triangle-similarity principle to the triangles OAO_C and OBo_c.

$$\frac{d}{1} = \frac{\delta}{x_0} \qquad \text{Eq.26}$$

Similarly, substituting Eq26 in Eq25, an expression for the distance of the target plane to the camera origin, d, is obtained (Eq27).

$$d = \frac{\rho}{r_0} = \sqrt{\frac{-{\lambda_2}^2}{\lambda_1.\lambda_3}}.\rho \qquad \text{Eq.27}$$

The 3D coordinates of the center of the target (Figure 20), expressed in the (X'', Y'', Z'') frame, correspond to the translation vector, $\vec{T}$, which can be calculated in terms of the original coordinate system, (X, Y, Z), as:

$$\vec{T} = R_C \cdot [\delta \quad 0 \quad d]^T \qquad \text{Eq.28}$$

However, as a consequence of Eq18, the ambiguity still remains. There are two possible solutions for the translation vector, i.e., $\vec{T_1}$ and $\vec{T_2}$, depending on which sign of θ is chosen.

5.3.2.3 Extracting the Rotation Angle

Disregarding the ambiguity, the use of a circle has provided a closed form solution to the determination of the 3D pose of a tagged object. Unfortunately, due to the circle symmetry, the view of a planar circle does not permit the determination of rotations around the Z–axis of the target coordinate system, orthogonal to the target plane. No matter how much a circle is rotated around the Z–axis, its projected image looks the same. This means that from the rotation matrix, R_C, only the angles (α, β) around the axes X and Y can be obtained. Therefore, it is necessary to calculate a new rotation matrix, R'_C, from which the angle γ around the Z–axis can also be recovered.

Any rotation matrix can be expressed in terms of unit vectors. Towards this end, the vector columns of the matrix, R'_C, are established with the three unit vectors defining the target-centered coordinate system. Among these vectors, the one that corresponds to the target's Z–axis is already known ($\vec{n}$). In order to calculate the other two column vectors, namely $\vec{r}_x$ and $\vec{r}_y$, it is necessary to use two correspondences between points expressed in the target-frame, and their projections in the image plane (Figure 21). The first point correspondence is given by the known center of the target, $\overrightarrow{OO_C}$ (i.e., $\vec{T}$), and its projection in the image, $\overrightarrow{Oo_c}$. For the second correspondence, the back-projection of the synchronization point, denoted by a small star in Figure 16, is utilized. In Figure 21, this point and its back-projection on target plane are named as x_1 and X_1 respectively. Thus, the second correspondence is given by the unknown synchronization point vector, $\overrightarrow{OX_1}$, and its projection in the image, $\overrightarrow{Ox_1}$. This point can be uniquely identified in every projection of a tag. The following computations are applied to calculate this correspondence.

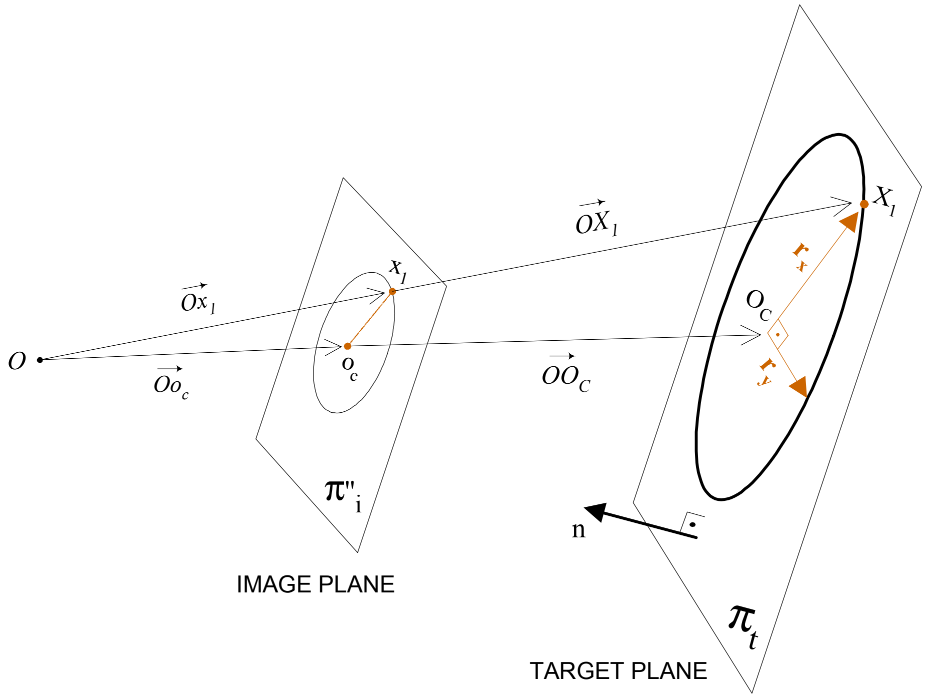


Figure 21. The unit vectors on the target plane.

Given a vector $\vec{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$, any other vector with the same direction and origin is given in homogenous coordinates by Eq29, where s indicates a free scale factor applied to the vector's modulus:

$$\vec{P(s)} = \begin{bmatrix} \vec{p} \\ s \end{bmatrix} \qquad \text{Eq.29}$$

Therefore, the correspondence between the 3D point X_1 and its projection in the image x_1 are given by:

$$\vec{OX_1} = \begin{bmatrix} \vec{Ox_1} \\ s_c \end{bmatrix} = \begin{bmatrix} x_{1x} \\ x_{1y} \\ x_{1z} \equiv 1 \\ s_c \end{bmatrix} \qquad \text{Eq.30}$$

The scale factor, s_c, can be determined by considering that the point X_1 belongs to the target plane π_t, i.e.,

$$\vec{P(s)}^T \cdot \pi_t = 0 \qquad \text{Eq.31}$$

where π_t is given by the *point normal plane* equation (Farin & Hansford 1997). This equation states that given a point p, and a normalized vector $\vec{n}$, i.e., $\|n\| = 1$, bound to p, a plane is defined by the locus of all points x that satisfy the equation:

$$\begin{aligned} \vec{n}.(\vec{x} - \vec{p}) = 0 \quad &\Rightarrow \quad n_1 x + n_2 y + n_3 z - (n_1 p_1 + n_2 p_2 + n_3 p_3) = 0 \\ &\Rightarrow \quad Ax + By + Cz + D = 0 \end{aligned} \qquad \text{Eq.32}$$

where, $A = n_1$, $B = n_2$, $C = n_3$, and $D = -(n_1 p_1 + n_2 p_2 + n_3 p_3)$. It can be proven that $|D|$ reflects the distance of the plane to the coordinate origin. Therefore, based on Eq32, the equation of plane π_t can be represented in homogenous form as:

$$\pi_t = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{n} \\ -d \end{bmatrix} = 0 \qquad \text{Eq.33}$$

where $\vec{n}$ represents the previously calculated orientation of the target plane with regard to the camera, and d represents the calculated distance from the camera center-of-projection to the target plane.

Thus, substituting Eq33 into Eq31, and using the known coordinates of the synchronization point (x_{1_x}, x_{1_y}), the value of s_c is:

$$s_c = \frac{\vec{Ox_1} \cdot \vec{n}}{d} \qquad \text{Eq.34}$$

Finally, the Cartesian coordinates of X_1, expressed in the original coordinate system, (X, Y, Z), are calculated by combining Eq34 and Eq30:

$$\vec{OX_1} = \begin{bmatrix} x_{1x}/s_c \\ x_{1y}/s_c \\ (x_{1z} \equiv 1)/s_c \end{bmatrix} \qquad \text{Eq.35}$$

The first column of the rotation matrix, $\vec{r_x}$, can hence be calculated as the unitary vector joining $\vec{OO_C}$ and $\vec{OX_1}$ in Figure 21, i.e.,

$$\vec{r_x} = \frac{(OX_1 - OO_C)}{\|OX_1 - OO_C\|} \qquad \text{Eq.36}$$

The second column of the rotation matrix, $\vec{r_y}$, can be estimated by considering that the columns of a rotation matrix, or the unitary vectors defining a coordinate system, are orthogonal. Therefore, a third orthogonal vector can be obtained by taking the cross product of the other two:

$$\vec{r_y} = \vec{n} \times \vec{r_x} \qquad \text{Eq.37}$$

Thus, the rotation matrix defining the transformation between (X, Y, Z) and (X_t, Y_t, Z_t), finally, is (Figure 17):

$$R'_C = \begin{bmatrix} \vec{r_x} & \vec{r_y} & \vec{n} \end{bmatrix} \qquad \text{Eq.38}$$

The matrix R'_C is a 3D rotation resulting from the composition of three consecutive rotations, namely R_z, R_y and R_x, around the coordinate axes by angles γ, β and α respectively. The form of R'_C is given by:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.39}$$

$$R'_C = R_x \cdot R_y \cdot R_z \qquad \text{Eq.40}$$

$$R'_C = \begin{bmatrix} \cos\beta.\cos\gamma & -\cos\beta.\sin\gamma & \sin\beta \\ \sin\alpha.\sin\beta.\cos\gamma + \cos\alpha.\sin\gamma & -\sin\alpha.\sin\beta.\sin\gamma + \cos\alpha.\cos\gamma & -\sin\alpha.\cos\beta \\ -\cos\alpha.\sin\beta.\cos\gamma + \sin\alpha.\sin\gamma & \cos\alpha.\sin\beta.\sin\gamma + \sin\alpha.\cos\gamma & \cos\alpha.\cos\beta \end{bmatrix} \qquad \text{Eq.41}$$

Given Eq38 and Eq41, the angles α, β and γ can be obtained. The angle β, to be visible by the camera, must be between $[-\pi/2, \pi/2]$, and can be calculated directly from $R'_{C_{13}}$, as:

$$\beta = \arcsin\left(R'_{C_{13}}\right) \qquad \text{Eq.42}$$

The angle α can also be directly obtained from the third column (i.e., $\vec{n}$) of R'_C. Again, to be visible, this angle is constrained to the values in range $[-\pi/2, \pi/2]$:

$$\alpha = \arcsin\left(\frac{R'_{C_{23}}}{\cos\beta}\right) \qquad \text{Eq.43}$$

Finally, the angle γ is calculated. However, there is no visibility constraint for this angle, therefore, two equations set with $R'_{C_{12}}$ and $R'_{C_{11}}$ are used in order to remove the uncertainty generated by arc function:

$$\begin{aligned} \gamma_1 &= \arcsin\left(\frac{-R'_{C_{12}}}{\cos\beta}\right) \\ \gamma_2 &= \arccos\left(\frac{-R'_{C_{11}}}{\cos\beta}\right) \end{aligned} \qquad \text{Eq.44}$$

If both $\cos\gamma_1$ and $\cos\gamma_2$ have the same sign, then γ is set to γ_1, otherwise γ is set to $\pi - \gamma_2$.

Eventually, the angles α, β and γ determine the orientation of a tag with respect to the viewing camera. However, it should not be forgotten that the ambiguity remains, since there are two possible solutions for the target-plane normal vector, $\vec{n}$. The following section describes the method developed to break this ambiguity that generated solution pairs for the translation vector, $\vec{T}$, and rotation matrix, $R = R'_C$, so far.

5.3.2.4 Breaking the Ambiguity

As noted in the previous two sections, the pose extraction method returns two feasible solutions. This result can be explained geometrically by observing Figure 22. The circles with the same radius, illustrated in Figure 22a and Figure 22b, are projected to the same ellipse in the image plane. Consequently, it is necessary to use additional information, apart from the reference ellipse, in order to be able to decide which of the two solutions is the real one. In the original TRIP system, this additional information is provided by the projection of the synchronization point. The projection of this point is identified in the code deciphering stage of the target recognition process (section 5.3.1.6). Likewise, the location of the synchronization point in the target coordinate system is also known, since the tags are generated with a fixed structure (Figure 16). The projections of this point, obtained by means of the two rigid body transformations returned by the "pose extraction" process, are compared with the known projection of this point in the image. The calculated projection lying closer to the known projection in the image shall correspond to the right solution.

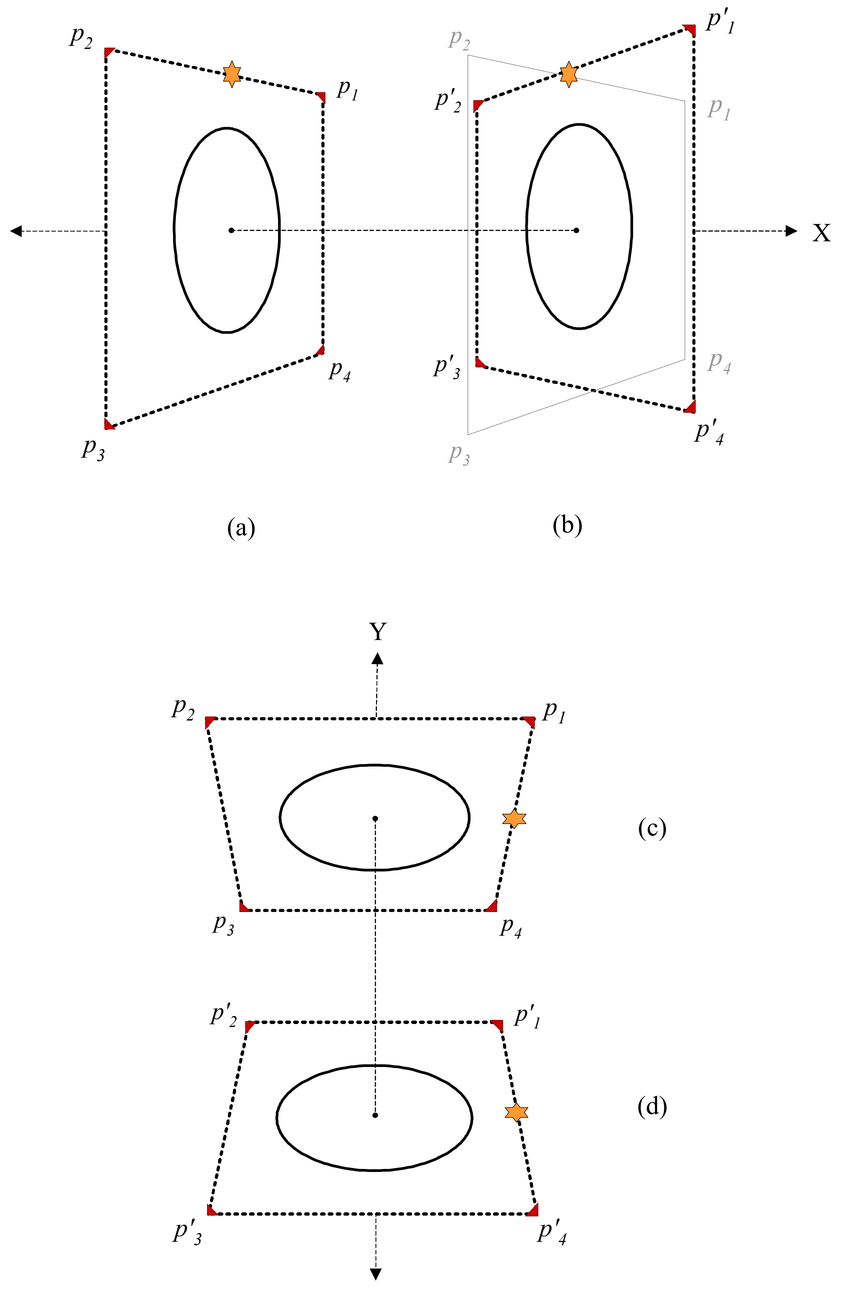


Figure 22. Circle projection ambiguity.
(a) and (b) depict the two different tag orientations that generate the same ellipse when projected on the image plane. (c) and (d) depict the same situation arisen along *Y*-axis.

However, the synchronization point alone may not be sufficient to break the ambiguity. In Figure 22, some possible tag orientations are given, where the associated synchronization points are illustrated with stars. When the two rigid body transformations returned by the “pose extraction” process are applied on the known target-centered synchronization point, identical results will be obtained as depicted in Figure 22a and Figure 22b. Same situation arisen along a different axis is also demonstrated in Figure 22c and Figure 22d. The calculated synchronization point projections will lie on the same location in the image. There may be very small differences in the numerical values of the pixel coordinates, but these differences are not comparable, especially for the tags that reside far from the camera, since the pixel sampling error of the cameras are already beyond these very small values. Therefore, it is not possible to judge whether these very small numerical differences are generated by the calculated projections or the camera errors.

Moreover, in the TRIP system, the position of the projected synchronization sector used as the reference value for comparison is determined in the code deciphering stage. As described in section 5.3.1.6, this determination is performed with the help of the reference ellipse parameters, thus the reference value used for the comparison inherits the ambiguity.

Towards the removal of these drawbacks, a more robust constraint is employed with fitting four *benchmark points* on the tag structure. When the projection of these points are calculated with the solutions returned by the “pose extraction”, at least two points out of four are projected to different positions in the image. The projections of these points for the controversial orientations are illustrated in Figure 22b, where $P=(p_1,p_2,p_3,p_4)$ represent the projections for the first solution, and $P'=(p'_1,p'_2,p'_3,p'_4)$ for the second.

As illustrated in Figure 23, the extreme points of the four gussets located around the tag are utilized for benchmark points, $P_t=(e_{t_1},e_{t_2},e_{t_3},e_{t_4})$. The target-centered coordinates of these points are known with the fixed tag structure as also depicted in the figure. The locations of these points are obtained by using a method independent from the reference ellipse parameters. Even though the tag is scanned around the ellipse, the point locations are not directly obtained by the ellipse parameters as done in the original method, but the tag image is scanned around the angles $t=\theta_i$, where $\theta_i=(2i-1).\pi/4$ for $i=1...4$. These angle ranges correspond to the corners where the gussets are located (Figure 24). The extreme points are obtained by scanning the corners in the binary image (Figure 11b) with respect to the calculated t angle values. The black-white transition at the outermost border gives out location of the extreme points, namely projected benchmark points, $P = (e_1, e_2, e_3, e_4)$.

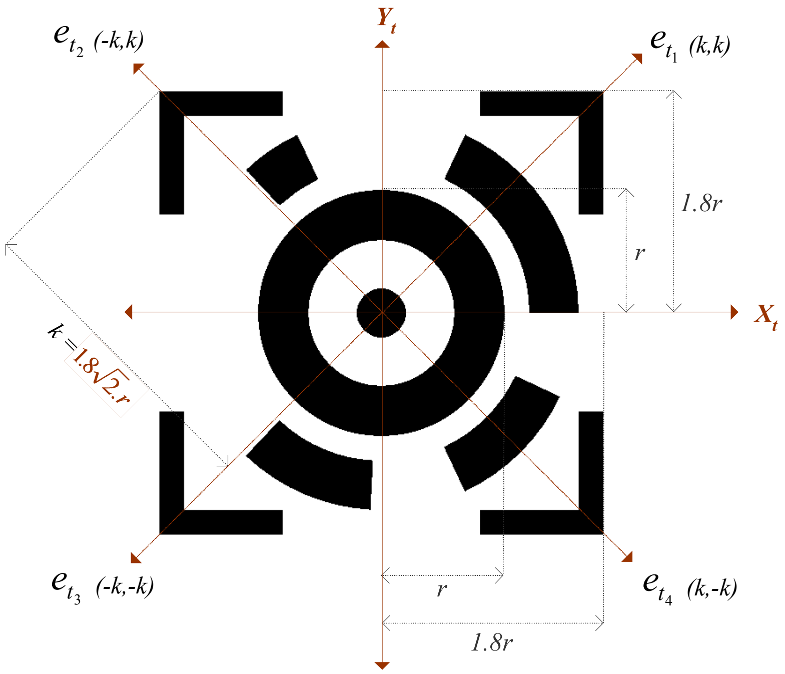


Figure 23. Modified tag structure designed to break the ambiguity. Benchmark points, $P_t = (e_{t_1}, e_{t_2}, e_{t_3}, e_{t_4})$, are depicted on the target-centered coordinate system.

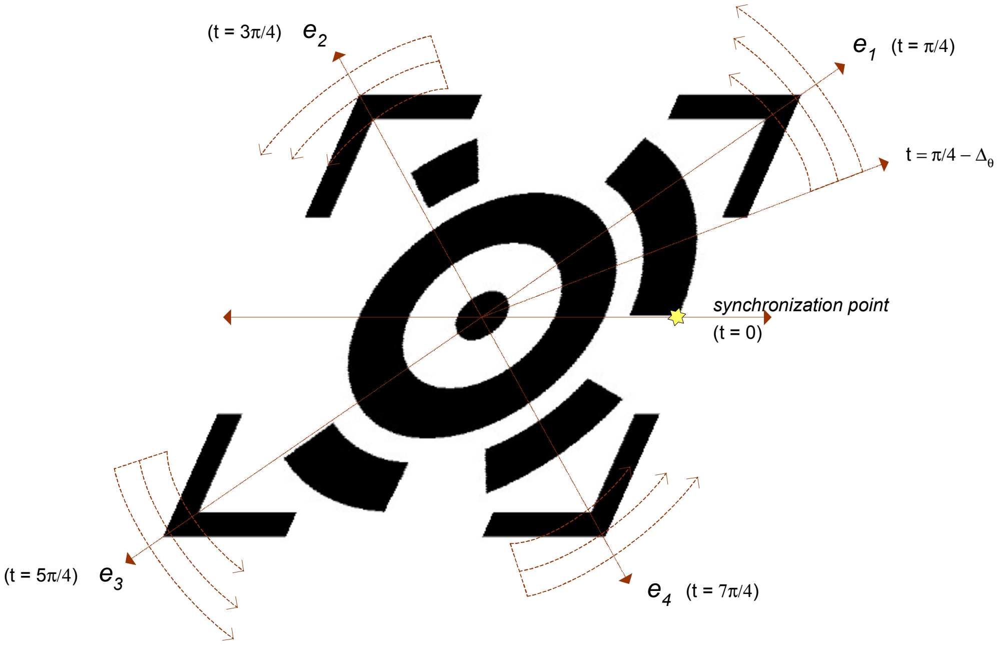


Figure 24. Extracting the location of projected benchmark points, $P = (e_1, e_2, e_3, e_4)$, in the binary image.

After the determination of the projected benchmark point locations that will be further used for comparison, the two rigid body transformations, $R_1, \vec{T_1}$ and $R_2, \vec{T_2}$ returned by the "pose extraction" process are applied on the known target-centered gusset points, $P_t = (e_{t_1}, e_{t_2}, e_{t_3}, e_{t_4})$. The projection of these points from the target plane into points $P_C = (p_1, p_2, p_3, p_4)$ and $P'_C = (p'_1, p'_2, p'_3, p'_4)$ in the image plane can be calculated by applying the planar projective transformation (Faugeras 1993) for both of the solutions:

$$\vec{P_C} = M \cdot \left(R_1 \cdot \vec{P_t} + \vec{T_1} \right) \qquad \text{Eq.45}$$

$$\vec{P'_C} = M \cdot \left(R_2 \cdot \vec{P_t} + \vec{T_2} \right) \qquad \text{Eq.46}$$

where M is the camera-calibration matrix that performs the mapping between the pixel coordinates and the camera coordinates (Eq6). The planar projective transformation exploits the fact that every point lying on the target plane has a value of $Z = 0$. Therefore, the equations Eq45 and Eq46 can be defined for each benchmark point as:

$$\begin{bmatrix} s.u_i \\ s.v_i \\ s \end{bmatrix} = M \cdot \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{32} & T_z \end{bmatrix} . \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \qquad \text{Eq.47}$$

where (x_i, y_i), for $i = 1...4$, are the known coordinates (Figure 23) of the benchmark points on the target frame, $P_t = (e_{t_1}, e_{t_2}, e_{t_3}, e_{t_4})$. The coordinates of the calculated projection of the benchmark points are depicted as (u_i, v_i). These are the coordinates of points, $P_C = (p_1, p_2, p_3, p_4)$, calculated for $R = R_1$, $\vec{T} = \vec{T_1}$, and the coordinates of points, $P'_C = (p'_1, p'_2, p'_3, p'_4)$, calculated for $R = R_2$, $\vec{T} = \vec{T_2}$.

In the final stage, the calculated projections, P_C and P'_C, are compared with the known projection of points (Figure 24) in the image, $P = (e_1, e_2, e_3, e_4)$. The right solution eventually projects the calculated results into the points lying very close to P:

$$R, \vec{T} = \begin{cases} R_1, \vec{T_1}, & if \ \|P_C - P\| < \|P'_C - P\| \\ R_2, \vec{T_2}, & if \ \|P'_C - P\| < \|P_C - P\| \end{cases} \qquad \text{Eq.48}$$

6 ENHANCED LOCATION SENSING

As given in Figure 10, the original TRIP system divides the location sensing procedure into two phases. First is the "target recognition" phase, where the tags are detected, parameters of the reference ellipses (projection of the reference circle on the camera image) are extracted, and the identification numbers are decoded. Second is the "pose extraction" phase, where the tag locations are computed from the reference ellipse parameters. After the implementation of these modules, it has been observed that the method provides a closed form solution towards the identification of the objects and determination of their locations in space.

On the other hand, when the network cameras are employed, the system achieves a performance below the requirements. The TRIP system is originally implemented on images captured by digital cameras that provide uncompressed, high quality data. However, working on raw images is not applicable in distributed environments such as buildings. Towards this end, as mentioned in section 4.5, network cameras (netcams) are selected as sensor devices, where the images can be transported in wide areas through HTTP. Like other network devices, netcams are designed to convey data as fast as possible, and therefore apply compression to images prior to transmission. This generates smoothed input images, and causes tag regions to lose their sharp details. In addition to such artifacts, netcams produce relatively lower resolution images. An increase in the camera-tag distance reduces the pixel resolution of the tag regions below the required level, and makes the identification codes harder to decipher, even though the tags are detected and reference ellipses are extracted properly.

Towards the compensation of these drawbacks, the original method is enhanced with the employment of two additional phases as depicted in Figure 25. Firstly, "adaptive sharpening" phase is undertaken, where the images blurred by the netcam effect are sharpened prior to being processed in target recognition. This implementation provides a beforehand augmentation in the image quality.

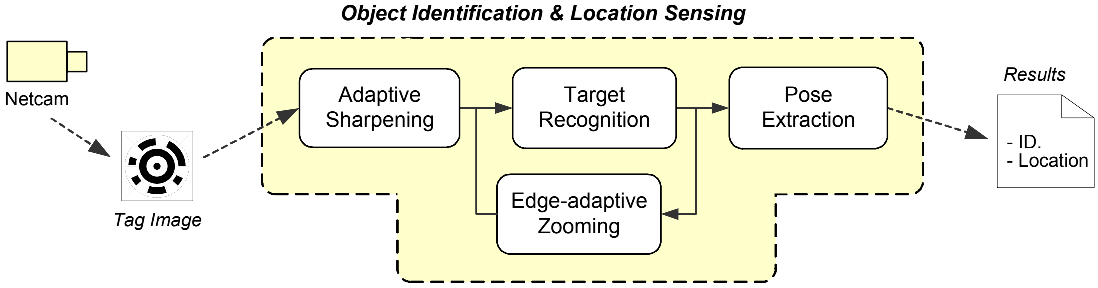


Figure 25. Enhanced object identification and location sensing.

After the images are processed in target recognition, the “edge-adaptive zooming” phase is executed. In this phase, a zooming procedure is applied locally to the *spurious* tags from which the code could not be deciphered or validated. Edge-adaptive zooming is repeated in coordination with the target recognition phase, until the tags are validated. However, this iteration becomes ineffective after a while, since the zoomed image regions lose their details for further processing. Such an unfortunate case indicates a false alarm situation or the presence of an unidentified tag.

6.1 Adaptive Sharpening

As mentioned above, network cameras are designed to convey data as fast as possible for consumer comfort, and therefore they apply compression on the original images prior to the transmission. In our implementation, the subject netcam is applying JPEG transformation with a 10:1 compression ratio even at the highest quality level. This process generates smoothed input images, and causes the tag regions to lose their sharp details. Additionally, some of the objects in the scene may reside out of the effective focus range and demonstrate a blurred vision.

In order to prevent these artifacts, “adaptive sharpening” algorithm (Battiato et al. 2003) is applied on the input images before being employed in target recognition. This method, first, restores the original image by *unsharp masking* process. In unsharp masking, the original image is blurred (unsharpened), and a fraction of the unsharp image is subtracted from the original. In other words, a fraction of the unsharp image masks the original. The image processed by the unsharp masking can be expressed as:

$$g(n_1,n_2) = a \cdot f(n_1,n_2) - b \cdot f_L(n_1,n_2) \qquad \text{Eq.49}$$

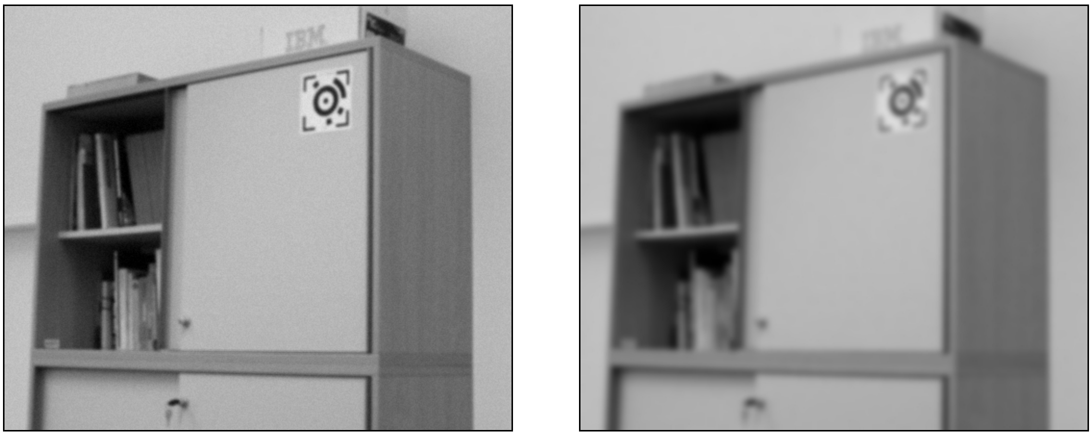
where $f(n_1,n_2)$ is the original image, $f_L(n_1,n_2)$ is the low-pass filtered or unsharp image, a and b are positive scalars with $a > b$, and $g(n_1,n_2)$ is the processed image. The parameters, n_1 and n_2, represent the pixel coordinates, where $n_1 = 0...M$ and $n_2 = 0...N$ for the image with $M \times N$ resolution.

In order to acquire the unsharpened (blurred) image, $f_L(n_1,n_2)$, the 3×3 low-pass filter, u_L, given in Eq50, is applied to the original image, $f(n_1,n_2)$, as stated in Eq51.

$$u_L = \frac{1}{4+6m} \cdot \begin{bmatrix} 1 & m & 1 \\ m & 2m & m \\ 1 & m & 1 \end{bmatrix} \qquad \text{Eq.50}$$

$$f_L(n_1,n_2) = u_L * f(n_1,n_2) \qquad \text{Eq.51}$$

The original image $f(n_1,n_2)$ is convolved by the low-pass filter with a factor of m, which determines the weight of the origin pixel over its neighborhood. The results of low-pass filtering are demonstrated in Figure 26 with a sample image.



(a) (b)

Figure 26. The results of low-pass filtering.
(a) Original image acquired from the network camera (after being transformed to gray scale. See Appendix A for details). (b) Low-pass filtered image.

An image can be considered as a combination of its low and high frequency components. Thus, the original image, $f(n_1,n_2)$, in Eq49 can be rewritten as a sum of the low-pass filtered image $f_L(n_1,n_2)$ and a high-pass filtered image $f_H(n_1,n_2)$:

$$g(n_1,n_2) = (a-b) \cdot f_L(n_1,n_2) - a \cdot f_H(n_1,n_2) \qquad \text{Eq.52}$$

From Eq52, it is clear that high frequency components are emphasized over low frequency components and that unsharp masking is some form of high-pass filtering. Since the edges or fine details of an image are the primary contributors to the high-frequency components of an image, high-pass filtering often increases the local contrast and sharpens the image. Consequently, the black-white transitions in the tag images are sharpened, enabling a proper extraction of the reference ellipses.

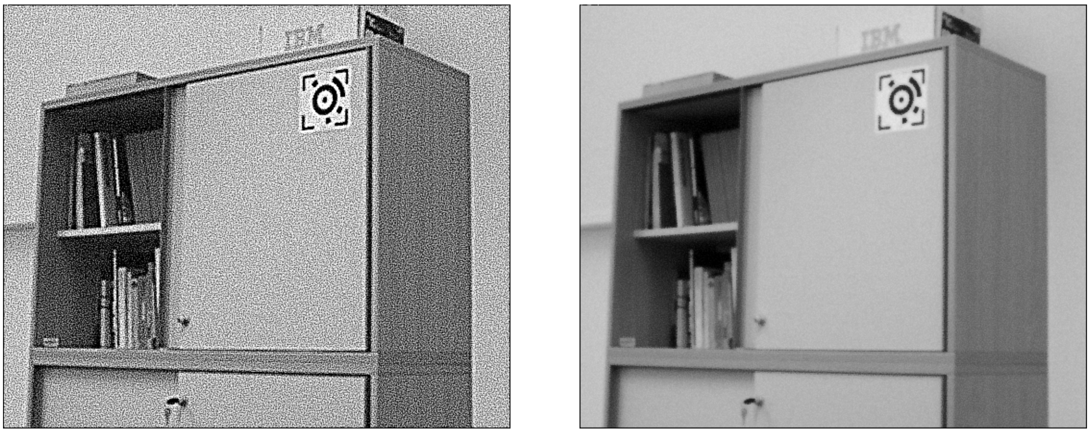
However, because a high-pass filter emphasizes the high frequency components, and background noise typically has significant high frequency components, high-pass filtering tends to increase the background noise power. Therefore, a sharpened image appears more noisy than the unprocessed image. The accentuation of background noise is a substantial limitation of unsharp masking. In our case, the network cameras expose a uniform noise with approximately 45 dB SNR (signal to noise ratio). Uniform noise generates variations in the image intensity (brightness of the pixels), where the variations are effected in accordance to a uniform distribution within a narrow intensity range. Therefore, uniform noise is visible as grain in the pixel level intensities. An example of this type of noise is depicted in Figure 26a. (Please note that the noise in this image is manually generated, and its level is exaggerated when compared with the netcam for visibility). Directly sharpening the images increases the effectiveness of this noise, and generates false pixels in the early binarization phase. The "adaptive sharpening" algorithm reduces these artifacts by combining adaptively restored image with the original one as defined in Eq53:

$$r(n_1,n_2) = k \cdot g(n_1,n_2) + (1-k) \cdot f(n_1,n_2) \qquad \text{Eq.53}$$

where $f(n_1,n_2)$ is the original image, $g(n_1,n_2)$ is the restored (sharpened) image, and $r(n_1,n_2)$ is the processed image. The parameter, k, is used as a measure of texturization that defines the intensity difference of the pixel (n_1,n_2) in the original image from its eight-connected neighborhood. As depicted in Eq53, for the pixels that expose a significant difference from its neighborhood (like edge boundaries of the tag circles), the weight of the sharpened image is emphasized over the original one. However, for the image pixels that reside on a homogenous region, the original image is taken into consideration. The effects of directly high-pass filtering, i.e., unsharp masking, and adaptively sharpening applied on the original image (Figure 26a) are given in Figure 27.

6.2 Edge-Adaptive Zooming

In addition to the camera artifacts mentioned above, an increase in the tag-camera distance reduces the pixel resolution of the tag images. This complicates the deciphering of identification codes, even though the tags are detected and reference ellipses are extracted properly in the target recognition phase. As a consequence of this drawback, the identification codes are not validated with the parity check, and the tags are marked as spurious.



(a) (b)

Figure 27. (a) Unsharp-masked image. The sharpening process also augments the power of the noise. (b) Adaptively sharpened image.

To solve the problem, "edge-adaptive zooming" algorithm (Battiato et al. 2000) is applied locally to the spurious tags from which the identification codes could not be deciphered or validated. The major problem in zooming is to properly fill the new pixels that come to existence after doubling the image resolution. There are several algorithms, like bilinear and bicubic zooming that interpolate the intensity values of the intervening pixels. Edge-adaptive zooming, as opposed to its counterparts, fills the new pixels with considering the discontinuities and sharp luminance variations in the images. This feature of the algorithm restricts the smoothing effect of the interpolation, and preserves the enhancement performed in the adaptive sharpening phase.

Towards the implementation, the algorithm performs a gradient controlled and weighted interpolation. The method, however, does not require a pre-gradient computation, since the relevant information is collected during the zooming process. The algorithm includes four successive steps, explained as follows:

(1) The original $M \times N$ image, $f(n_1, n_2)$, is expanded to the zoomed image, $z(n_1, n_2)$, size, $(2M-1)\times(2N-1)$, where

$$f(n_1, n_2) = z(2n_1, 2n_2) \qquad \text{Eq.54}$$

After the expansion, the case in Figure 28 occurs, where the "black" pixels mark the first step enlargement. Following steps involve the process of filling the spaces marked with “green” and “blue” (pixels with at least one even coordinate), and “red” (pixels with both odd coordinates) in the figure.

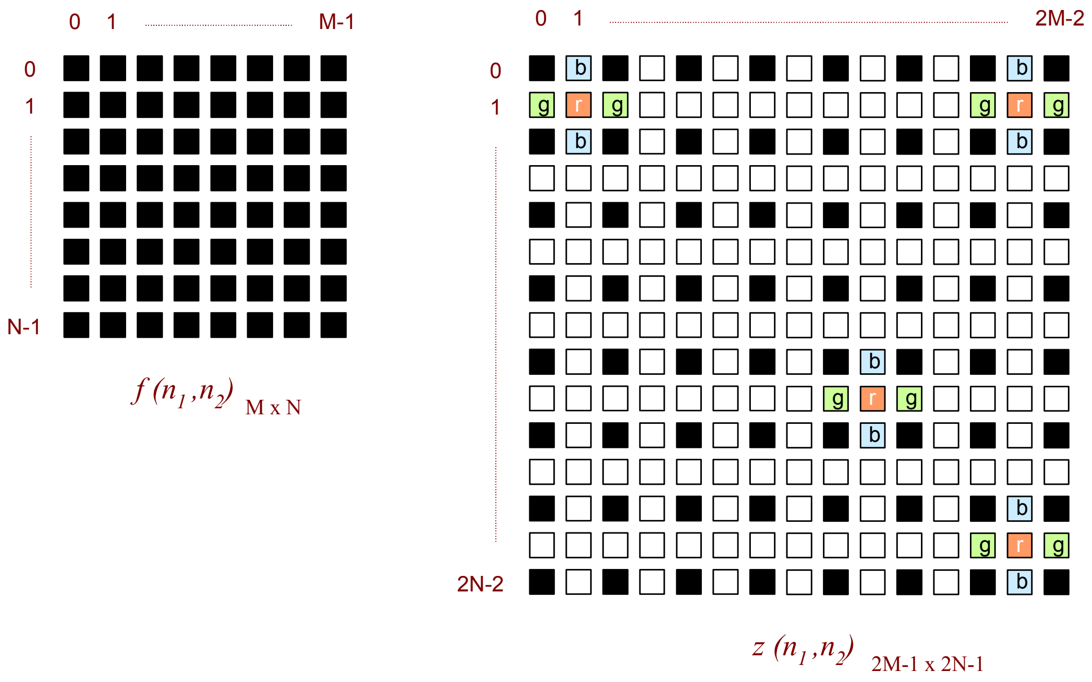


Figure 28. The expansion of the original image, $f(n_1,n_2)$, to the zoomed image, $z(n_1,n_2)$.

(2) Following steps fill the empty pixels with respect to some required conditions. In the second step, the algorithm scans only the “red” ones to examine the conditions. Towards this end, it tries to detect the presence of an edge for “red” pixels in southwest–northeast, northwest–southeast, north–south and east–west directions.

The value of the “red” pixel is assigned in accordance with the southwest–northeast and northwest–southeast edge direction if one is detected. The values of neighboring “green” and “blue” pixels are assigned in case of an edge presence in north–south (vertical) and east–west (horizontal) directions respectively. The assignments are performed with averaging the known values of two neighboring “black” pixels that lie on the edge direction.

In case of uniformity, the overall average computed with four neighboring “black” pixels is assigned to the “red” pixel. If none of the above conditions are met (no edge, and no uniformity), the pixel is left unassigned.

(3) In the end of the previous step, there may be several “red”, “blue” and “green” pixels left unidentified. Successive scans of the zoomed image take care of these empty holes.

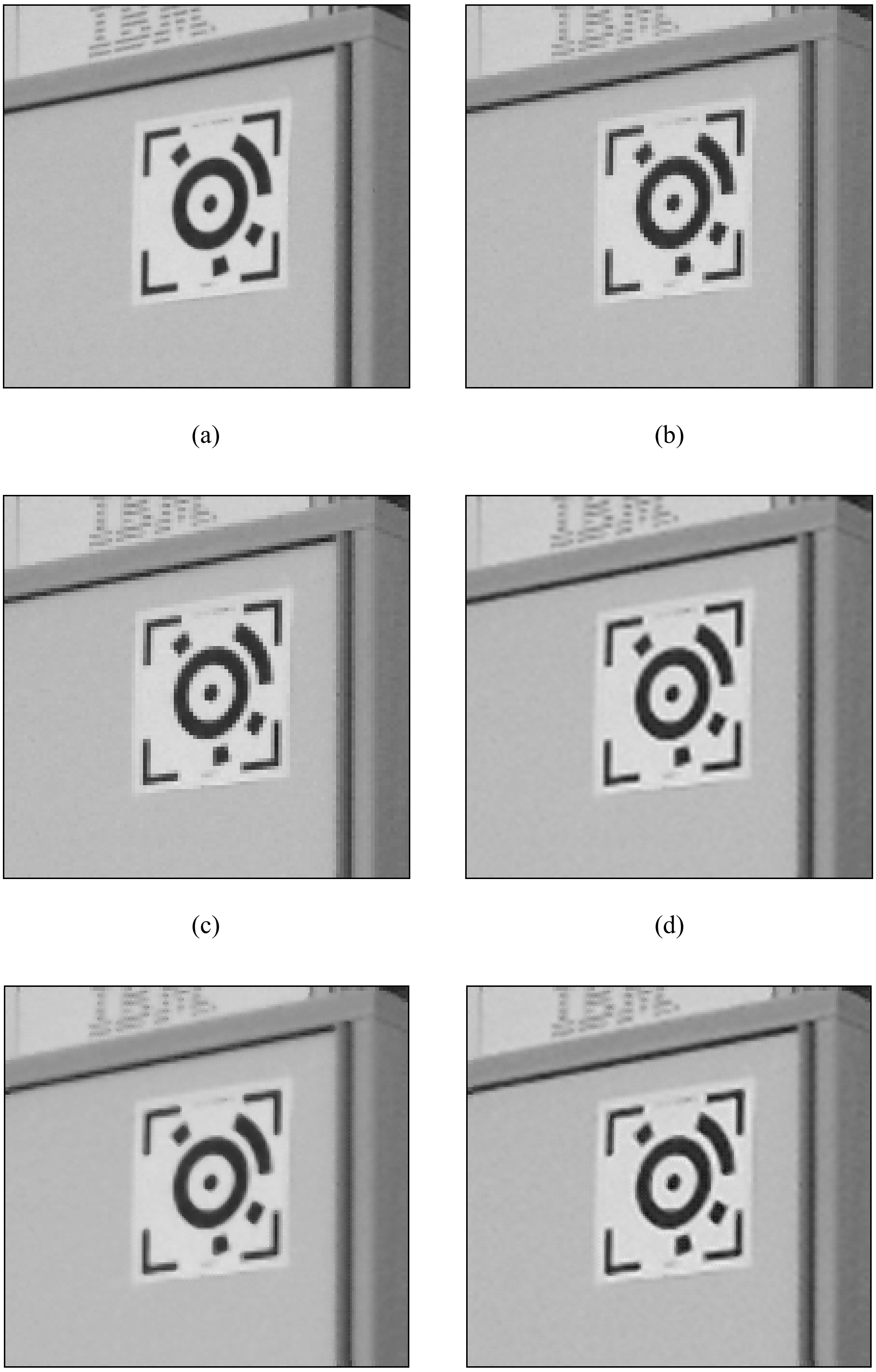
The third step scans the “blue” and “green” pixels in image, $z(n_1, n_2)$. For each scanned “blue” pixel, if the neighboring “red” pixels are assigned, the algorithm checks for the presence of a horizontal or vertical edge. And if one is detected, it assigns a proper value to the “blue” pixel in accordance with the edge direction. For vertical edges, the known values of “red” pixels are used, whereas for horizontal edges, the known values of “black” pixels are utilized in the assignment. If the neighboring “red ” pixels are not assigned, the algorithm checks only the presence of a horizontal edge, for which the assignment can be performed with using the known “black” pixels.

Same condition is also applied to the “green” pixels. However, this time, the neighboring “red” pixels are used for horizontal edges, whereas “black” pixels are used for vertical ones.

At the end of this step, all the pixels, whose spatial dependences from the neighborhood values are “simple”, are assigned to a value. Using the information collected so far, the remaining holes are filled at the next step.

(4) The final step scans the image, $z(n_1, n_2)$, looking for unidentified remaining pixels, and fixes the holes using a suitably weighted mean value. The weighted mean is computed with calculating the mean of relevant neighboring pixel values, after quantizing them to a selected m number of bins. In this case, the more frequent values are weighted less so as to guarantee that a better detail preservation is achieved in the zoomed image. For “red” pixels, the weighted mean values of the “black” neighbors are used. After the accomplishment of “red” pixels, the “green” and “blue” pixels are filled with the weighted mean values of the “black” and “red” neighbors.

The qualitative comparison of the edge-adaptive zooming with different zooming algorithms is given in Figure 29. The original image used for testing is depicted in Figure 29a before being downsampled with a ratio of 50%. The result of the first test method, zoomed image using pixel replication technique, is illustrated in Figure 29b. This is the simplest method that fills the empty pixels with directly replicating the original ones. Pixel replication is not suitable to enlarge photographic images, since it does not provide any anti-aliasing, and therefore increases the visibility of jaggies. Being hardly a technical term, *jaggies* refer to the visible steps of diagonal lines or edges in a digital image. Also referred to as *aliasing*, these steps are simply a consequence of the regular, square layout of a pixel.



(a) (b)

(c) (d)

(e) (f)

Figure 29. Results obtained with different zooming algorithms.
(a) Original image before downsampled by 50%. (b) Image zoomed with pixel replication, (c) bilinear interpolation, (d) bicubic interpolation, (e) original edge-adaptive zooming, (f) edge-adaptive zooming combined with bicubic interpolation.

The zoomed image using bilinear interpolation is illustrated in Figure 29c. Bilinear interpolation determines the value of a new pixel based on the average of the four pixels in the nearest 2×2 neighborhood. The averaging has an anti-aliasing effect and therefore produces relatively smooth edges. Bicubic interpolation, given in

Figure 29d, is more sophisticated, and produces a better result than bilinear interpolation. A new pixel is a bicubic function using 16 pixels in the nearest 4×4 neighborhood. This is the method most commonly used by image editing software and digital cameras for resampling images. The high order averaging obtained by the bicubic function takes also the derivatives of the neighboring pixels into account. This prevents the step-like boundary problem of the pixel replication, and copes with the bilinear interpolation blurring.

The result obtained with edge-adaptive zooming algorithm is finally given in Figure 29e. When compared with pixel replication and bilinear interpolation methods, it can be seen that a better anti-aliasing is provided with edge-adaptive zooming. Against the smoothing side effect of averaging, the algorithm takes into account the information about the discontinuities, while doubling the input image. However, when compared with bicubic interpolation, it can be observed that the averaging along an edge direction with only two neighboring pixels generates more smoothing. The utilization of weighted mean values in the fourth step of the algorithm provides a protection for the sharp details only when the bin number is kept very low during the quantization. However, in this case, the method produces undesired textures in homogenous regions, and generates jaggies like the bilinear interpolation method. If the bin number is selected with an acceptable value, the method does not provide a sufficient protection against smoothing. This artifact suppresses the advantage gained from the edge adaptation. Towards this end, the algorithm is combined with the bicubic interpolation method. The weighted averaging proposed in the original algorithm is replaced with the bicubic interpolation performed in the nearest 4×4 neighborhood. However, since the interpolation is applied along an edge direction, only the four pixels lying on the associated edge are utilized. All of the 16 neighbors are employed in uniformity cases, when the overall averaging is required. The interpolation kernel used in the combined method can be defined as:

$$h(t) = \begin{cases} 1-2|t|^2+|t|^3, & \text{if } 0\le|t|<1 \\ 4-8|t|+5|t|^2-|t|^3, & \text{if } 1\le|t|<2 \\ 0, & \text{otherwise} \end{cases}$$

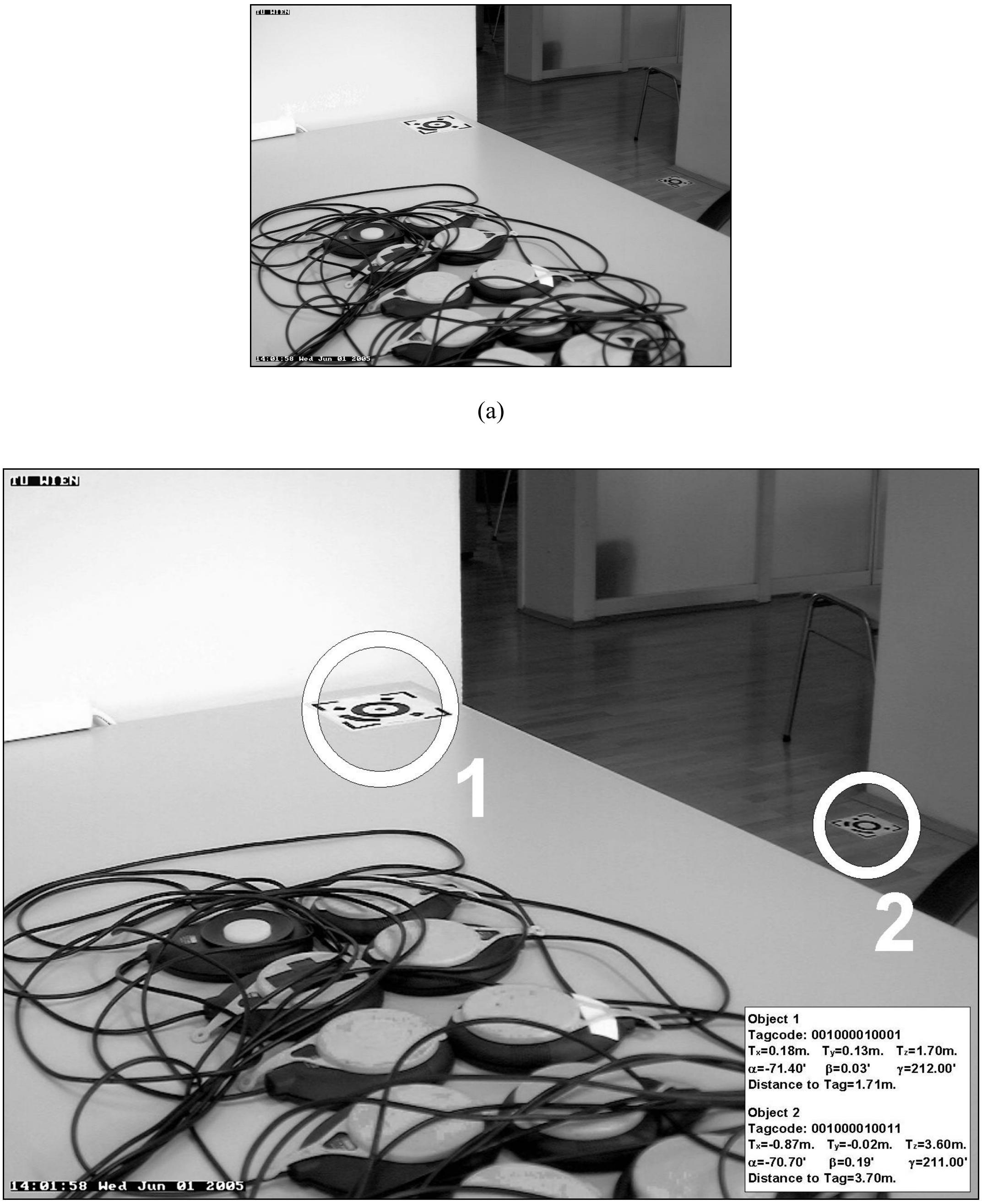
$$h(n_1,n_2) = h(n_1)\cdot h(n_2) \qquad \text{Eq.55}$$

where t represents the distance between the new pixel and its neighbors. In the kernel, the distance is given in a normalized manner, so that $t=0.5$ between two successive pixels on the image. The result of the combined solution is illustrated in Figure 29f. Even though the algorithm complexity is raised to the level of bicubic interpolation, the advantages of both methods are integrated, enabling an edge-adaptive zooming with augmented anti-aliasing, and obviated smoothing side effect.

6.3 Results of Enhanced Location Sensing

As given in Figure 25, the zoomed tag regions are re-processed in the target recognition phase with sharpened images and enlarged resolutions, enabling the identification of the tags that could not be validated in the previous attempt. After the proper identification of tag codes and extraction of reference ellipse parameters, the pose extraction phase is executed. If zooming is applied, the matrix of camera intrinsic parameters, M (Eq6), is adjusted before the execution, since the relation between the image coordinates and camera coordinates changes with respect to the zoom factor (explained in section 10.1 in more detail).

After the implementation of the enhanced location sensing system, its performance is examined with the network camera. Towards this end, a typical office environment (test-bed) is constituted with several office equipments that possess a significant role in the model generation. These equipments are used as test objects, and are tagged for the location sensing. They are placed at several distances from the camera in the range of 0.7 and 3.7 meters, and with several incidence angles (angles between the normals of the target and image plane) in the range of 0 and 72 degrees. The test-bed is scanned with a netcam and pan-tilt unit pair, and the results are recorded for the original and enhanced system. Towards a solid model generation of the test-bed, even the objects lying at the utmost distances must be identified. However, as stated in the beginning of the chapter, the original system is insufficient towards the meeting of this requirement, and performs an effective sensing only up to 3 meters distance and 60 degrees incidence angle. The tags lying beyond these limits generate problems, since their images are not either sharp or large enough for a proper identification. A sample situation is illustrated in Figure 30. In Figure 30a, the raw image acquired directly from the netcam is given. The test objects visible in this image are the table located at 1.65 meters with 72 degrees incidence angle, and the floor located at 3.69 meters with 72 degrees incidence angle. The original system is not able to identify both of the objects. On the other hand, as given in Figure 30b, the system is able to identify and locate the objects after the enhancements are applied.



```
Object 1
Tagcode: 001000010001
Tx=0.18m.   Ty=0.13m.   Tz=1.70m.
α=-71.40'    β=0.03'      γ=212.00'
Distance to Tag=1.71m.

Object 2
Tagcode: 001000010011
Tx=-0.87m.   Ty=-0.02m.   Tz=3.60m.
α=-70.70'     β=0.19'        γ=211.00'
Distance to Tag=3.70m.
```

(a)

(b)

Figure 30. Results obtained for a sample scene in the test-bed
(a) from the original system, (b) from the enhanced system. Please note that the resolutions are scaled proportionally to fit the images on the page.

Camera distances and incidence angles of the tags in the test-bed limit the performance of the original system, whereas the enhanced method is able to identify all of the test objects in the space. The location results do not expose a substantial variance when compared for the objects identified by both of the methods. Nevertheless, it has been observed that the enhanced method demonstrates a lesser deviation from the ground-truth values. The test is performed after the implementation of the entire system, and the overall execution is examined. Therefore, a detailed explanation of the test platform and the evaluation of the test results are given in chapter 11, following the chapters where the related system design and implementations are described.

7 OCCUPANCY SENSING

In addition to the object location data, the model generator unit utilized within the project (Figure 2) needs to be aware of the occupancy information in the space for a comprehensive model construction. Even though there are several products in the market (see section 3.2), it is more convenient to integrate a vision-based solution for occupancy detection by using the same infrastructure developed for location sensing rather than to implement a discrete system all over again. Occupancy sensors operate based on detection of motion, assuming that occupancy creates movement. Towards this end, a method is implemented for detecting motion from the sequential camera images. The temporal changes in the gray scale intensity (brightness) values of the pixels are evaluated to sense the motion within the camera's field-of-view. As mentioned before, the original camera images are in RGB format. The images are converted to gray scale before the implementation of this method. The details of the conversion are given in Appendix A.

Such visual motion detection methods based on temporal intensity changes are currently applied for several purposes. The visual motion can be an important source of information for surveillance systems. Objects of interest can be detected as their motion becomes apparent. Similarly, objects can be tracked on the basis of their motion defined trajectories (Wildes 1998).

In our case, the visual motion data is not utilized for extracting object information, but instead for sensing occupancy. For this reason, classifying or tracking the moving patterns are not a concern. The classification of image sequences into two particular groups, "occupied" or "not occupied", is sufficient for judgment. However, occupancy does not necessarily generate motion, and motion does not necessarily generate intensity change in image pixels. One example of this common problem is illustrated in Figure 31a. A smooth sphere rotating under constant illumination does not generate changes in the images of the shooting camera. Second problem is illustrated in Figure 31b. In contrast to the first situation, the intensity values change even though there is no object motion. Such illuminance variations in the scene may cause false occupancy results.

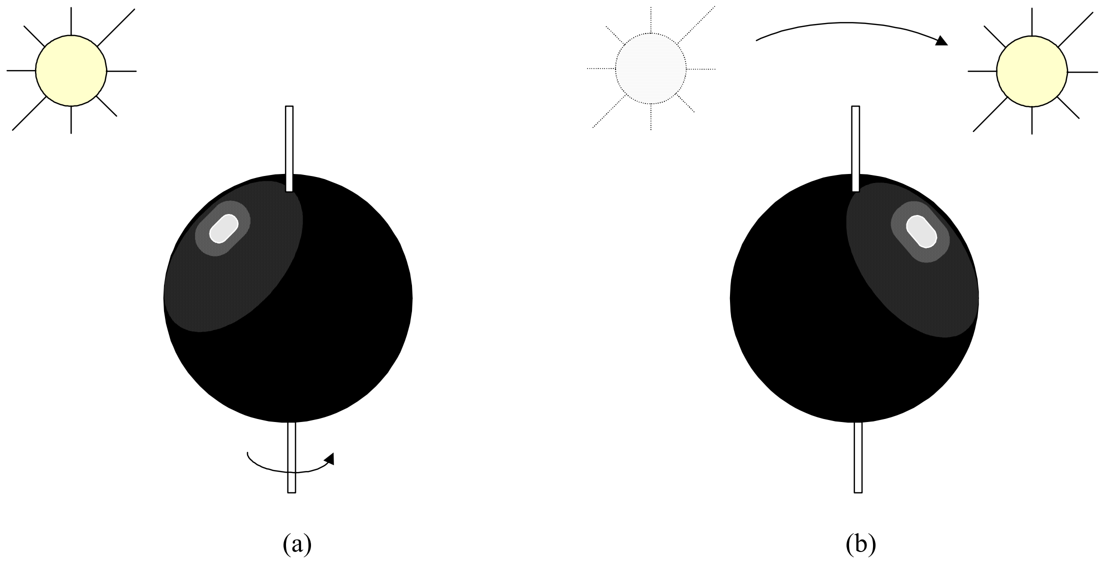


Figure 31. An example for the common motion detection problems.
(a) A smooth sphere is rotating under constant illumination, but the image does not change. (b) A fixed sphere is lighted with a varying illumination, and this causes the image to change even though there is no object motion.

7.1 Homomorphic Filtering

The case illustrated in Figure 31a represents a common problem of occupancy sensors and requires the employment of additional methods, which were not further pursued in our project. We assumed that the occupancy eventually generates motion, and this motion consequently leads to intensity changes in the image sequences.

However, it is not possible to assume such limitations for the second case, since the illumination may change within temporal images. Even though the light source does not move as fast as the illustration in Figure 31b, the light level variations of the uplights in the test-bed may generate intensity changes. Additionally, the aperture of the camera is managed automatically by the camera mechanism, and its width can vary within image sequences, which leads to similar intensity changes in the image pixels. Finally, the noise in the images produces slight variations in the pixel level, and the total sum of these variations become substantial when calculated for the overall image. In order to overcome these problems, a model utilized in Toth et al. 2000, and well known in homomorphic image filtering is used. In this model, the image intensity is considered to be generated by an incoming illumination, which is reflected by the surfaces of the objects in the observed scene. For diffuse surfaces that reflect light equally in all directions, the relation between observed intensity, y, illumination, i, and reflectance, r, is multiplicative. The intensity of the τ^{th} frame in an image sequence can be modeled as:

$$y_\tau(n_1, n_2) = i_\tau(n_1, n_2) \cdot r_\tau(n_1, n_2) \qquad \text{Eq.56}$$

with n_1, n_2 being the pixel index. The verification of this model proposed for diffuse surfaces can be found in Horn 1986. Most surfaces in the built environment are diffuse reflectors, but transparent elements such as glass, or shiny objects such as smooth metallic surface cause specular reflection. The model is not accurate in the case of such objects, but this rarely affects the occupancy detection effectiveness.

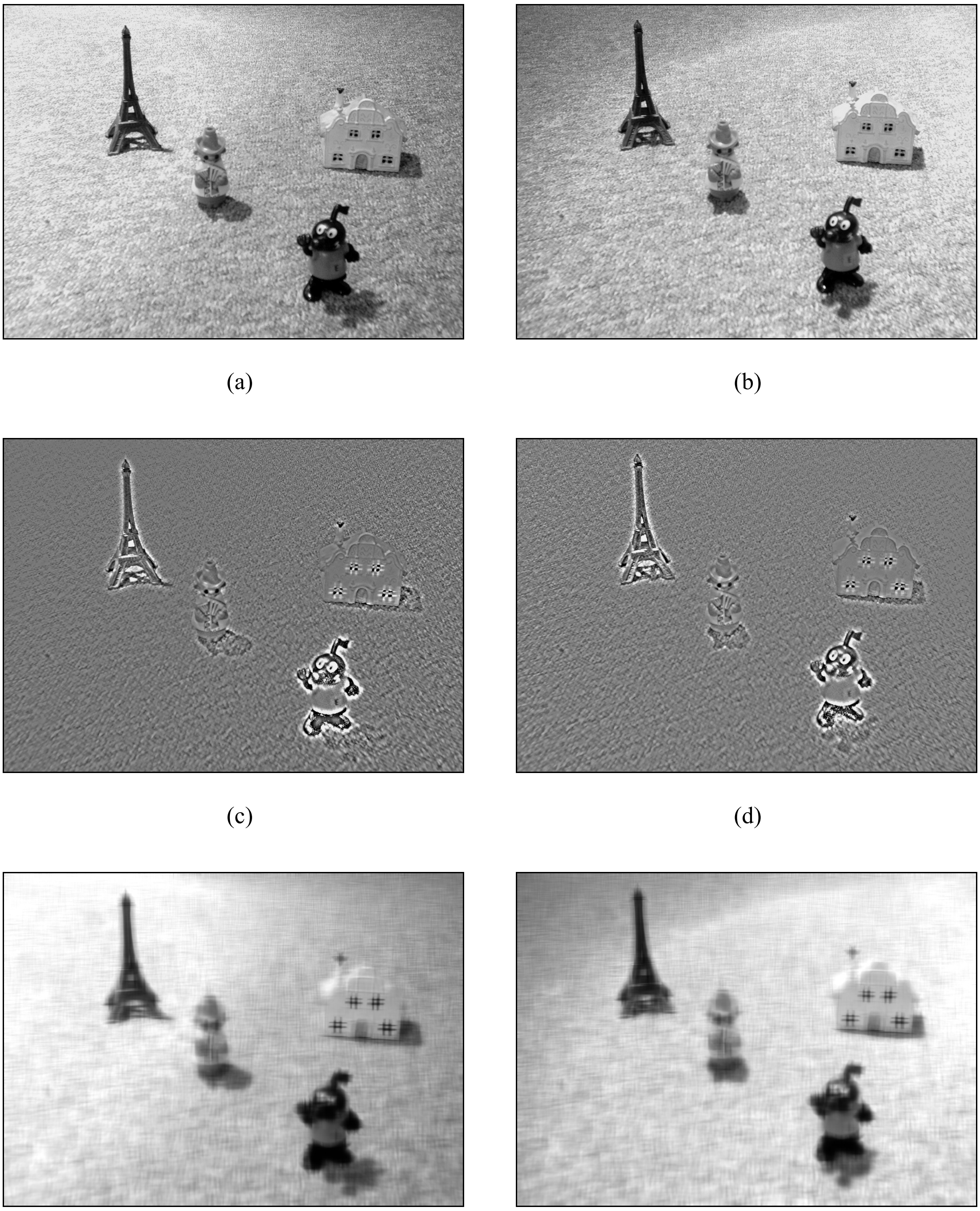
In many realistic cases, the scene illumination, $i(n_1, n_2)$ can be assumed to be spatially slow-varying. These slow-varying regions correspond to the low frequency components of the image. On the other hand, the reflectance map, $r(n_1, n_2)$, contains medium and high frequency details, where the effects of the illumination are suppressed. These details typically correspond to object information on the image. Therefore, it is more convenient to utilize the reflectance maps rather than the intensity images for evaluating the pixel changes in the video sequences.

In Figure 32, an example is demonstrated with a scene illuminated by a changing light source. The scene is firstly illuminated by a spot of light from the left side as depicted in Figure 32a, and then from the right side as depicted in Figure 32b. It can be observed that the reflectance maps of the images, r (given in Figure 32c and Figure 32d respectively), are identical, since the effect of the varying illumination is strongly suppressed, while the object information is preserved. In the illumination images, i (given in Figure 32e and Figure 32f), however, the light spot is very prominent whereas object details are blurred.

Since the reflectance maps provide a robust determination of occupancy, the raw camera images must be decomposed into their illuminance and reflectance components. Towards the decomposition of such multiplied signals, homomorphic filtering is utilized (Figure 33). First, the logarithm is applied to transform the multiplicative relation between y, i and r (Eq56) into an additive one, i.e.,

$$\log(y_\tau(n_1, n_2)) = \log(i_\tau(n_1, n_2)) + \log(r_\tau(n_1, n_2)) \qquad \text{Eq.57}$$

After applying the logarithm, the image is low-pass filtered using a binomial filter-kernel (with $size = 21$ for the sample images in Figure 32). Although the log-nonlinearity modifies the spectral content of illumination and reflectance components, it is in practice often justified that the log-illumination is still spatially slow varying. Therefore, the log-illumination still corresponds to the low frequency component, and can be revealed by low-pass filtering. Subtraction of the low-pass filtered output from the logarithmic original yields to the extraction of the high-pass component. Exponentiation of both high-pass and low-pass components eventually separates the image into its illuminance and reflectance maps.



(a) (b)

(c) (d)

(e) (f)

Figure 32. Decomposition of the images into illumination and reflectance components. (a) Image illuminated from the left side by a spot-light. (b) Same image illuminated from the right side. Reflectance components are given in (c) and (d) respectively. The effects of illuminance are suppressed in both of the images. Illumination components are given in (e) and (f). This time, the light-spot is very prominent in both of the images.

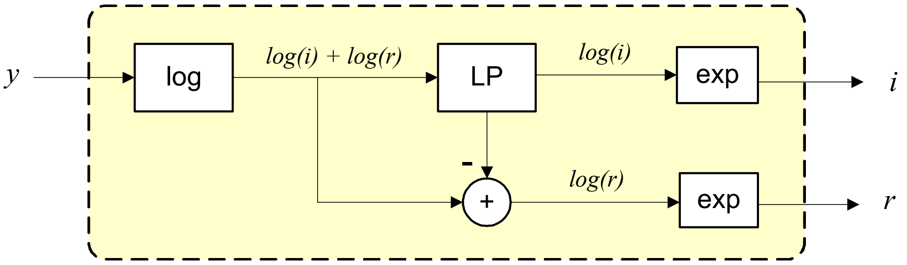


Figure 33. Homomorphic filter for multiplied signals.

7.2 Illumination-invariant Change Detection

After the extraction of reflectance components, the temporal video image sequences are processed in order to determine the presence of occupancy in the scene. Towards this end, the detector mechanism shown in Figure 34 is applied to the reflectance maps sequentially.

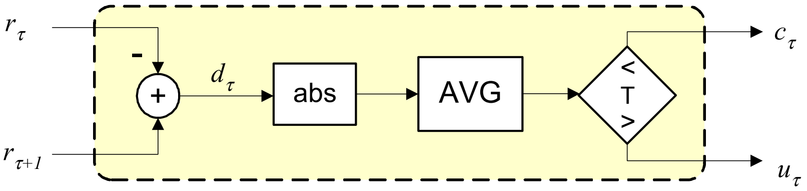
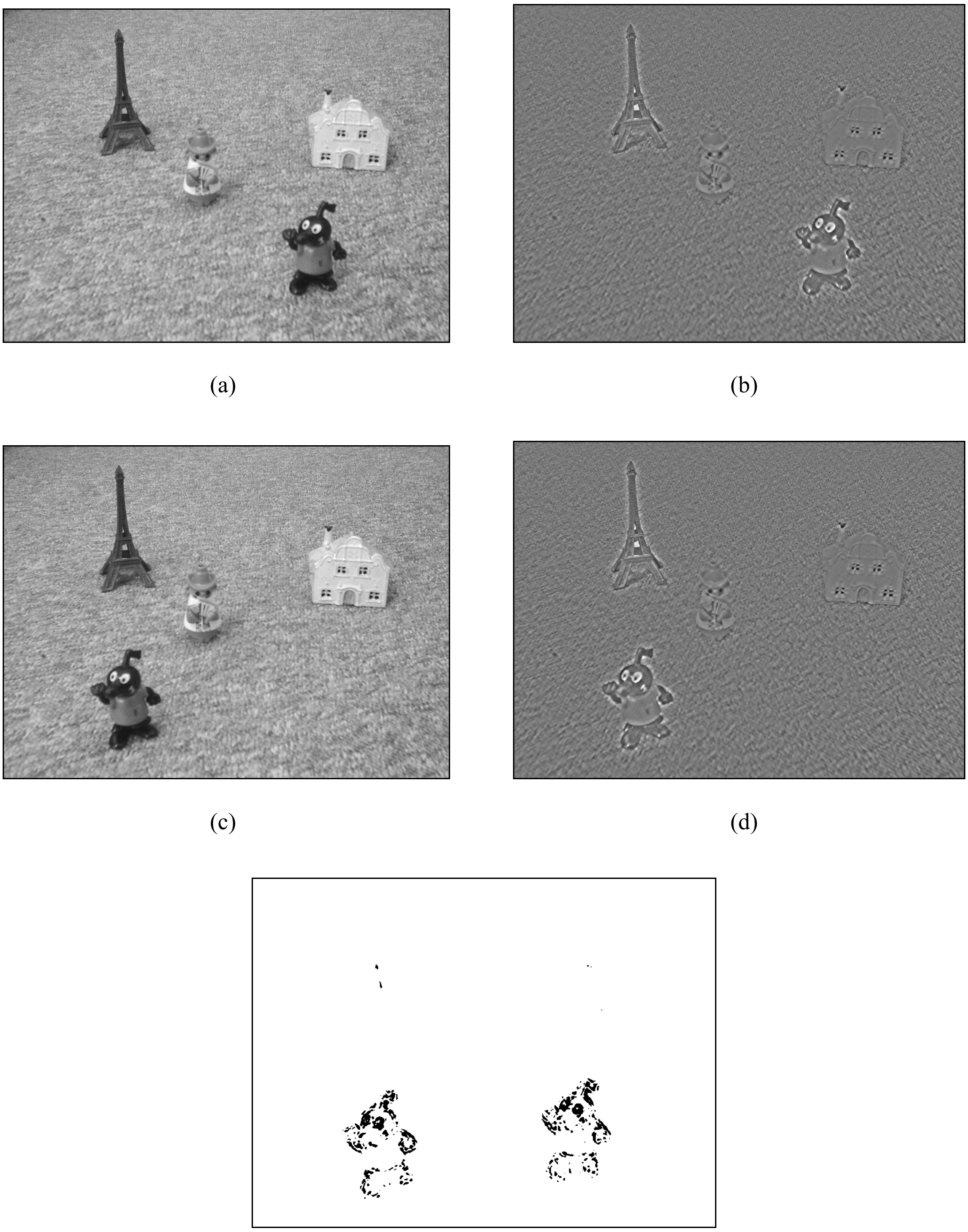


Figure 34. Change detection in sequential reflectance components.

The reflectance of the τ^{th} frame is subtracted from the $\tau+1^{\text{th}}$ frame, and the absolute value of the difference, d_τ, is calculated so as to provide the change between frames in pixel level. These pixel-based changes are, however, not sufficient to directly judge for occupancy, since the underlying reason for the change may be a noise in that particular image location. Therefore, the change values are averaged in a 3×3 neighborhood. The averaged results are compared with a threshold, and the ones that exceed this threshold are marked as real object change, c_τ, and the others as unchanged, u_τ. Eventually, the particular groups of changed values determine the presence of occupancy. An example is demonstrated in Figure 35, where two sequential video shots are given in Figure 35a and Figure 35c respectively. The cartoon cat figure in the first image moves to the left. The extracted reflectance components, shown in Figure 35b and Figure 35d, are processed with the detector mechanism described above. After the processing, the pixels denoted as object change, c_τ, are marked on a new image illustrated in Figure 35e. The moving object, namely the cat figure, in the image is strongly salient. The marks on the right side demonstrate a change, since the cat is not present any more, and its shape is replaced with the background. On the other hand, the marks on the right side demonstrate the change generated by the new cat shape replacing the background in that region.



(a)

(b)

(c)

(d)

(e)

Figure 35. The results of the change detector.
(a) τ^{th} image in the video sequence, (b) and its reflectance component. (c) $\tau+1^{\text{th}}$ image in the video sequence, (d) and its reflectance component. (e) The demonstration of object changes, c_{τ}.

8 VIOLAS CONCEPTUAL DESIGN

Our primary goal is to collect visual data from the sensors, and extract the object information (identification and location of the objects together with the occupancy data in the space) required by the lighting control system. So far, the sensors and algorithms developed and utilized for this purpose are described in the thesis. However, to achieve this goal, and to adapt the outcome applications to the built environment, a more comprehensive scheme is required. This scheme must manage the data flow between the applications, generate accurate and consistent sensing information, and provide the transmission of these results to the lighting control system. Towards this end, a conceptual scheme is designed with isolated blocks, each of which performs a different and designated task. We named the resulting system that wraps this scheme as VIOLAS, vision-based object location and occupancy sensing.

This section describes the conceptual process flow in VIOLAS. The functional blocks given in the design scheme execute a specific algorithm or a bunch of algorithms to accomplish their purposes. Please note that, the realization of these blocks may reside on different applications, therefore, the following explanations do not give information about the software implementation, but only the designed process flow.

8.1 Hardware Interface

Hardware interface is the first initiated unit that isolates the software parts of VIOLAS from the hardware devices. Thus, the effect of any change in the hardware to the overall system is minimized. Hardware interface, briefly, performs the communication with the cameras. In case of the existence of a pan-tilt unit, it also performs the motion of these devices. So, the purpose of the hardware interface is concisely to (1) set the connection with the network camera, (2) set the necessary camera adjustments (resolution, compression rate...etc), (3) acquire the images, and convert these raw images into an understandable format (Appendix A) for further processing, (4) control the pan-tilt unit, if one is attached to the camera.

For each camera, several parameters must be known for communication and image processing. These required parameters are retrieved from a camera table defined in the VIOLAS database to harbor such sensor data (Figure 36). Parameters stored for each camera are: (1) an ID number that uniquely defines the camera in the system (2) details for a HTTP connection, i.e., IP address and the communication port of the camera (see section 4.4 for netcams), (3) camera model, (4) camera intrinsic parameters (see section 5.3.2.1.1 for details), (5) camera's location in the room, (6) horizontal and vertical FOV (7) pan-tilt unit availability, (8) model of the pan-tilt unit, if one is available, (9) pan range, (10) tilt range, (11) pan angle, (12) tilt angle, (13) camera's on-the-fly status, (14) camera's location on the pan-tilt unit, (15) status, (16) last-activity time.

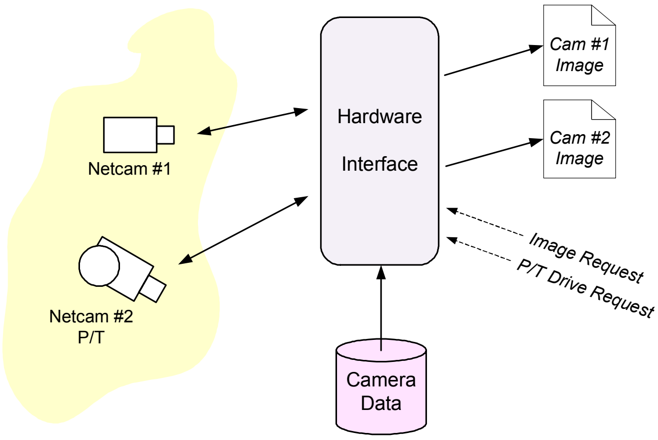


Figure 36. Hardware interface.

The model of the camera is stored in the database, since some models apply custom image-compression methods that require the utilization of a plug-in component in the image acquisition software. In order to avoid implementing different programs for different cameras, the acquisition methods are collaborated in the hardware interface, and the corresponding one is applied based on the camera model. Similarly, the pan-tilt unit models are also collaborated, as each model implements a proprietary protocol in the data link layer of the communication (see section 4.6.1.2 for details about pan-tilt units). Based on the selected model, the corresponding communication protocol is undertaken. The FOV values and pan-tilt ranges retrieved from the camera table are also utilized during the control of pan-tilt device motions.

The outputs of the hardware interface are the images. The ID of the camera is attached on the images together with the camera's calibration and location values. If a pan-tilt unit is available for that camera, the hardware interface also attaches the information of camera's location on the pan-tilt unit, and the pan-tilt angle positions from which the image is taken. These parameters are further used during sensing activities. The status and last-activity-time parameters are utilized by VIOLAS for tracking the available cameras in the system. The detailed usage of these parameters are described in the following sections.

8.2 Sensing Core

The different sensing activities of the system are collaborated under the sensing core unit. The purpose of the sensing core is to process the collected images, and extract the (1) identification, (2) location, and (3) occupancy information in the scenes as described in the previous sections (Figure 37). The sensing core also conveys the camera parameters attached on the incoming images to the output results. Therefore, the following units that process the results are able to be aware of the information about the camera from which the results are generated.

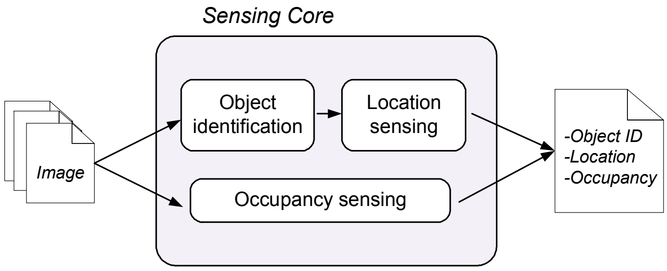


Figure 37. Sensing core.

8.3 Coordinate Transformation

Outcome of the sensing core regarding the location data is the position and orientation information with respect to the coordinates of the camera from which the processed image is acquired (Eq48). In this state, the location data is not usable for a model generation. By using 3D transformations, coordinate transformation converts the position and orientation data with respect to camera coordinates to a feasible form: the position and orientation data with respect to the real-world coordinates, i.e., room coordinates (Figure 38). In order to perform such a transformation, the camera’s location in the room must be utilized (Figure 39a). In other words, the translation and rotation values between the two coordinate systems must be known (see Appendix C for 3D transformations). Towards this end, the location of each camera inside the relevant room is stored in the camera table, and conveyed to the coordinate transformation unit as described in the previous sections.

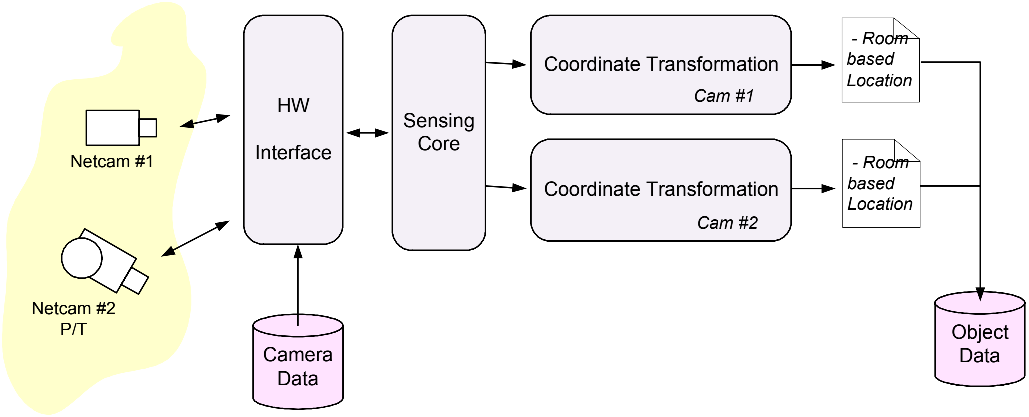
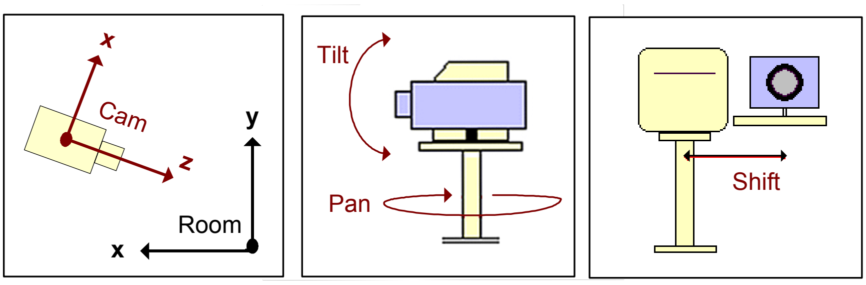


Figure 38. Coordinate transformation.

Thus, the location parameters attached on the input data uniquely identifies the transformations between each of the camera reference frames (camera coordinate systems) and the world reference frame (room coordinate system). For describing the relative positions of the origins of the two reference frames, a 3D translation vector, $\vec{T}$, is used. An orthogonal 3×3 rotation matrix, R, aligns the corresponding axes of the two frames. In a common notation, the relation between the coordinates of a point in room and camera frame, P_{room} and P_{cam} respectively, is

$$P_{room} = R \cdot P_{cam} + \vec{T} \qquad \text{Eq.58}$$

where translation is defined in room coordinates, and rotation is defined from room to camera. As a general notation, the 3D transformation, i.e., translation and rotation, that brings the room reference frame onto the camera, performs the coordinate transformation from camera to room reference frames.



(a) (b) (c)

Figure 39. Parameters considered in the coordinate transformations.
(a) Cameras' locations inside the room are important to define the transformations. (b) If P/T unit is involved, pan and tilt angles must also be considered. Most P/T devices are manufactured with built-in potentiometers that provide the angle values. (c) Position of the camera on the P/T unit is not negligible.

If the camera is attached on a pan-tilt unit, two sequential rotations must be applied. First rotation, R_1, is defined from room to P/T, where P/T device is assumed to take the place of the camera with its original position ($pan = 0°$, $tilt = 0°$). Second rotation, R_2, is defined from P/T to camera, where pan and tilt angles are involved (Figure 39b). Pan rotation of these devices defines the motion capability around their y-axis, and tilt rotation defines the motion capability around their x-axis. Pan-tilt units do not provide any motion capability around the z-axis. The $tilt$ angle rotation is implemented first in order to get the desired total rotation that overlaps the P/T and camera reference frames. This order does not match the general notation used so far: $\gamma \rightarrow \beta \rightarrow \alpha$. This mismatch occurs because of the angle notation difference of the pan-tilt units. Namely, pan angle does not correspond to β, and $tilt$ angle does not correspond to α. The provided $tilt$ angle is not the geometric value, but the relative angle adjustment applied to move the netcam vertically. Therefore, the sequential implementation of these angles in the order of $tilt \rightarrow pan$ provides R_2, the rotation from P/T to camera.

Eventually, the two rotations are given respectively as:

$$\begin{aligned} R_1 &= R_{Room \rightarrow P/T} \\ R_2 &= R_{P/T \rightarrow Camera} \end{aligned} \qquad \text{Eq.59}$$

If we assume that the camera is mounted on top of the pan-tilt unit, there is no additional translation other than the translation of the P/T device, $\vec{T}$. Therefore the equation becomes:

$$P_{room} = (R_1 \cdot R_2) \cdot P_{cam} + \vec{T} \qquad \text{Eq.60}$$

However, practically it is impossible to place the camera right at the top of the pan-tilt unit: there is always a shift from the origin of the P/T coordinate system (Figure 39c). This shift, $\vec{S}$, is also stored in the camera table, and conveyed to this unit as a part of the camera parameters. It is defined in the pan-tilt unit's coordinate system, and must be transformed into the room specific values, $\vec{S}'$, in order to be added in the final equation:

$$P_{room} = R_1 \cdot R_2 \cdot \left(P_{cam} + \vec{S} \right) + \vec{T} \qquad \text{Eq.61}$$

$$P_{room} = R_1 \cdot R_2 \cdot P_{cam} + \vec{T} + \vec{S}' \qquad \text{Eq.62}$$

Eventually, the total rotation, $R_{total} = R_1 \cdot R_2$, and the total translation, $\vec{T}_{total} = \vec{T} + \vec{S}'$, give the final relation between the locations of the objects in the room and camera frame.

On the other hand, the outputs of the sensing core are not point coordinates like P_{cam}, but again translation and rotation values defined from camera to tag. In other words, the outputs are the values that perform the coordinate transformation from tag reference frame to camera reference frame. So, a final transformation must be applied to acquire a total transformation between tag and room coordinate systems.

Let $R_{Cam\to Tag}$, and $\vec{T}_{Cam\to Tag}$ be the outputs of the sensing core, and P_{tag} be a point on the tag reference frame. Then:

$$P_{cam} = R_{Cam\to Tag} \cdot P_{tag} + \vec{T}_{Cam\to Tag} \qquad \text{Eq.63}$$

Assuming that a pan-tilt unit is also involved (Eq62), this makes a total coordinate transformation defined from tag to room reference frames:

$$P_{room} = R_1 \cdot R_2 \cdot \left(R_{Cam\to Tag} \cdot P_{tag} + \vec{T}_{Cam\to Tag} \right) + \vec{T} + \vec{S}' \qquad \text{Eq.64}$$

$\vec{T}_{Cam\to Tag}$ is defined in the camera coordinate system, and must be transformed into the room specific values, $\vec{T}'_{Cam\to Tag}$, in order to be added in the final equation:

$$P_{room} = R_1 \cdot R_2 \cdot R_{Cam\to Tag} \cdot P_{tag} + \vec{T} + \vec{S}' + \vec{T}'_{Cam\to Tag} \qquad \text{Eq.65}$$

The total rotation, R_{total}, and the total translation, $\vec{T}_{total}$, finally give the location of the objects in the room with respect to the tag frame:

$$\begin{aligned} R_{total} &= R_1 \cdot R_2 \cdot R_{Cam\to Tag} \\ \vec{T}_{total} &= \vec{T} + \vec{S}' + \vec{T}'_{Cam\to Tag} \end{aligned} \qquad \text{Eq.66}$$

The coordinate transformation is also applied for occupancy results, if the detecting camera is attached on a pan-tilt unit. The position of the camera is known with the provided *pan* and *tilt* angles. Together with the camera's FOV range information retrieved from the camera table, this enables the extraction of the region within which the occupancy is detected. The rotation in Eq60, $R = R_1 \cdot R_2$, provides the desired transformation. For translation values, a fixed vector, $\vec{T} = [0 \quad 0 \quad 2]$, is used, assuming that occupancy takes place in the 2 meters distance of the camera. Even though this calculation does not provide a precise location data, it enables the acquisition of a region information for the occupancy. Finally, the results (both occupancy and tag locations) are recorded in an object table together with the related camera information (Figure 38).

8.4 Data Fusion

Data fusion unit is designed to be executed continuously in certain time intervals. In these timer activations, the coordinate-transformed location data acquired from all cameras are combined by this phase. The input data are retrieved from the object table, and the fused outputs are stored in the fusion table (Figure 40). There are two phases of the data fusion, namely tag-level fusion and object-level fusion.

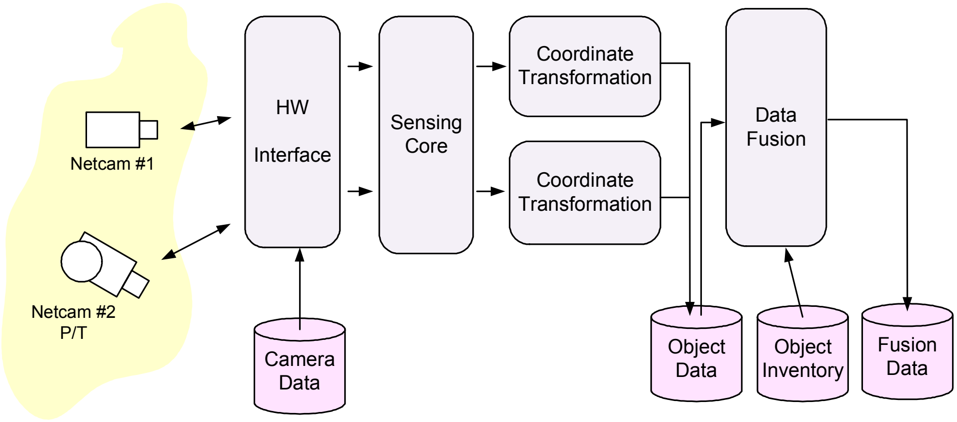


Figure 40. Data fusion.

8.4.1 Tag-Level Fusion

The same tag can be detected with more than one camera (Figure 41), or one camera assigned to multiple instances of the "sensing core" (to be discussed in chapter 9). This will eventually generate repeated tag records coming from multiple cameras (or sensing cores) in the system.

Data fusion combines these records by taking the identification time and uncertainty data into account. Most up-to-date and certain information is selected as the final, unique tag information. Time and uncertainty are assigned by the sensing core unit after the object identification. Uncertainty is generated with respect to the pose data (particularly, the parameters of the reference tag ellipse). This provides information about the accuracy of the location sensing. As distance increases (as reference ellipse gets smaller), the deviation of the location information from the real values increases as well (see chapter 11 for details). Therefore, if the identification times of the multiple records are close to each other, the one detected by the proximate camera is selected as the final result.

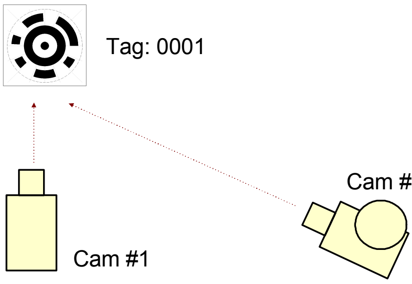


Figure 41. Tag-level fusion.

8.4.2 Object-Level Fusion

The second phase of the data fusion is implemented in the object-level. In this phase, the (fused) tag information is transformed to object information.

When a tag is created (to be explained in section 8.6), the related object information (name, description, dimensions…etc) is also entered in the object-inventory table. Therefore, the system is aware of the object information with the identified tag ID.

The system also enables the attachment of multiple tags on a larger object to reduce the occlusion possibility and to increase the line-of-sight between tags and cameras. This requires the second level fusion in order to prevent redundant object records when both tags are identified (Figure 42). As with the tag-level fusion, most up-to-date and certain information is selected as the final, unique object information.

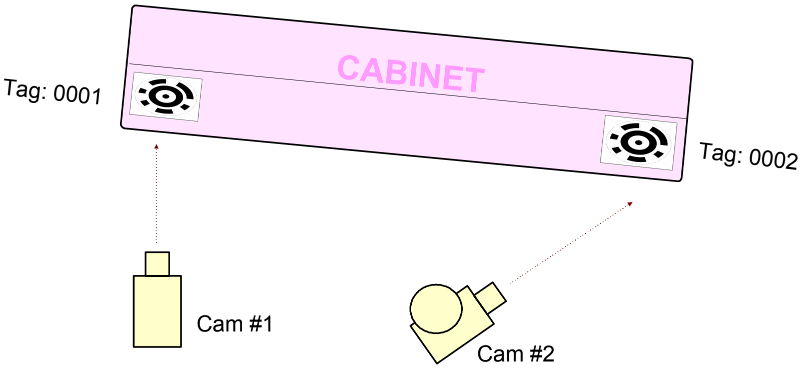


Figure 42. Object-level fusion.

In addition to location, the occupancy data is also managed in data fusion. The time stamps are attached for detected occupancies in the sensing core, as done for tag identifications. Based on the detection times, the duration of occupancies are determined. The ones that exceed the lifetime periods are removed. Finally, the output, unique location and occupancy information, is stored in the fusion table.

8.5 Communication Interface

Output of the data fusion, i.e., the final and consistent data, is transformed into XML-like data packets for convenient data communication, and transferred to the lighting control system through the communication interface (Figure 43). The communication is designed with a TCP/IP socket server implemented in the interface. It enables the connection of not only the lighting control system, but also any other third party applications that can prospectively download and process the data packets. Communication interface conveys the object data to the clients in the course of the first connection and afterwards, whenever a change occurs in the environment.

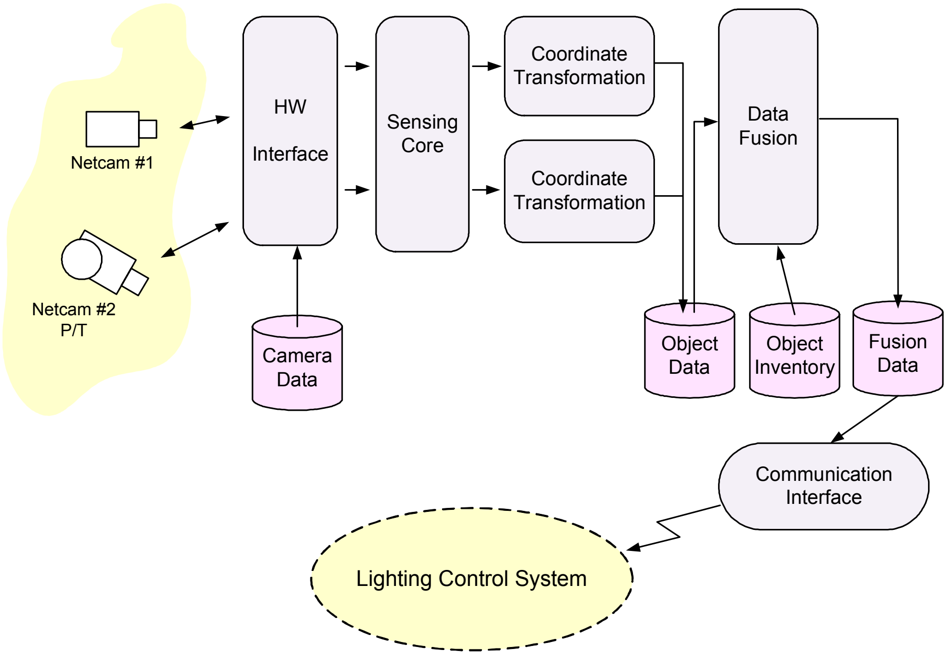


Figure 43. Communication interface.

8.6 User Interface

User interface provides the communication between VIOLAS and the operator. As mentioned above, the system needs the existence of some predefined information. This unit enables an operator to add and modify such data, and to see the system results on the screen. It is comprised of several subunits that allow the access to distinct data groups (Figure 44).

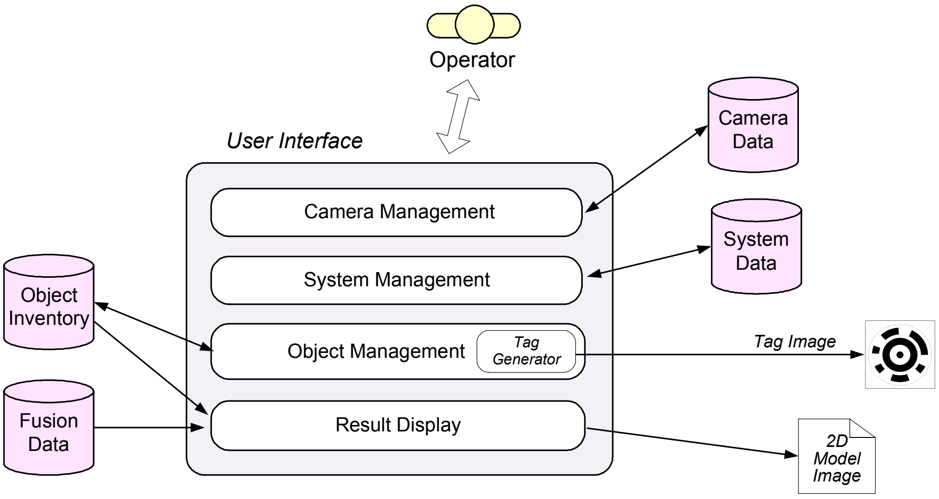


Figure 44. User interface.

Firstly, the camera management subunit allows the operator to modify the camera parameters mentioned in section 8.1. In addition, system requires certain information, like the occupancy lifetime values mentioned in section 8.4, or other values that are substantial for the execution (to be described in chapter 9). These values are stored in a system table, and the system management subunit enables the operator for their modification.

Similarly, with the object management subunit, user interface provides the modification of the object inventory. The operator can introduce new objects to the system, and for each object, he/she can enter related information, i.e., (1) the object's name, (2) description, (3) dimensions of the object, and (4) parameters of the tag that will be attached on the object.

As mentioned above, the system allows the attachment of multiple tags on a single object, but each tag code is created uniquely and automatically by the system. The tag generator program, designed as a part of the object management, produces a new code number for each tag demand, creates the tag image with respect to this automatically supplied tag code, and finally, enables the image to be printed on a printer device.

Figure 45 illustrates the hierarchy of the records stored in the object inventory. Each object data possesses its proprietary parameters as mentioned above, and additionally the parameters of the tags that are attached on the object. For each tag, (1) the code number, i.e., ID number, and (2) its location on the object is defined. The object dimension and tag location values are utilized in the next subunit, result display.

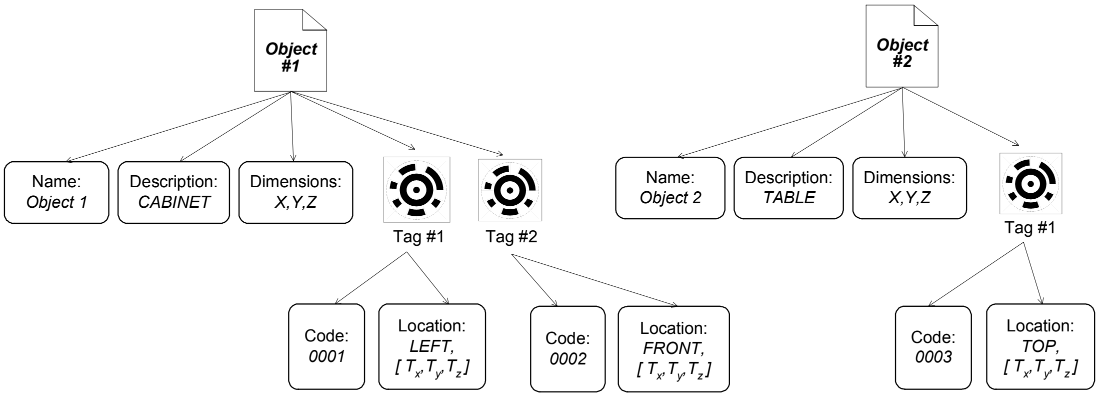


Figure 45. Object and tag hierarchy in the object inventory.

The result display subunit combines the sensed location values retrieved from the fusion table with the values retrieved from the object inventory, and generates the 2D graphical representations of the objects inside the room. Towards this end, the system assumes every object as a minimum bounding-box covering the original object shape (Figure 46). So, each object can be represented by eight points, $P_1, P_2, P_3, \ldots, P_8$, with the following corresponding coordinates defined on the object's local coordinate system:

$$\begin{aligned}
P_1 &= [0 \quad 0 \quad 0] \\
P_2 &= [X \quad 0 \quad 0] \\
P_3 &= [X \quad 0 \quad Z] \\
P_4 &= [0 \quad 0 \quad Z] \\
P_5 &= [0 \quad Y \quad 0] \\
P_6 &= [X \quad Y \quad 0] \\
P_7 &= [X \quad Y \quad Z] \\
P_8 &= [0 \quad Y \quad Z]
\end{aligned} \qquad \text{Eq.67}$$

where X, Y, Z mark the dimensions of the object in the order of width, height and depth as shown in Figure 46. The results stored in the fusion table are the final total rotation, R_{total}, and total translation, $\vec{T}_{total}$, values explained in Eq66. These values transform the point coordinates defined with the tag coordinate system to the point coordinates defined with the room coordinate system. However, this transformation cannot be directly applied to the bounding-box point locations, $P_1, P_2, P_3, \ldots, P_8$, since they are defined with respect to the object's local frame. Therefore, these point coordinates must firstly be redefined with respect to the tag reference frame.

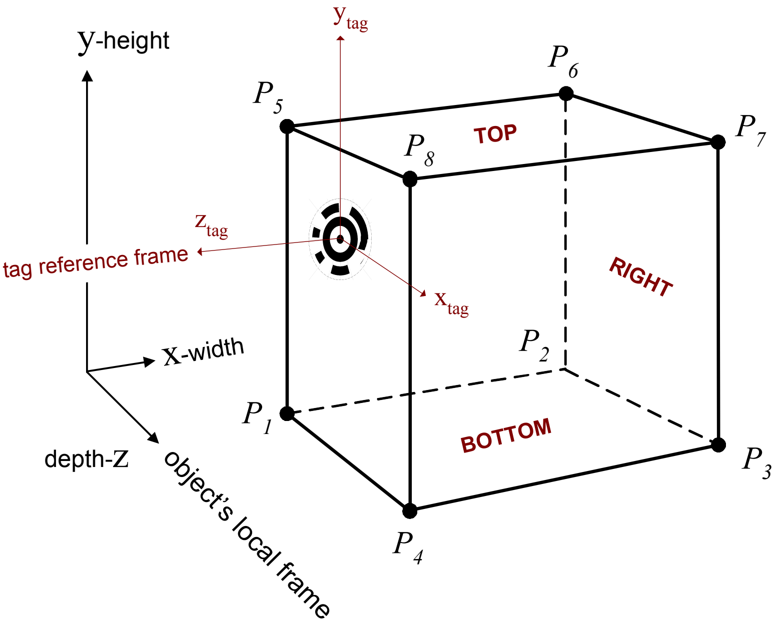


Figure 46. Object and tag reference frames.

Towards this end, the transformation parameters of each tag, i.e., the location of the tag on the object, are stored in the object inventory. For rotation, instead of defining variable angles, six fixed values are defined associated with the sides of the bounding box: (1) right, (2) left, (3) top, (4) bottom, (5) front, (6) rear. So, when the tag is attached on the object, its orientation must comply with this definition in the object inventory. The rotation matrix, which is a function of rotation angles around each axis, $R(\alpha,\beta,\gamma)$, is defined for each façade as follows:

$$\begin{aligned}
R_{right} &= R\left(0, \quad -\pi/2, \quad 0\right)\\
R_{left} &= R\left(0, \quad \pi/2, \quad 0\right)\\
R_{top} &= R\left(\pi/2, \quad 0, \quad 0\right)\\
R_{bottom} &= R\left(-\pi/2, \quad 0, \quad 0\right)\\
R_{front} &= R\left(0, \quad 0, \quad 0\right)\\
R_{rear} &= R\left(0, \quad -\pi, \quad 0\right)
\end{aligned} \qquad \text{Eq.68}$$

where the angles are defined as radians, and the rotations are given from tag to object so as to provide the coordinate transformation from object to tag frame. When a tag is generated in the object management subunit, its orientation is stored in the object inventory with these predefined rotations, $R_{Tag\rightarrow Object} \in \left[R_{right}, R_{left}, R_{top}, \ldots, R_{rear}\right]$.

Similarly, the translation of each tag, $\vec{T}_{Tag\rightarrow Object} = \left[T_x \quad T_y \quad T_z\right]$, is defined with respect to the tag reference frame in the object inventory as a part of the tag location data (Figure 45). The result display subunit applies the transformation to each of the bounding box points:

$$P_i' = R_{Tag \to Object} \cdot P_i + \vec{T}_{Tag \to Object} \qquad \text{Eq.69}$$

where $i = 1 \ldots 8$. Subsequently, the coordinates of the new points, $P_1', P_2', P_3', \ldots, P_8'$, defined on the tag are transformed to room coordinates as follows:

$$P_i'' = R_{total} \cdot P_i' + \vec{T}_{total} \qquad \text{Eq.70}$$

The final point coordinates $P_1'', P_2'', P_3'', \ldots, P_8''$ give the real location of the object's bounding box inside room. These points are drawn on an image with a bird-eye view, in other words, abstracting from the height coordinate information. While drawing the image, the meter values are scaled to pixel values so as to fit the room inside the scene. The result display subunit allows the operator to adjust the scaling, so images with different resolutions can be generated based on the requirement. The objects are constructed in the image with the reconnection of the corner points. However, images are not drawn object by object, but instead, the bounding boxes are separated to their panels, and the images are formed panel by panel, where the panels with minimum height are drawn first. This drawing method enables the hindmost panels lie behind the proximate ones that are more visible to the viewer.

Similarly, the occupancy information is also drawn on the image with the same formula (Eq70) used for object location. However, this time, the point locations are not required. As mentioned in section 8.3, the transformations are also stored in the fusion table for occupancies. Based on these transformations, rough 2D region representations are drawn on the room image for occupancy detections. An example of a display output is given in chapter 11. Please note that these 2D graphical representations are not the space models defined in the lighting control system, and utilized by the simulator application (Figure 2), but instead a projection of the sensing results on the screen for user convenience.

Finally, it can be concluded that the diagrams given in Figure 43 and Figure 44 summarize the conceptual scheme of VIOLAS, where as input, the camera images are acquired, and as output, the sensed object information is stored in relevant tables, and consequently transmitted to the lighting control system.

9 IMPLEMENTATION

The implementation scheme of VIOLAS figures out how the blocks in the conceptual scheme are implemented and executed.

As mentioned in chapter 2, the solution intended for model generation should comply with the requirements specific to building environments. VIOLAS can be installed to a room equipped with one camera, whereas it can be applied to an entire building with one hundred rooms, equipped with one hundred cameras. The latter scenario involves intensive processing loads of data acquired from multiple sensors. To accommodate such loads, multiple computing resources must be utilized that can function in parallel and that can be reconfigured in a scalable fashion.

For this reason, VIOLAS is implemented in a distributed structure, where the subcomponents of the system communicate on Internet platform. Communication and data sharing is ruled by the distributed component object model (DCOM) protocol enabling software components to communicate directly over a network in a reliable, secure, and efficient manner (DCOM 2004). Distributed structure of VIOLAS provides scalability, incremental growth and enhanced performance derived from parallel operation. Additionally, remote data access permits information, resource sharing, and load balancing that allows efficient resource utilization.

Based on the above structure, VIOLAS software is divided into server and client tiers (Figure 47). *Application server* lies on the server tier. This module is the heart of the system that achieves two vital activities, resource management and data integration. Concerning its resource management activity, application server controls the distributed components, including the sensors and client modules, lying on the client tier. Sensors are the network cameras that fit in this structure by conveying video images like as distributed network devices. Client modules are the *image processing unit*s (IPU) implemented on different computers scattered across a facility. They process the input images captured from netcams, and perform the sensing activities. Towards this end, application server is liable for establishing their connection with the available netcams in the system. Concerning the data integration activity, application server combines the results obtained from multiple IPUs, and subsequently transfers them to the lighting control system.

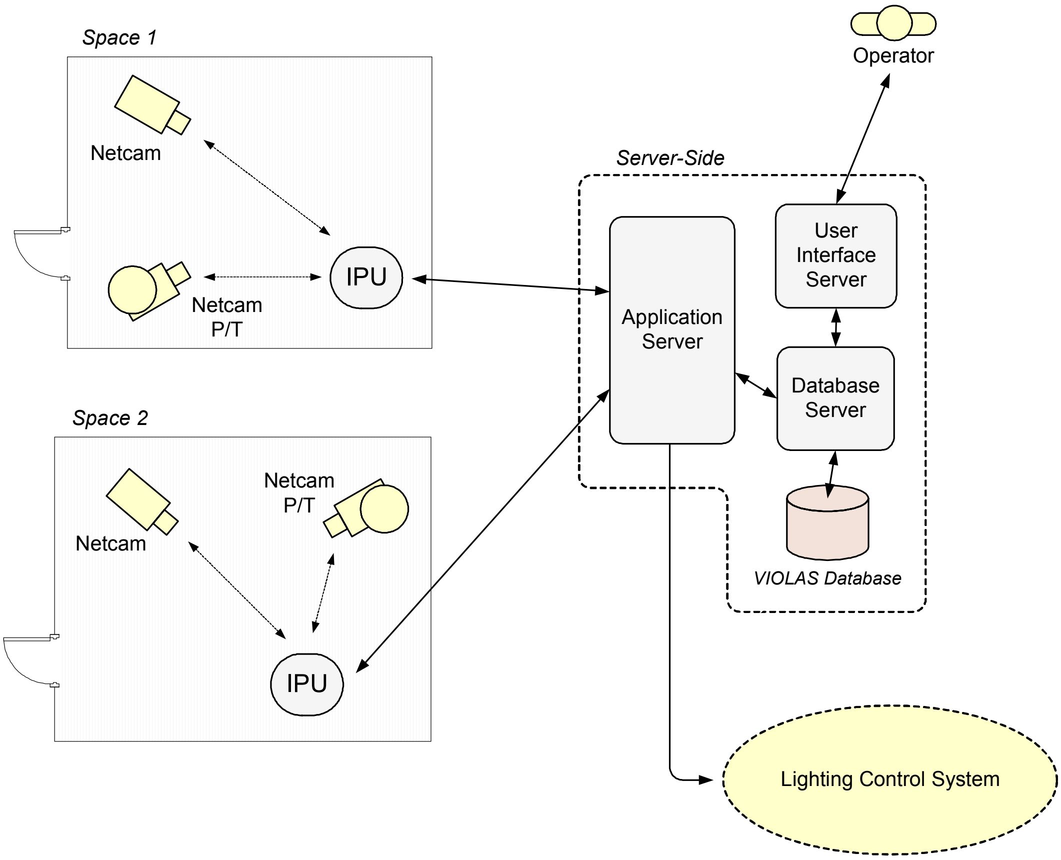


Figure 47. Distributed structure of VIOLAS.

Database server is the second subcomponent that lies on the server side. This module handles the data access demands of other modules, and provides their connection with the VIOLAS database. *User interface server* is the last module of the server tier that provides the communication between the operator and VIOLAS.

The modules are implemented on the Windows operating system with using Borland C++ software development environment. The implementation details of each module are given in the following sections.

9.1 Image Processing Unit

Image processing units (IPUs) are the programs that lie on the client side, and run parallel on different computers scattered around the facility. IPUs are the consumers of the system that exploit resources, i.e., sensors, namely, the cameras. Their main job is to acquire images from the cameras, and process them to extract the object information. Hence, the hardware interface, sensing core and the coordinate transformation units explained in the conceptual design are implemented inside this module. An IPU also performs additional processes to maintain the course of execution within the overall system. The structure of an IPU is given in Figure 48.

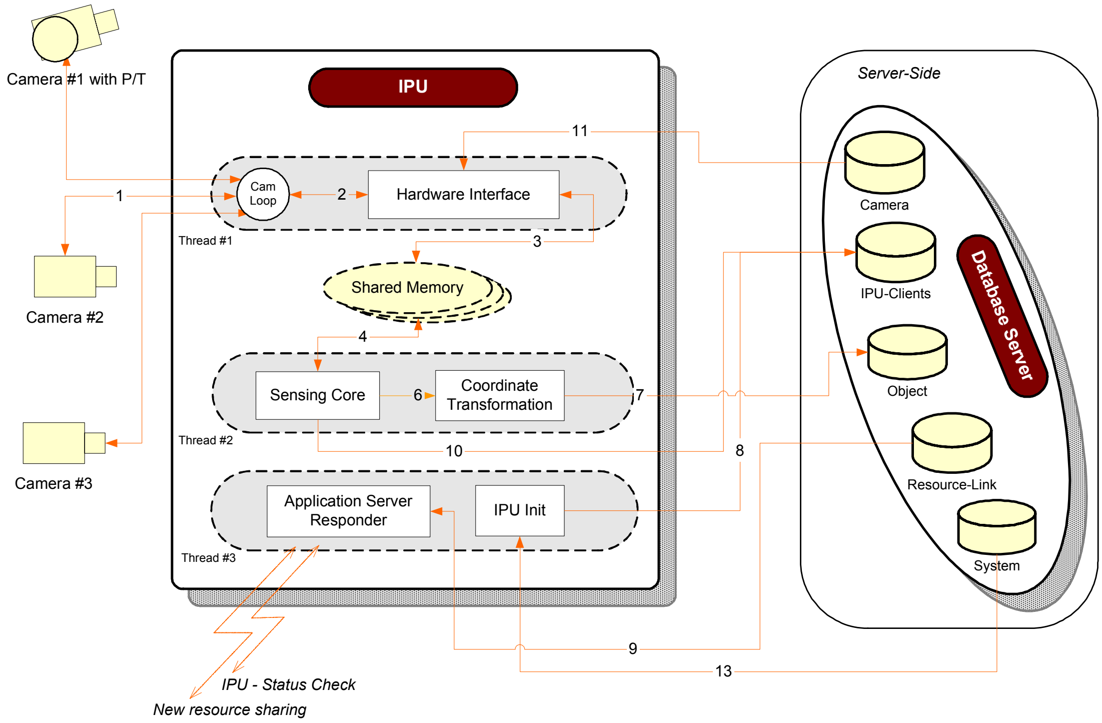


Figure 48. Structure of the image processing unit.

An IPU program runs three concurrent threads each of which is responsible with a distinct job as described in the following.

9.1.1 IPU Control

The IPU control mechanism provides the proper execution of the IPU and the communication with the application server in the course of VIOLAS operation. Towards this end, service thread (thread#3 in Figure 48) is implemented. This is the main thread created with the start of the program. Other threads are created within this thread. It involves three major functions:

First one, the IPU init function, runs at the startup of the program, and registers the IPU client to the system. The steps taken by this function are as follows: (1) Fetch the IP of the host computer. (2) Access the IPU-clients table (for reading), and check for other running IPUs. All data regarding IPU clients (ID number, IP address, communication port, performance, status, and last-activity time) are stored in the

IPU-clients table, and can be accessed after connecting the database server (to be explained in section 9.4). Based on the table lookup, get a unique ID number that has not been reserved before. (3) Access the IPU-clients table again to check for clients running on this computer only, and get a port number that also has not been reserved before. For this purpose, the function also makes a reading from the system table to learn the base-port number for the IPU clients (shown with dataflow line 13 in Figure 48). It may be possible for multiple IPUs to run on the same computer with the same IP, thus, each distinct IPU must possess a unique port for communication. (4) Access the IPU-clients table for the last time, and add this IPU together with the computed communication information (Figure 48, line 8).

Second function, the application server responder, provides the communication between the IPU and the application server. The tasks undertaken by this function are as follows: (1) Respond to the application server's “control” request sent for checking the status of the IPUs. The application server understands the status of the IPUs (whether they are active or not) by sending this request message with a TCP connection. (2) Fetch the application server's “new resource sharing” request. The IPU recognizes the cameras that shall be connected for image acquisition through this message. In order to learn the new resources, i.e., netcams, assigned to the IPU, the function accesses the resource-link table (Figure 48, line 9). Please note that, at the startup of the IPU, no cameras are assigned. The application server runs a new resource sharing function after recognizing the new IPU client. It rearranges the assignments between the cameras and the IPUs, and stores this arrangement in the resource-link table. Consequently, the application server sends a "new resource sharing” request to all IPUs whose camera assignments are changed. This is how the IPU (consumer) lines up its cameras (resources) at the initial state.

9.1.2 Image Acquisition

Image acquisition is handled by the image acquisition thread (thread#1 in Figure 48) in the IPU. This thread runs the hardware interface explained in the conceptual design, thus, acquires images from the cameras, transforms them into a usable format for image processing, and, in case of the existence of a pan-tilt unit, performs the motion of these devices (Figure 48, line 1).

However, the existence of a pan-tilt unit is not a sufficient criterion to perform its control. All IPUs are graded as *master* by default when they are first assigned to a camera by the application server. As a result of this assignment, multiple IPUs may be sharing one camera, a possible situation that can occur when the number of computing sources exceeds the number of sensors. This arrangement prevents the presence of jobless IPUs, and provides an efficient usage of these distributed modules in such “famine times”. On the other hand, the shared camera may be equipped with a pan-tilt unit. In order to eliminate the inconsistencies, the application server keeps one of the IPUs as the *master,* and degrades the others as *slave*. Only the master IPU is authorized to drive the pan-tilt unit. This ranking is conveyed to the image acquisition thread with the application server responder that also informs which cameras will be connected (as described in the previous section). Afterwards, the hardware interface firstly sets the connection with the cameras and the pan-tilt units if one is available, i.e., attached and authorized. Towards this end, hardware interface retrieves the communication details about these designated cameras from the camera table, together with other camera parameters that will be conveyed to further functions (Figure 48, line 11).

In chapter 4, where the visual sensors are introduced, the communication structure of the network cameras is also mentioned. Netcams possess a processor inside, and they are capable of connecting to the networks and broadcasting their images using the HTTP protocol like a regular web server. Thus, any application acting as a web client can access the camera, and download the images. Furthermore, the netcam manufacturers also provide plug-in components that can easily be embedded inside the applications. These plug-ins save the programmers from the workload of constructing a web client module and parsing the incoming data to pick the image inside. VIOLAS utilizes a Versacam-IC4 (Pentax 2006) and an IQeye3 (Iqinvision 2006) camera, and employs their plug-in components for communication. Using the plug-in component for Versacam-IC4 is necessary, since this camera, unlike its counterparts, employs a specific image compression technique (wavelet). Actually, most netcams use JPEG image compression that can be resolved in the web client module. As mentioned in the design scheme, camera table stores the model information for each camera. Based on this model information, the proper plug-in component is selected.

After the camera, the communication is set with the pan-tilt unit. As also mentioned in chapter 4, pan-tilt units are comprised of two main parts: the head and the controller. P/T head is the part where the camera or the camera housing is mounted. It involves two motors for executing the pan and tilt functions. P/T controller is simply an electronic interface employed to enable the control of the head by an external device. It receives control signals from the external device, and converts them to proper electrical voltages that drive the head part. The communication between the controllers and the external devices is performed by serial protocols like

RS232, RS485, or RS422 in physical layer. The protocols of this layer determine the cable connections and electrical voltages that correspond to 0s and 1s. In VIOLAS, a Visca DCP-24 device (GNT 2006) is employed for the controller part, and a Mustang P25 (Bewator 2006) is used for the head. The controller provides the physical connection between the head and the external device using the RS232 protocol. The external device can be either a keyboard-controller used widely in surveillance systems, or a PC. In our case, it is the network camera. Based on the given structure, netcams provide a serial output for the control of pan-tilt units. A 9-pin D port (D9) mounted for this purpose at the back of the IQeye3 is shown in Figure 49.

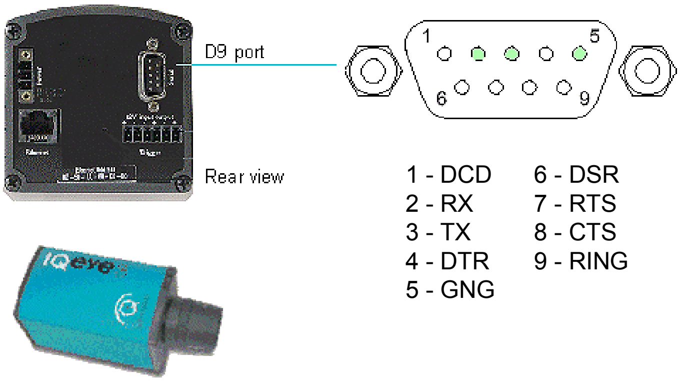
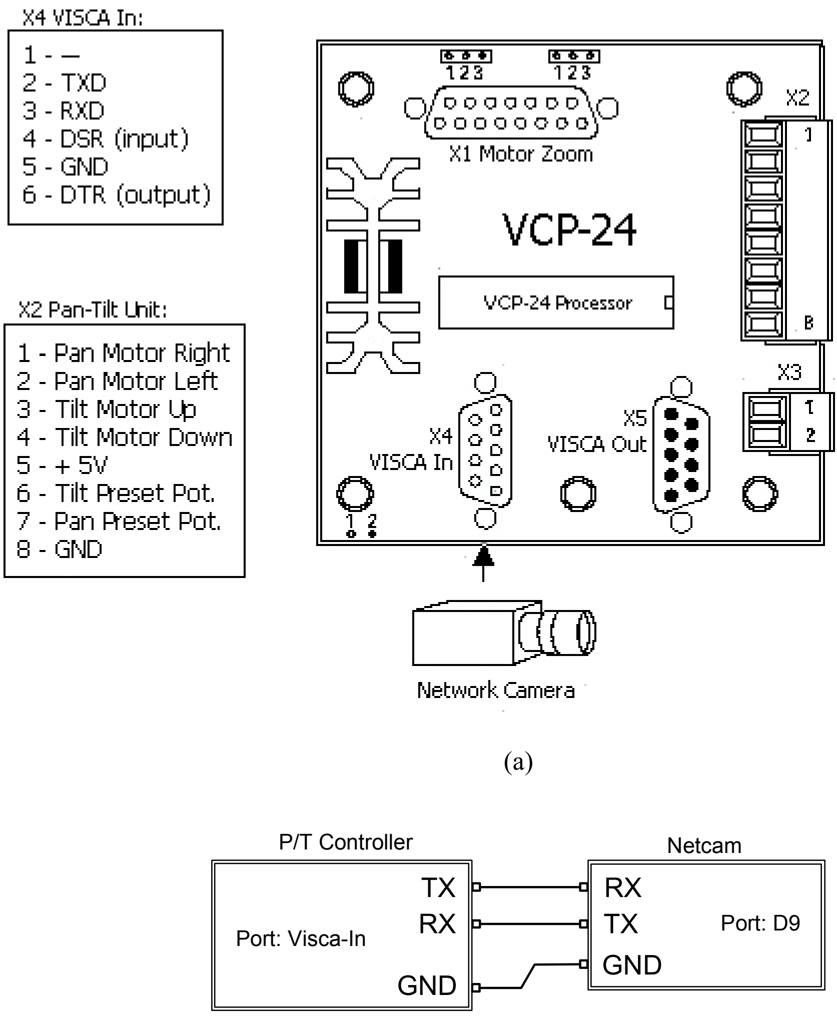


Figure 49. Serial D9 port of an IQeye3 camera.

In RS232 protocol, only the RX (data receive), TX (data transmit), and GND (ground) pins of a D9 port are utilized. These pin outputs are connected to the corresponding input pins of the Visca controller as shown in Figure 50 (Visca-in port marked with X4 sign). The P/T head is also connected to the controller so that Visca can convert the incoming commands into signals that drive the head motors.

The IQeye3 cameras provide a flexible serial output control. After setting a TCP socket connection to the camera's 3001st port, any dataset sent out is directly conveyed to the D9 output by the camera. The input data is also transferred through the same path back to the socket server that sets the connection. However, it is not possible to say that this serial output control method is standard among all network cameras. Some manufacturers, like that of the Versacam-IC4, instead use the plug-in components to control the serial output, and they accept a limited number of control commands predefined in a lookup table.



(a)

(b)

Figure 50. (a) Visca DCP-24 controller (GNT 2006). (b) RS232 connection layout between the camera and the controller.

The same situation is also present for the data link layer of the communication. Data link layer resides at the top of the physical layer, and these two layers together define how the communication is set between the controller and camera. Data link layer defines the data flow control, error handling and handshake between the two nodes. CRS (cyclic-redundancy-check), checksum are some of the famous error-handling algorithms, and CTS/RTS (clear-to-send/ready-to-send), XON/XOFF are some of the popular flow-control methods. The protocols of this layer combine these methods, and provide an error-free communication. In contradiction to the physical layer, there are no standard protocols for data link among P/T controllers. Main pan-tilt manufacturers develop their own protocols for communication. Among these proprietary protocols, some do not provide error handling, whereas some make use of CRS or checksum. Handshake between the nodes is usually not applied. Each protocol uses its own flow control, which is usually a simple quasi stop-and-wait method. These controller protocols also define the set of commands (pan-left, tilt-right...etc) used to control the P/T heads.

Because of this variety in the data link protocols, netcam manufacturers produce configurable serial outputs. The Visca controller uses the following values in the data link layer: 9600 baud data transfer speed, 8 bit data length (in one data packet), 1 start and 1 stop bits (used to designate the begin and end points of the data packet), no XON/XOFF, no hardware handshake. Prior to the connection, the IQeye3 and Versacam-IC4's serial outputs are configured with these values. (Detailed information about the layers and protocols used in computer communications can be found in Tanenbaum 2002).

So far, how the communication is set between the IPU, camera and the pan-tilt unit is described. As seen above, there is no direct connection between the IPU and the pan-tilt unit, since all the communication is set through the netcam. Thus, no additional infrastructure is required between the IPU and the pan-tilt devices (also shown in Figure 7, chapter 4). After the connection, hardware interface can drive the pan-tilt unit by sending commands through this communication channel. The command sets are also proprietary, and different datasets are used among different pan-tilt models and manufacturers. VIOLAS uses one model, Visca DCP-24, but it is also possible to employ different models of P/T controllers with a similar method used for cameras: based on the pan-tilt model information stored in the camera table, proper data link protocol and command set can be selected for communication.

Because of the variety, P/T manufacturers provide command set documents for the developers. Regarding the pan-tilt control, these command sets mostly cover the management of the movement with the supplied pan and tilt angle values. The controllers may provide the control of devices other than the pan-tilt units, like motorized zooming, thus the command sets may include additional definitions. A sample command of the Visca DCP-24 controller is given below in hexadecimal code:

```
8x 01 06 03 vv ww 0y 0y 0y 0y 0z 0z 0z 0z FF
```

The first `8x` and last `FF` bits mark the header and terminator parts. These values are present in all commands so that Visca can partition the incoming data. The following bits, `01 06 03`, mark the definition of the command, which is "relative position drive" in this particular example. This command sets the relative coordinates between the current position to the target position. The parameters, *`vv`* and *ww,* are determined by the developer, and they define the pan and tilt motion speed respectively. Finally, the parameters, `yyyy` and `zzzz`, define the new pan and tilt positions in relative values. Another example can be the following command that sends the pan-tilt unit to its home position, i.e., the position where pan and tilt angles are 0 degrees.

```
8x 01 06 04 FF
```

The relative-positioning command given above is utilized by the hardware interface to drive the pan-tilt unit. As mentioned in the design scheme, the angles that define the FOV range of the camera-lens are stored in the camera table. These parameters are used so that the pan-tilt unit is moved relatively by a tilt angle of vertical FOV degrees, or by a pan angle of horizontal FOV degrees in each step. This prevents the camera from skipping scenes between the two successive steps. The pan and tilt ranges are also defined in the camera table, thus the device can scan only a portion of the room when it is not necessary to perform a full circular movement, most likely when the pan-tilt unit is placed in the corner.

Starting from the home position, the room is scanned with a certain path. Firstly, the left part of the pan-tilt unit (corresponds to the positive pan angle values) is scanned. In each pan position, the device performs a vertical scanning from top to bottom with the tilt angles adjusted to the vertical FOV. After accomplishing the vertical scan within the tilt-range, the device moves to the next pan position. This time, it performs a vertical scanning from bottom to up. Using this path, the device scans its left side within the defined pan-range, then, returns to its home position, and performs the same course on the right side. Thus, the entire room is scanned with the minimum step, and without allowing any scene repetitions.

After the end of each step, pan and tilt angles of the P/T unit change, and these values are important for the further coordinate transformation. The hardware interface of the hereby IPU writes the new pan and tilt angles back to the camera table, enabling the slave IPUs to be aware of the current pan-tilt positions. Therefore, the IPUs access the camera table right before every image acquisition to retrieve the up-to-date pan and tilt angles.

However, the new pan and tilt angles are not updated immediately. The motion of the pan-tilt unit must be synchronized with the angle values. It takes some time for the device to reach to its next position. This period is adjusted with a timeout value. Images acquired within this timeout duration are not taken into consideration, since the camera takes these images on the fly. The camera’s on-the-fly status is written in the camera table right after the movement ignition. Thus, all the IPUs using this camera are aware of its status, and disregard the processing results within this period. After the timeout duration, the pan and tilt angles are updated on the database together with removing the on-the-fly flag, which enables the processing of images in the next step.

By using the methods described above, the hardware interface acquires an image from the camera, and drives the pan-tilt unit to its next position. After a short period of suspension, the camera loop unit runs, and switches the camera. This activates the hardware interface to acquire a new image from the next assigned netcam, and, maybe (to be explained in the following), to drive its pan-tilt unit to the next position. With the reiteration of the loop unit's execution, pan-tilt devices are moved step by step, and images are continuously procured from the cameras for processing.

The method for acquiring images and driving pan-tilt units are described so far. However, when to perform these actions is not mentioned, because it is not under the image acquisition thread's control to decide how frequently to run the camera loop unit, and when to drive the pan-tilt device to its next step. This thread only implements the P/T driving action, and does this with the activation of another thread: image processing. Image processing thread prevents the movement of the pan-tilt unit, until it processes sufficient images to sense the scene context. Therefore, the hardware interface may pursue to acquire more images from the same position, even though a pan-tilt device is attached to the camera, and the IPU is a master authorized for its control. The camera loop frequency is also controlled by the image processing thread. The loop is reiterated in every t millisecond, value of which is determined dynamically by image processing (explained in detail in the following section).

The acquired images (Figure 48, line 2) are transformed to a standard data structure, MS Windows *Bitmap* object (Figure 48, line 3), and stored in the shared memory with the parameters of the camera (pan-tilt angles, on-the-fly status and the others given in the design scheme) from which the image is taken. Shared memory is created dynamically depending on the number of cameras assigned to that particular IPU. Each camera has its own shared memory space, which is actually a global memory space shared by the image acquisition and image processing threads. Since these threads are running concurrently within the IPU program, their simultaneous access to the global memory may cause inconsistencies. In order to prevent this artifact, the accesses of the threads to the shared memory are *mutually excluded*. A programming standard, "semaphore" structure (Tanenbaum 1992), is employed to perform the mutual exclusion. The employment of this structure avoids the simultaneous access of the threads to the shared memory. The second thread demanding access to the global memory space is suspended until the first comer accomplishes its access. The CPU power is not consumed by the second thread during its suspension, enabling a high system efficiency. Since the shared memories of the cameras are isolated, the image processing thread can read the image of the first camera, while the image acquisition thread writes the image of the second. This augments the efficiency with preventing the threads to suspend each other for irrelevant data access.

As seen above, all of the control and access mechanisms implemented for the VIOLAS hardware components are wrapped inside the hardware interface. Other software components in the system access the hardware devices in an isolated manner, without being obliged to know the underlying communication details. In case of any hardware device change, the modification of the hardware interface function is sufficient for the operation of the overall system.

9.1.3 Image Processing

Image processing is provided with the implementation of the image processing thread (thread#2 in Figure 48). This thread retrieves the images from the shared memory together with the required parameters (Figure 48, line 4) as stated above, and processes them so as to sense the object data. Towards this end, image processing thread executes the sensing core and coordinate transformation units of the design scheme. The dataflow lines 6, and 7 in Figure 48 show the data transfer from sensing core to the coordinate transformation, and subsequently, to the object table.

The image processing thread is bestowed a substantial control over image acquisition. This control provides the synchronization between the image acquisition (producer thread) and the image processing (consumer thread). Firstly, the data acquisition frequency, t, is set by the image processing thread. This adjustment balances the image production rate with the image consumption speed, which is necessary for the efficient usage of the CPU among the threads. Within the IPU, image processing substantially consumes the CPU of the computer rather than the image acquisition. The execution of image acquisition with a high frequency results in the production of high amount of images that cannot be processed on time and overwritten by the new ones. This is a waste of the CPU power, which is very valuable for the image processing, Therefore, the value of t, amount of time that takes for image processing, is clocked by the image processing thread, and used as a break. This value is conveyed to the image acquisition over the shared memory, and also stored in the IPU-clients table (Figure 48, line 10).

Secondly, image processing thread activates the movement of the pan-tilt units towards their next position. After processing sufficient images for sensing the context information in the scene, thread writes its movement demand as a data to the corresponding camera's shared memory (if the IPU is ranked as master). After proceeding to the next camera queued up by the camera loop, the hardware interface first checks for the presence of the movement demand, and if any, drives the pan-tilt unit to its next step. The sufficiency of sensing is essentially determined by the occupancy detection, where a sequence of temporal images must be processed. Thus, only the master IPU executes the occupancy detection in the sensing core, if the camera is shared among multiple IPUs.

Eventually, it can be concluded that the IPUs are implemented based on the concurrent and synchronized execution of the image acquisition and processing threads. Another alternative to the above software design is to implement a distinct thread for each camera and to run the image acquisition and processing sequentially. However, this scheme reduces the performance for single-processor computers because of the overhead that occurs on the CPU take-over among the threads. Especially, if the amount of cameras increases per IPU, the performance decreases critically, where the cameras are blocking each other without any system control. Therefore, such a design is avoided, and the above structure is utilized.

9.2 Application Server

As the key component in VIOLAS, the application server controls the overall system. First, it manages data integration by combining the results coming from parallel running IPUs. Second, it dynamically performs the assignment of resources to consumers. In other words, the application server detects changes in the status of the cameras and IPUs in the system, and accordingly rearranges the assignments of cameras to the existing IPU clients. While performing these assignments, application server also balances the workload of IPUs. The structure of the application server is given in Figure 51.

The application server is comprised of five distinct functions. First three functions undertake the resource management activities, whereas the last two functions implement the data transfer and integration.

9.2.1 Resource Management

It is not feasible to assign an operator to continuously and manually manage the large amount of distributed components (cameras and IPUs). Therefore, the implementation of resource management is important for VIOLAS towards its adaptation to built environments. For this purpose, three functions given in the following are implemented.

First function is the IPU controller that is executed continuously to check the status of the IPU clients. It connects to the database server, and reads the available IPUs and their communication details (Figure 51, line 3) from the IPU-clients table. Then, it sends a “control” request to each of the IPUs (see also application server responder in Figure 48). If the IPU responds within a timeout duration, its status is updated as “active”. If the IPU fails to respond in the given timeout duration, or an error occurs during the connection, its status is updated as “inactive”. Afterwards, the status of the IPUs are updated back in the IPU-clients table. If the IPU is active, the last-activity-time data is also updated with the current time. This function also accesses the system table to retrieve the connection timeout period and lifetime of IPU clients (Figure 51, line 1). The inactive clients whose last-activity times exceed the lifetime duration are removed from the system by this function. Please note that the application server verifies the IPU with the ID number conveyed in the response message in order not to mistake the client for a program that gives a very different service in the same computer and from the same port.

The second function, camera controller, checks the status of the cameras in a similar way. It reads the available cameras and their communication details (Figure 51, line 4) from the camera table, and then sends a “control” request to each of the cameras. The netcams have a built-in FTP server. If the camera's FTP server responds within a timeout duration, the camera's status is updated as “active”. If the camera's FTP server fails to respond in the given timeout duration, or an error occurs during the connection, the camera's status is updated as “inactive”. Similar with the IPU controller, the status of the cameras are updated back in the camera table. If the camera is active, the last-activity-time data is also updated with the current time. This function also accesses the system table to retrieve the connection timeout period and lifetime of cameras (Figure 51, line 2). The inactive cameras whose last-activity times exceed the lifetime duration are removed from the system by this function.

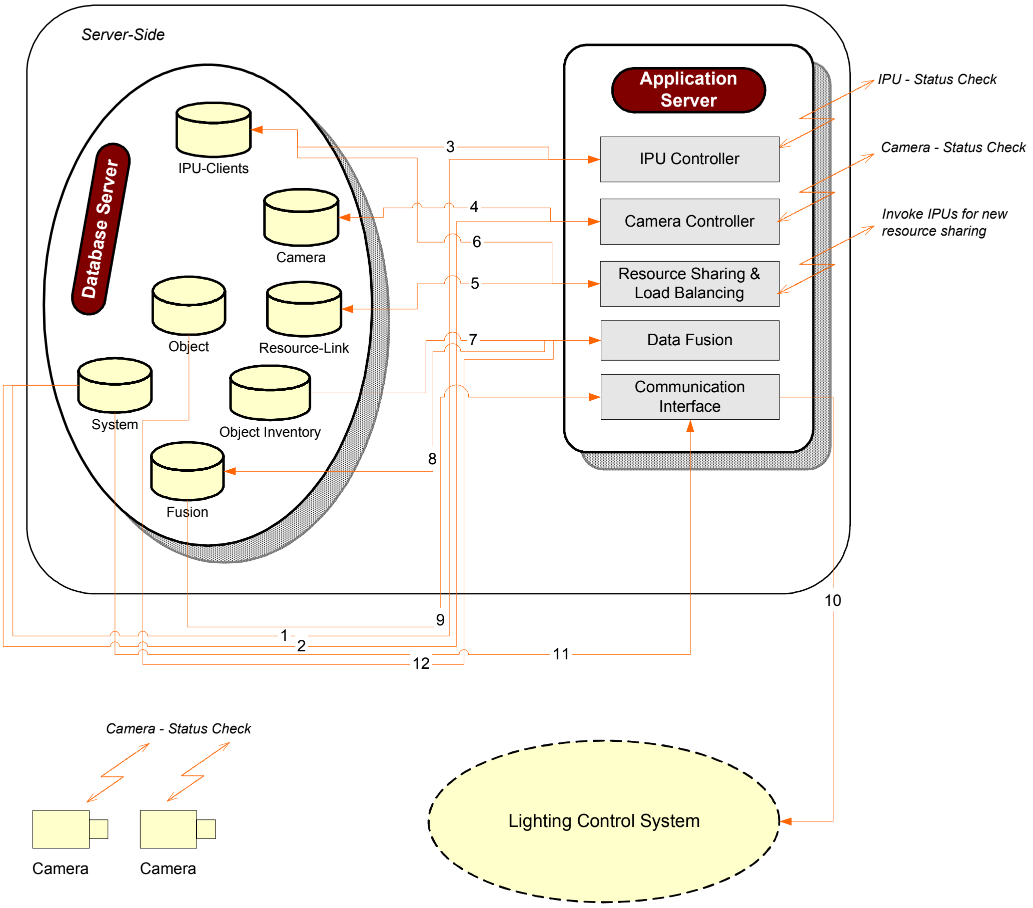


Figure 51. Structure of the application server.

Please note that, even though the status data of the cameras and IPUs are managed within the local memory of the application server, these values are also updated on the database so that the operator can also track the status of distributed client components in the system (Figure 53).

The third function, resource sharing and load balancing is executed at the initial run of the system and whenever the status of the cameras or IPUs change. This module assigns the active cameras (resources) to the active IPUs (consumers). This assignment is performed in such a manner that the workloads applied on the IPUs are continuously balanced. In order to provide this balancing, IPU clients feed their performance data back to the system (Figure 48, line 10). This function retrieves the IPUs' performance data from the IPU-clients table to decide the amount of cameras that will be assigned for each IPU (Figure 51, line 6).

In addition to the amount, resource sharing and load balancing function also determines which cameras will be assigned to a specific IPU by taking the network domain of the cameras into consideration. The cameras that lie on the same network area or on a network area close to the IPU are assigned to reduce the network load and increase the image transfer speed.

Consequently, as a result of such assignments, multiple IPUs can share a single camera, when the number of computing sources exceeds the number of available sensors. This arrangement prevents the existence of jobless IPUs, and organizes an efficient operation. As mentioned before, in order to eliminate inconsistencies, the application server ranks one of the IPUs as the *master*, and degrades the rest to *slave*. The master IPU is authorized to control the pan-tilt unit if one is available for the shared camera.

Eventually, the assignment results (Figure 51, line 5) are written in the resource-link table, and the IPUs are invoked by sending a “new resource sharing” request. By the receipt of the request, IPUs rearrange their connections with the cameras (see also application server responder in Figure 48).

9.2.2 Data Integration

Data integration is vital for handling multiple results coming from concurrent IPUs. Towards this end, data fusion function is implemented. As seen from its name, data fusion function continuously executes the data fusion unit in the design scheme.

In the design scheme, the situations that can generate inconstancies and reiterated results are described as (1) multiple cameras detecting the same tag, or, (2) multiple cameras detecting the tags of the same object. On the other hand, as explained in the resource sharing and load balancing function, multiple IPUs can also create reiterated results when they share the same camera. Thus, data fusion combines the sensed object data acquired from all of these parallel-running resources and consumers (Figure 51, line 7). The results are updated in the fusion table (Figure 51, line 8).

Communication interface function similarly executes the communication interface unit defined in the design scheme. This function transfers the final object data (Figure 51, line 9) to the lighting control system. It is also possible to transfer the results to a third party application that connects to the application server. The communication port assigned for the interface is stored in the system table (Figure 51, line 11). This function is executed similar to the resource sharing and load balancing function: at the initial run of the system, and whenever the status of the results change.

9.3 User Interface Server

User interface server implements the user interface unit of the design scheme. It provides the communication between VIOLAS and an operator through his/her web browser program, which can be executed from any computer on the network. The communication is achieved by common gateway interface (CGI). This method provides the execution of programs on the server platform, eliminating any hardware or software requirement on the client side. CGI programs are special applications that can run upon the initiation of a web server. They can be written with variable programming languages, and are supported by all the web servers independent from the manufacturer.

9.3.1 Common Development Scheme for CGI Applications

The basic structure of a CGI program developed with Borland C++ is given in Figure 52. CGI is a standard developed to generate a common platform to marshal the interaction between the web services and the programs. CGI program is executed externally by the web servers, and the program brings a dynamic feature to the static characteristic of the web pages. Using the CGI, web servers can acquire input data from the user, run the CGI program, and convey the generated results back to the web browser. The main benefits of the CGI are: (1) ability to process the input values, (2) ability to provide an interface for data access that web servers cannot manage individually (like connecting to a database).

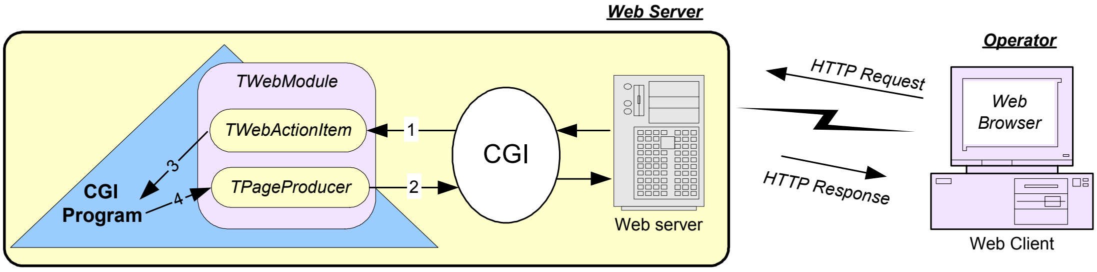


Figure 52. Common gateway interface.

CGI programs are not different from the regular programs, where specific inputs are used, and the outputs are declared with specific rules. CGI programs cannot be executed standalone. They need web servers, and must be placed in certain locations determined by the web server like “cgi-bin” or “scripts” folders. It should not be forgotten that the necessary file execution rights are given for the CGI programs within the operating system. The Internet users run these programs, and the lack of execution rights may result in failure.

CGI works in the following manner: (1) The web browser user sends an HTTP request to the web server without being aware of the status of the resource, whether it is static or dynamic. (2) Web server interprets the request, and understands that CGI should be started, so it initiates a new process. (3) Web server assigns the variables that involve the input data inside the HTTP request (Figure 52, line 1) as the environmental variables of the process so that the process can access them. (4) Web server executes the CGI program inside the new process to fulfill the demand of the user. (5) CGI program reads the input variables, processes them, and writes the results (Figure 52, line 2) to the standard output, STDOUT. (6) Web server reads the generated results from the STDOUT, constructs an HTTP response by adding the convenient data (header…etc), and finally sends it to the web browser.

In addition to this standard flow, Borland C++ provides the following components that facilitate the implementation of a CGI program: (1) *TWebModule*, (2) *TWebActionItem*, (3) *TPageProducer*.

TWebModule is the object name of the web module component that provides the communication between the CGI program and the web server. When a new web application, i.e., CGI program, is created in Borland C++, it automatically contains a web module. The web module serves as a repository for non-visual components: *TWebActionItem* and *TPageProducer*. A CGI application can have only one web module.

Web module also enables the CGI program to respond to HTTP request messages by passing the request objects, *TWebRequest* (Figure 52, line 3), and response objects, *TWebResponse* (Figure 52, line 4), to the appropriate “action items”. The *TWebModule* object manages a collection of action items, which know how to respond to HTTP request messages. Each action item component, *TWebActionItem*, performs a specific task in response to a given type of request message. Action items can completely respond to a request, or perform part of the response, and allow other action items to complete the job.

Page producers are the auxiliary components (*TPageProducer*) that help the action items to construct HTML codes for HTTP responses. *TPageProducer* takes an HTML template, and converts it by replacing special HTML-transparent tags with customized HTML code. A set of standard response templates filled in by page producers can be stored when generating the response to an HTTP request message.

An HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form:

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

The angle brackets, "<", and ">", define the entire scope of the tag. A pound sign "#" immediately follows the opening angle bracket with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The tag name can be any valid identifier, and identifies the type of conversion the tag represents.

Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form "ParamName=Value", where there is no space between the parameter name, the equals symbol, "=", and the value. The parameters are separated by whitespace. The angle brackets make the tag transparent to HTML browsers that do not recognize the "#TagName" construct.

9.3.2 Implementation in VIOLAS

User interface server is a CGI program that works as defined in the above structure. The flow diagram of the server program is given in Figure 53. The user interface server is placed inside a web server, and the requests of the operator are conveyed through the CGI interface of the web server. The web server runs the user interface server, and transfers the requests to the module as stated above. The following action items are declared inside the web module of the CGI program: (1) for logging in the system, *Login*, (2) regarding camera management, *CamList*, *CamView*, *CamQuery*, *CamCommit*, (3) regarding system management, *SysQuery*, *SysCommit*, (4) regarding user management, *UserList*, *UserQuery*, *UserCommit*, (5) regarding object management, *ObjectList*, *ObjectQuery*, *ObjectCommit*, (6) regarding result display, *ResDisp*, and finally (7) regarding the IPU tracking, *IpuList*.

The *Login* action item is the default action that is implemented when no action detail is given. It checks the "username" and "password" inputs of the operator (Figure 53, line 1) by connecting to the database server, and verifying the data with the ones defined in the users table. *Login* action item gives access to the main menu, if they are correct. Otherwise, it generates an invalid-user page with using its relevant *TPageProducer* component.

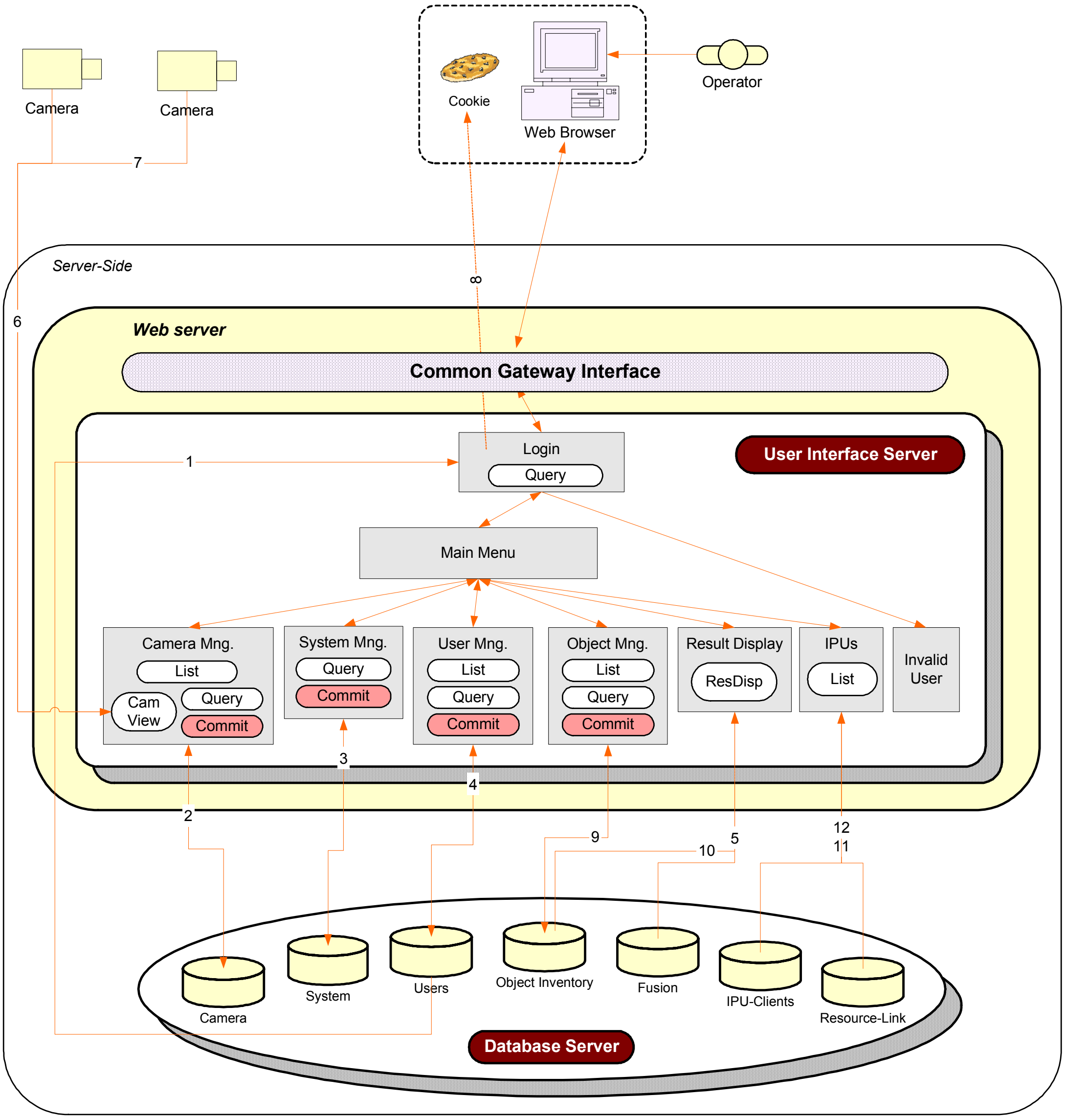


Figure 53. Structure of the user interface server.

If the “username” and “password” are correct, this action item places a *cookie* inside the web client's computer (Figure 53, line 8) to recognize the further connections of the client. This is how the user interface server identifies the connecting clients, and writes their usernames on the top of the web pages. On the other hand, the cookies have a timeout duration. If the operator's connection time exceeds this period, the login request is repeated by the user interface program. The cookie acceptance must be enabled in the setup menu of the web browser, otherwise user interface server will not be able to place the cookie, and the operator will get an “invalid user” message each time he/she logs on.

The action items named with *xxxList* nomenclature are presented at the first link of the main menu. Their function is to list the records in the relevant tables (Figure 53, lines 2, 4, 9, 11 and 12). There is one exception to this structure: system management. System parameters are comprised of one single record in the system table, so system management link in the main menu leads the operator directly to the *SysQuery* action item.

xxxQuery action items are implemented when the operator wants to add, modify or delete a record in the table. The record number and the action type (delete or modify) are conveyed inside the HTML links that are created by the *xxxList* action items.

xxxCommit action items are implemented at the last stage to commit the “modify” or “delete” transactions on the database server. These action items are marked with a different color in the figure to state that some users may not possess an execution right for this function. Each time an action is executed by the client request, two a-priori functions are implemented: one tests the database connection, other checks the user validity by reading the contents of the cookie. A failure in the database connection results in the generation of a warning message page. A failure in user validation forces the user to re-login by directing him to the *Login* action item. Thus, this function prevents the direct access of the user to the action items other than the *Login* without being logged on to the system. While performing the a-priori user validation, *xxxCommit* action items also read the user types stored in the user table. There are two types of users defined in the system: (1) administrative users, (2) standard users. Only administrative users have the right to commit changes and deletions in the tables. In order to provide this capability, the *xxxCommit* action items check the user type before sending requests to the database server.

The above scheme marshals the data retrieval and manipulation. In addition to this structure, the *ResultDisp* action item reads the final results (Figure 53, line 5) from the fusion table, and generates a bitmap file that demonstrates the simple 2D figures of the objects, as defined in the result display subunit in the design scheme of VIOLAS.

Finally, the *CamView* action item is accessed by the link that is created in the web page of *CamList* action for each active camera. This action reads the ID of the camera as input, connects to that camera, and generates a new page where the raw camera images (Figure 53, lines 6 and 7) can be viewed. As explained in section 4.4, the network cameras have built-in web servers that publish raw images over HTTP, and enable the display of camera images in personal HTML documents.

9.4 Database Server

Database server is also an important module in VIOLAS, as its underlying structure enables the access to distributed COM objects over a network. COM is a software architecture that standardizes the programming interfaces, implementation models, and the data structures so that the non-compliances among different software platforms are removed. Derived from this architecture, distributed COM objects (DCOM 2004) are developed that can be employed by other applications remotely in a network environment. This allows the software components to access data or implement a function remotely from different computers.

Based on this model, the software components of VIOLAS, i.e., image processing units, application server and user interface server, act like client applications that request service from the database server. Database server manages the connection between these software components and the VIOLAS database where the information resides. It provides a convenient interface that responds to the client applications' requests like modify, add, delete, or retrieve.

Towards this end, database server also decodes the tables’ file format in the database. The VIOLAS database involves tables that store relevant data in XML format. XML files posses a certain tag structure like HTML files. Based on this structure, these tables are constructed with one metadata part, where the information about data structure is defined, and one data part, where the records are stored. Together with this partitioning, a specific tag nomenclature is used so that Borland C++, describing more specifically, the *TDataSetProvider* object (to be explained in the following), can resolve the data in the table. This file structure is illustrated in Figure 54 with a sample table. The list of available tables in the database is as follows: (1) camera table, (2) IPU-clients, (3) resource-link table, (4) object table, (5) fusion table, (6) object inventory, (7) system table, (8) users table.

9.4.1 Remote Data Module

“Remote data module” located inside the database server manages data transfer and manipulation between the database and client applications. It involves two Borland C++ components that undertake the data transfer and manipulation processes for the programmer: (1) *IAppServer*, (2) *TDataSetProvider* (Figure 55).

```
<?xml version="1.0" standalone="yes" ?>
<DATAPACKET Version="2.0">
 <METADATA>
  <FIELDS>
   <FIELD attrname="USERNAME" fieldtype="string" WIDTH="20" />
   <FIELD attrname="PASSWORD" fieldtype="string" WIDTH="20" />
   <FIELD attrname="NAME" fieldtype="string" WIDTH="20" />
   <FIELD attrname="AUTHORIZATION" fieldtype="string" WIDTH="20" />
  </FIELDS>
 </METADATA>
 <ROWDATA>
  <ROW USERNAME="admin" PASSWORD="admin" NAME="System Admin" AUTHORIZATION="1" />
  <ROW USERNAME="guest" PASSWORD="guest" NAME="Guest User" AUTHORIZATION="0" />
  <ROW USERNAME="oguzi" PASSWORD="oguzi" NAME="Oguz Icoglu" AUTHORIZATION="1" />
 </ROWDATA>
</DATAPACKET>
```

Figure 54. User table given as an example to show the structure of a XML file.

TDataSetProvider (object name of the dataset providers) supplies the mechanism by which *TClientDataSets* (object name of the client datasets) obtain their data. For each table, there is one “dataset provider” in the database server and one “client dataset” in the client application. Image processing unit, application server and the user interface server are the client applications of the database server that request database service. So, hereafter in this section, all of these software components of VIOLAS will be referred as “client applications”. Each of the client applications involves a client dataset for each of the tables it wants to access. Client datasets are the components in the client applications that provide access to a table in the VIOLAS database through the dataset providers (Figure 55).

Dataset provider procures data from the database to a client dataset, and resolves updates from a client dataset back to the underlying database. *TDataSetProvider* object serves as a data broker between the database server and the client dataset in the desktop client application. *TDataSetProvider* packages data from a table, and passes it in one or more transportable data packets to the client dataset. The client dataset receives the data packets, and reconstructs the data to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data, and sends the updates to the data provider residing on the database server. The data provider applies the updates back to the underlying database table.

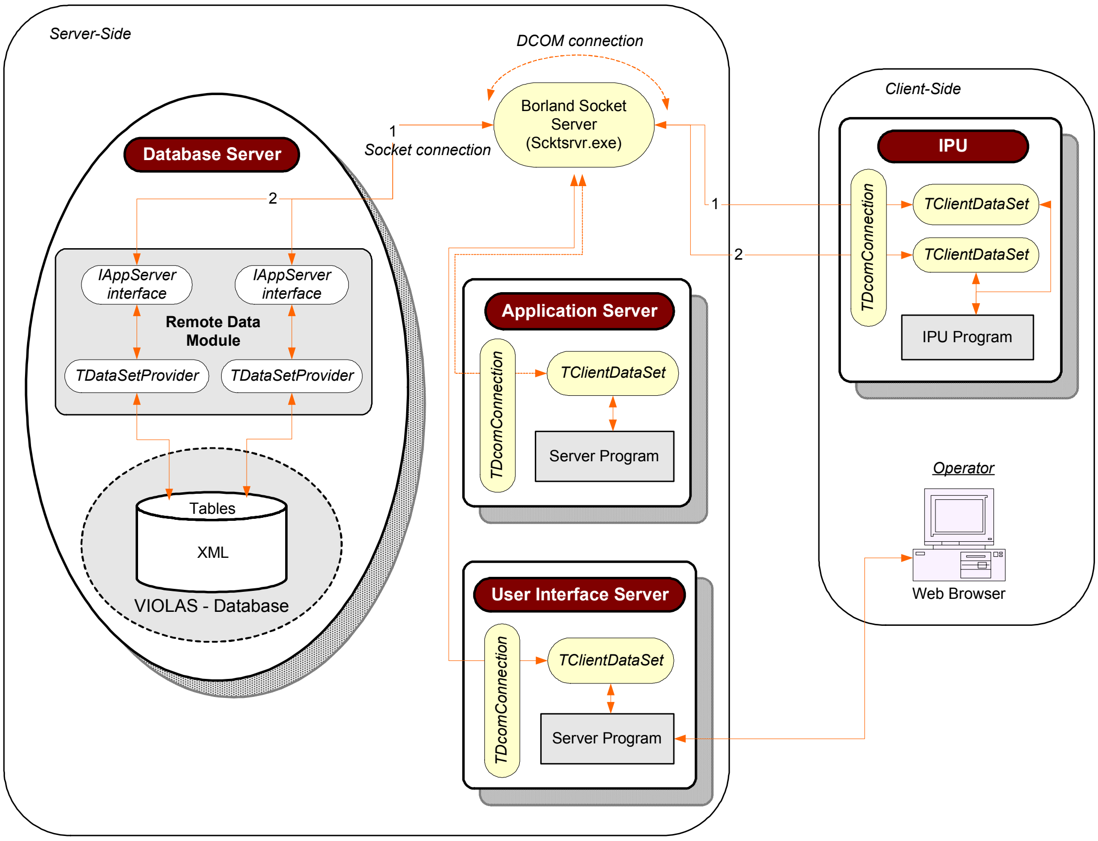


Figure 55. Structure of the database server.

Database Server does not establish any connection with client applications. Instead, connection is maintained by client applications. Client applications can access *TDataSetProvider* methods using the *IAppServer* interface of the remote data module. This interface provides the basis of communication for the distributed applications. Client datasets obtain an *IAppServer* instance from a connection component (explained in the following) in the client application. Then, the client datasets can communicate with the data providers directly by calling the data provider component's *TDataSetProvider* methods.

9.4.2 Communication with the Database Server

The client applications, additionally, require a connection component to connect to the database server. This component identifies the protocol for communicating with the database server. Each type of connection component represents a different communication protocol. One option is to use a DCOM connection component, *TDCOMConnection*, to establish and maintain the connection with the database server. *TDCOMConnection* uses the computer name of the host computer to identify the machine on which the server resides. Once the connection is established, the client application registers any or all of its client datasets with *TDCOMConnection*, and these client datasets use the *IAppServer* interface from the DCOM connection component to communicate with data providers on the database server. DCOM provides the most direct approach to communication, requiring no additional runtime applications on the server. However, the DCOM connection must be configured prior to the connection with using the application, DcomCnfg.exe, provided by the MS Windows operating system (DCOM 2004).

Another alternative is the usage of a socket connection component, *TSocketConnection*. The connection to the database server is established using sockets from any machine that has a TCP/IP address. This method has the advantage of being applicable to more machines, but does not provide for using any security protocols like the DCOM connection protocol. *TSocketConnection* identifies the server machine using the IP address or host name of the server system. *TSocketConnection* establishes the initial connection between the client application and the database server using TCP/IP as stated above. To use *TSocketConnection*, the database server must also be running the Borland socket server, Scktsrvr.exe (the socket dispatcher), program. Instead of instantiating the remote data module directly from the client, sockets use Borland socket server (Scktsrvr.exe), which accepts client requests, and instantiates the database server using COM. The connection component, *TSocketConnection*, on the client and Scktsrvr.exe on the server are responsible for marshaling *IAppServer* calls.

In addition, no matter which connection protocol they use, the client applications need *ServerName* or *ServerGUID* information to identify the database server on the server machine. *ServerName* identifies the name of the database server to which the client application should connect. *ServerGUID* also serves for the same purpose, and specifies the GUID of the remote data module's interface. Using *ServerGUID* rather than *ServerName* to identify the database server is more robust, because it does not require the database server to be registered on the client system. However, either *ServerName* or *ServerGUID* must be provided so that the dispatch connection can create and communicate with the appropriate server COM object.

9.4.3 Implementation in VIOLAS

The information required for connecting to the server machine (IP address and port number for a socket connection, or the computer name for a DCOM connection), and for connecting to the database server on that machine (*ServerName*, or *ServerGUID*) are stored in the client application's initialization file (*.ini). This file resides on the directory where the executable exists. The client applications check for the database server connection by sending a void request message prior to making a real connection. If they cannot achieve a connection, an error message is raised, and the application is terminated, since there is no meaning for the application to keep on working without a database connection.

An example of the above scheme is demonstrated in the database server structure diagram given in Figure 55. The IPU is accessing the camera table (Figure 55, line 1) and the IPU-clients table (Figure 55, line 2). For each table connection, a client dataset component, *TClientDataSet,* is employed in the client application. These client datasets use the *IAppServer* interface from the DCOM connection component, *TDCOMConnection,* to communicate with the dataset providers, *TDataSetProvider,* in the remote data module. The data in the tables are accessed and manipulated by the dataset providers that resolve the underlying XML files. The *TDCOMConnection* component implemented in the IPU module establishes the communication between the client application and the database server, and manages the *IAppServer* calls. The same structure also applies to each of the other client applications, i.e., application server and user interface server, that wants to access the tables in the VIOLAS database.

10 AUXILIARY PROGRAMS IN VIOLAS

In addition to VIOLAS described in the previous chapter, two additional programs are implemented in the course of the project development.

10.1 Camera Calibration

During the extraction of the location information (section 5.3.2), the cameras are assumed to be already calibrated, i.e., the *intrinsic parameters* of the camera (Eq6) are known. These parameters are used to the transform the point locations from the image coordinate system to the camera coordinate system. Specifically, they are utilized in Eq9 to transform the parameters of the reference ellipse defined in pixel units to focal-length units. They are also employed in Eq47 for breaking the ambiguity with a similar reason, to transform the coordinates of the benchmark points.

As explained above, the camera calibration is simply the extraction of the *intrinsic parameters* of the camera. For a camera with fixed optics, these parameters are identical for all the images within the camera. However, they are not identical for different cameras, even though they are from the same model. Even the cameras that are manufactured sequentially from the same production line may posse different intrinsic parameters. Hence, each camera in VIOLAS is calibrated, i.e., its intrinsic parameters are extracted and stored in the camera table, before being employed in the system. Towards this end, an external camera calibration program is implemented. This program extracts the intrinsic parameters as explained in the following.

The transformation matrix that maps the image points to camera coordinates is given by:

$$M = \begin{bmatrix} f/k_u & k_\alpha & u_0 \\ 0 & f/k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.71}$$

This matrix is formed with the intrinsic parameters of the camera, where f is the focal length, (u_0, v_0) is the image centre defined in pixel coordinates, k_u and k_v are the effective pixel size given in horizontal and vertical directions respectively, and k_α is the skew. For most CCD cameras, the pixels are almost-perfectly rectangular, thus skew is negligible, $k_\alpha \approx 0$.

In the above paragraph, it is mentioned that the intrinsic parameters change in each camera. However, this does not include the effective pixel size, k_u and k_v. These parameters are fixed values for each camera model, and they are announced by the camera manufacturers. However, we assume that the pixel sizes are unknown, and have to be extracted. The focal length, f, is a variable parameter that depends on the lens properties, and changes with respect to the zooming ratio in adjustable zooming lenses. In VIOLAS, the cameras are fixed to a constant focal length, and their zooming ratio does not change after the calibration. So, the focal length, f, can be taken as a constant parameter. As seen from Eq71, f is not an independent parameter, and can be embedded in the k_u and k_v parameters as: $K_u = k_u / f$, and $K_v = k_v / f$. With the addition of the image center, (u_0, v_0), there are four variables to calculate for the camera calibration.

As mentioned in breaking the ambiguity (section 5.3.2.4), the projection of points in the target plane (tag plane), P_{tag}, into points in the image plane, P_{image}, can be calculated by applying the planar projective transformation:

$$P_{image} = M \cdot \left(R \cdot P_{tag} + \vec{T} \right) \qquad \text{Eq.72}$$

where M is the camera calibration matrix, and R and $\vec{T}$ are the results of location sensing before being transformed to the real-world coordinates. As a verification of Eq72, it can be seen that $\left(R \cdot P_{tag} + \vec{T} \right)$ corresponds to the points defined on the camera coordinate system, the evident result of the application of location sensing on the target frame. The planar projective transformation exploits the fact that every point lying on the target plane has a value of $Z = 0$. Therefore, Eq72 can be defined as:

$$\begin{bmatrix} s \cdot x_{image} \\ s \cdot y_{image} \\ s \end{bmatrix} = \begin{bmatrix} K_u & 0 & u_0 \\ 0 & K_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{32} & T_z \end{bmatrix} \cdot \begin{bmatrix} x_{tag} \\ y_{tag} \\ 1 \end{bmatrix} \qquad \text{Eq.73}$$

where (x_{image}, y_{image}) are the Cartesian pixel coordinates that correspond to the point, $P_{tag} = \lfloor x_{tag} \quad y_{tag} \quad 0 \rfloor$, on the tag frame. Eq73 can be extended as:

$$u_0 = x_{image} - \frac{(x_{tag}.R_{11} + y_{tag}.R_{12} + T_x)}{(x_{tag}.R_{31} + y_{tag}.R_{32} + T_z)} \cdot K_u \qquad \text{Eq.74}$$

$$v_0 = y_{image} - \frac{(x_{tag}.R_{21} + y_{tag}.R_{22} + T_y)}{(x_{tag}.R_{31} + y_{tag}.R_{32} + T_z)} \cdot K_v \qquad \text{Eq.75}$$

As seen above, two equations, Eq74 and Eq75, can be constructed from one tag. The unknown variables in these equations are: P_{tag}, P_{image}, R, $\vec{T}$, and the intrinsic parameters, u_0, v_0, K_u, K_v.

In order to eliminate the extra unknown variables, firstly, we need a fixed point on tag, namely P_{tag}, whose location on the image, P_{image}, can be extracted without the usage of camera calibration. In fact, there is a point, the *synchronization point*, used in the code deciphering phase of the target recognition (section 5.3.1.6). The image location of this fixed point (depicted as a star in Figure 16) is extracted directly on the image without utilizing the calibration matrix. Secondly, we need the location results, i.e., R, $\vec{T}$, of this tag. However, it is not possible to compute the location without knowing the intrinsic parameters. Towards this end, these values are measured, rather than calculated. In other words, the tag is placed at a certain location whose transformation values, R, $\vec{T}$ are already known.

The scheme described above removes the unknown values other than the intrinsic parameters. However, at least four equations are needed to compute the four unknowns. Therefore, the camera calibration utilizes two tags, the planar projective transformation of which makes four equations. Hence, the calibration program captures an image from the camera to be calibrated. The image contains two tags whose locations are known. Such an image is called the *calibration image*, and one used for IQeye3 camera is given in Figure 56.

The location of the tags in the calibration image can be given as (considering tag1 as the tag on the left, and tag2 as the one on the right with respect to the viewer):

$$\begin{aligned} &R_1 = R_2 = (0, \quad \pi, \quad 0) \\ &\vec{T}_1 = [0.5 \quad 0 \quad 2], \quad \vec{T}_2 = [-0.5 \quad 2.5 \quad 2] \end{aligned} \qquad \text{Eq.76}$$

As stated above, these values are not sensed, but measured manually. Henceforth, the camera calibration program applies target recognition to the calibration image, and extracts the pixel location of the synchronization points on the tags. Employing these values together with the known, R, $\vec{T}$, calibration program calculates the intrinsic parameters, u_0, v_0, K_u, K_v, by using the equations given in Eq74 and Eq75.

Consequently, towards the operation of the system, these calibration values are entered in VIOLAS through the medium of the user interface server's camera management unit.

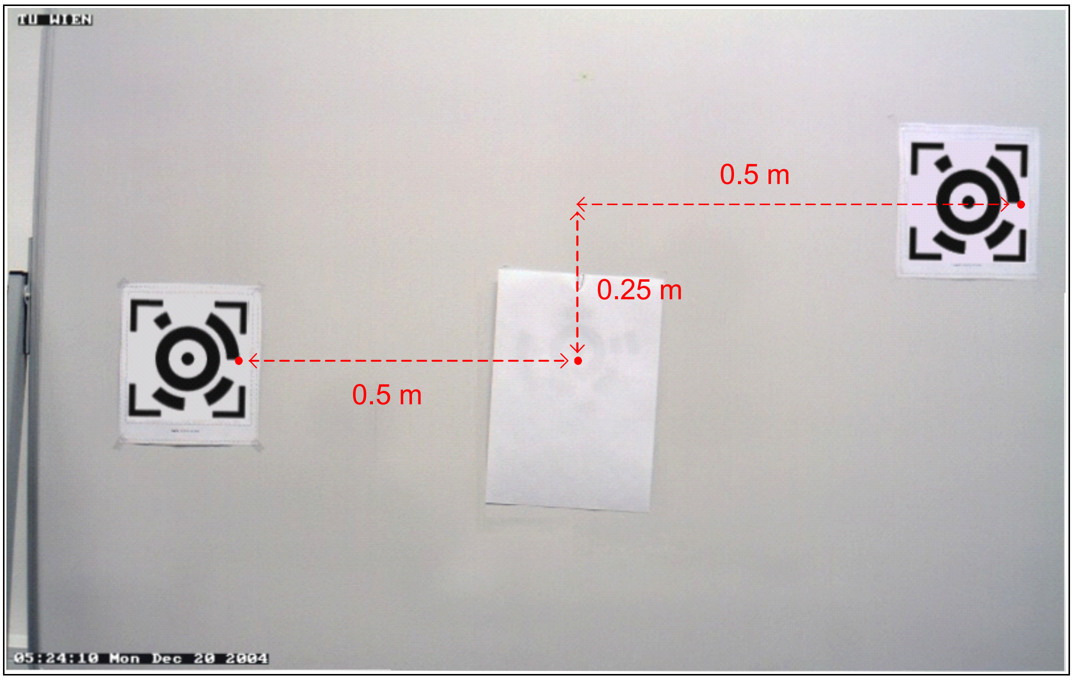
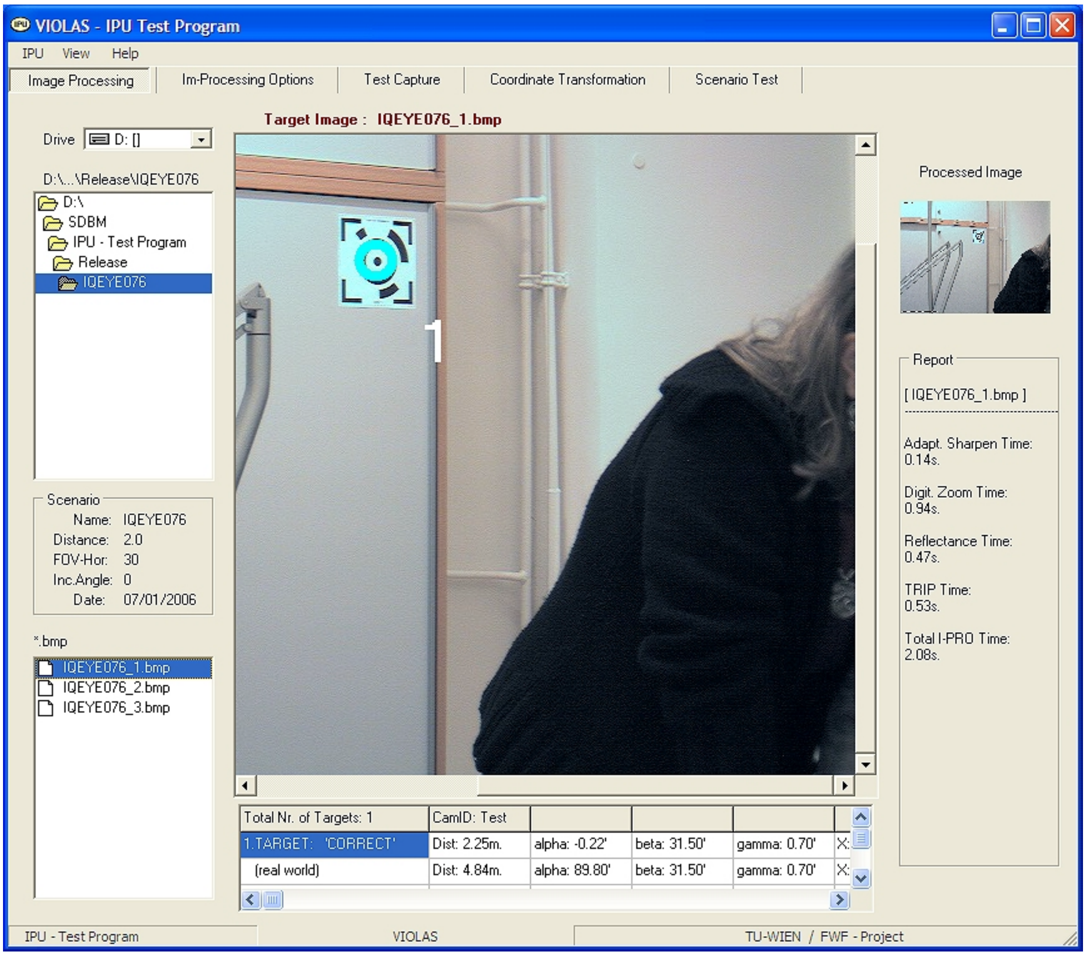


Figure 56. Calibration image.

As mentioned in the beginning of this section, the intrinsic parameters are dependent on the zooming ratio. Even though the camera lenses are fixed to a constant focal length, in other words, there is no optical zooming, the system performs a digital zooming on the input images as a part of the image enhancement procedure (described in section 6.2). The digital zooming affects the intrinsic parameters, however, does not necessitate a recalibration. Since the zooming in question is an artificial one performed digitally, in other words, simply doubles the resolution, the intrinsic parameters can be recalculated with applying the zooming ratio, k_z, as a factor. Thus, new parameters can be found by: $u'_0 = k_z \cdot u_0$, $v'_0 = k_z \cdot v_0$, $K'_u = k_z \cdot K_u$, and $K'_v = k_z \cdot K_v$.

10.2 Image-Processing Tester

The algorithms developed in the sensing core are tested with a standalone program, image-processing tester (Figure 57). It is not feasible to implement the sensing algorithms, and test their performance while VIOLAS is in operation. Towards this end, the location and occupancy sensing algorithms (together with coordinate transformation) are first implemented in the image-processing tester, and moved to the IPU modules after being tested in this auxiliary program.



Target Image : IQEYE076_1.bmp

Scenario
Name: IQEYE076
Distance: 2.0
FOV-Hor: 30
Inc.Angle: 0
Date: 07/01/2006

Report
[IQEYE076_1.bmp]
Adapt. Sharpen Time: 0.14s.
Digit. Zoom Time: 0.94s.
Reflectance Time: 0.47s.
TRIP Time: 0.53s.
Total I-PRO Time: 2.08s.

Total Nr. of Targets: 1	CamID: Test				
1.TARGET: 'CORRECT'	Dist: 2.25m.	alpha: -0.22'	beta: 31.50'	gamma: 0.70'	X:
(real world)	Dist: 4.84m.	alpha: 89.80'	beta: 31.50'	gamma: 0.70'	X:

Figure 57. Image-processing tester.

The structure of the image-processing tester is identical with the combination of the sensing core and coordinate transformation units given in the design scheme. The differences are the handling of input and output values. The inputs are not live video sequences, but the bitmap image files previously captured from the netcams, and stored in the computer. There is also no database connection, and the required camera parameters are read from an initialization file. Similarly, the results are displayed on the screen, or written to the local files for testing, rather than being transferred to the VIOLAS database.

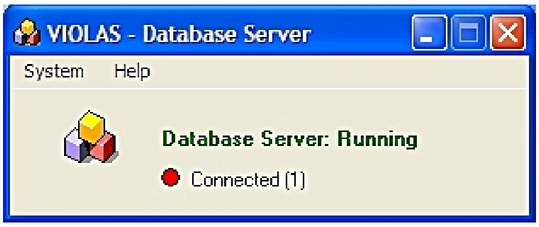
11 A DEMONSTRATIVE TEST

To evaluate the performance of VIOLAS, a demonstrative test is performed. Thereby, regularity of the system operation and accuracy of the sensing functions are observed.

11.1 Organization of the Test Platform

The test is implemented in Vienna Technical University, Department of Building Physics and Building Ecology. The university building is equipped with a local area network, thus, provides the required infrastructure. Within this network, our department possesses a domain conducted by a mainframe computer. The server applications, i.e., database server, application server and user interface server, are installed in the mainframe together with the VIOLAS database. This computer also runs a web server for broadcasting the department's web pages. So, the user interface server is placed under the specific folder where the web server can access and execute such CGI applications.

Before the operation of VIOLAS, required parameters must be initially defined through the medium of the user interface server. However, the database server is activated first so that the user interface can access the VIOLAS database. Figure 58 shows the database server program with one client connected: user interface server.

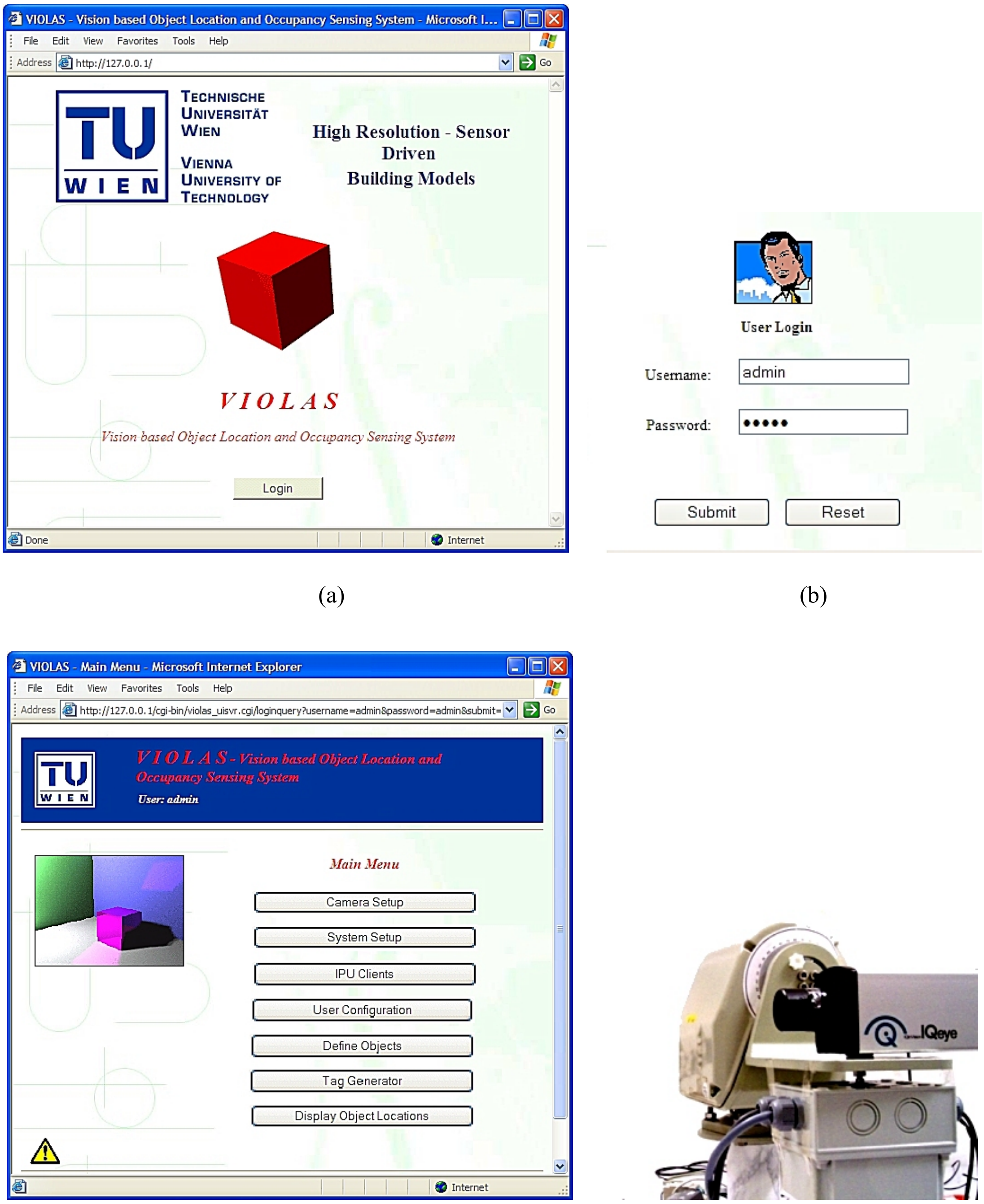


Database Server: Running

● Connected (1)

Figure 58. Database server program.

User interface server is activated with a web browser program executed from an operator's computer in the department. Figure 59 illustrates the connection to the user interface server and the main menu of the application.



VIOLAS - Vision based Object Location and Occupancy Sensing System - Microsoft I...

Address http://127.0.0.1/

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

High Resolution - Sensor Driven Building Models

VIOLAS

Vision based Object Location and Occupancy Sensing System

User Login

Username: admin

Password: •••••

VIOLAS - Main Menu - Microsoft Internet Explorer

Address http://127.0.0.1/cgi-bin/violas_uisvr.cgi/loginquery?username=admin&password=admin&submit=

VIOLAS - Vision based Object Location and Occupancy Sensing System

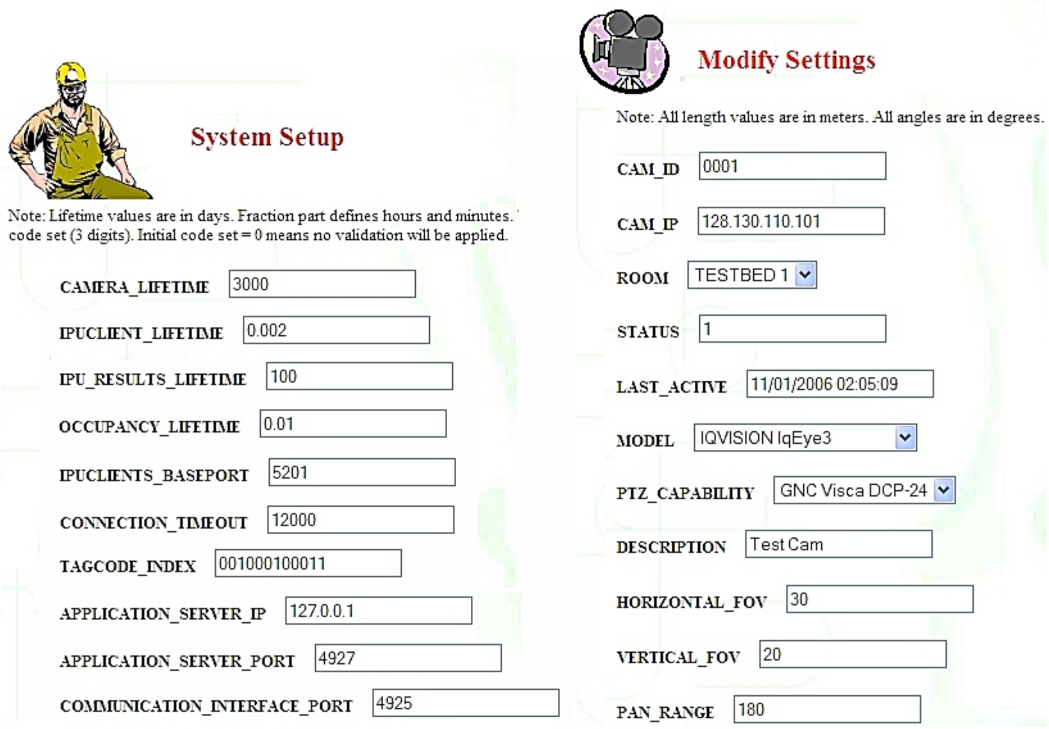
User: admin

(a) (b)

(c) (d)

Figure 59. User interface server, and the camera employed in the test.
(a) Opening screen. (b) User login. (c) Main menu. It can be accessed if the entered user information is valid. Otherwise, the user is directed to a message-screen warning for the invalid input. (d) Camera and P/T device employed in the test.

The test is implemented with an IQeye3 network camera mounted on the Mustang P25 pan-tilt unit (Figure 59d). The box under the camera houses the Visca DCP-24 controller. Firstly, the system and camera parameters are entered using the System Setup and Camera Setup menus as shown in Figure 60.



System Setup

Note: Lifetime values are in days. Fraction part defines hours and minutes.'
code set (3 digits). Initial code set = 0 means no validation will be applied.

CAMERA_LIFETIME: 3000
IPUCLIENT_LIFETIME: 0.002
IPU_RESULTS_LIFETIME: 100
OCCUPANCY_LIFETIME: 0.01
IPUCLIENTS_BASEPORT: 5201
CONNECTION_TIMEOUT: 12000
TAGCODE_INDEX: 001000100011
APPLICATION_SERVER_IP: 127.0.0.1
APPLICATION_SERVER_PORT: 4927
COMMUNICATION_INTERFACE_PORT: 4925

Modify Settings

Note: All length values are in meters. All angles are in degrees.

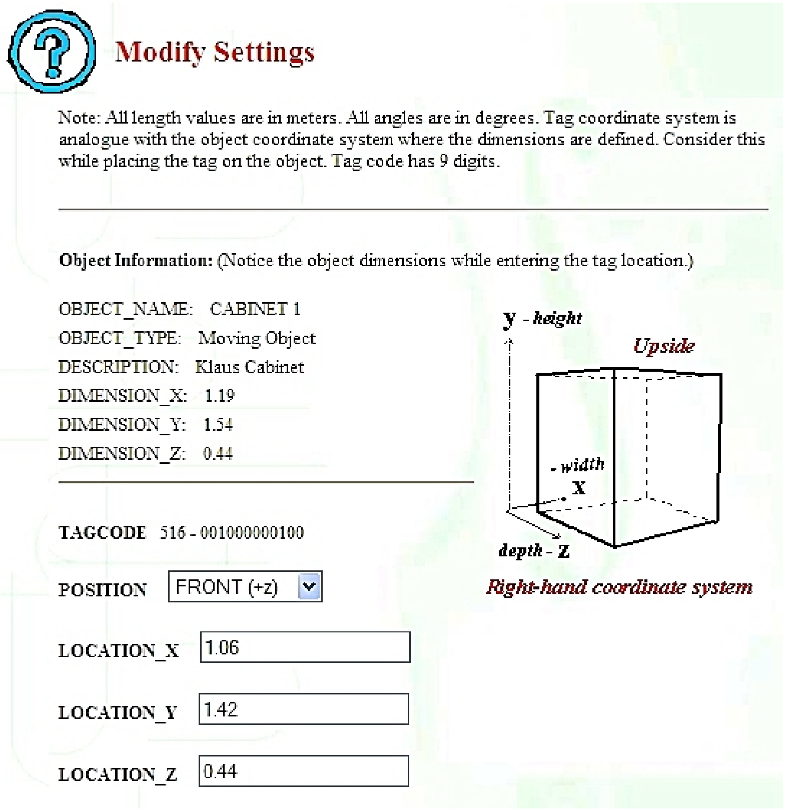
CAM_ID: 0001
CAM_IP: 128.130.110.101
ROOM: TESTBED 1
STATUS: 1
LAST_ACTIVE: 11/01/2006 02:05:09
MODEL: IQVISION IqEye3
PTZ_CAPABILITY: GNC Visca DCP-24
DESCRIPTION: Test Cam
HORIZONTAL_FOV: 30
VERTICAL_FOV: 20
PAN_RANGE: 180

(a) (b)

Figure 60. (a) System parameters setup. (b) Camera parameters setup.

Following the camera, objects in the test space are defined. Towards the implementation of the test, a typical office environment (test-bed) is used that involves 25 objects relevant for the lighting control system. For each object, a tag is generated. The tag codes are assigned automatically by the system, incrementing a base number defined in the System Setup menu. The rest of the tag information is entered by the operator. Consequently, the tags are printed and attached on the corresponding objects. Figure 61 illustrates the definition of an object. Figure 62 shows the list of the objects previously defined in the system. Using this list, available objects and their tags can be tracked, and new tag images can be reprinted, if the original ones on the object are damaged or torn. Figure 63 shows some of the tagged objects in the test-bed, and Figure 64, finally, shows the 2D sketch of the entire test-bed together with the tagged objects and the sensor devices. As also depicted in Figure 64, motion is generated at three points to test the occupancy sensing capability of the system.

The ground-truth data, i.e., actual location information of the objects, are measured before the implementation of the test. Position and orientation of each tagged object resulting out of this measurement are given in Table 3. In the table, orientation is denoted with the normal vectors of the tag planes. This vector represents the orientation of the tag expressed in the room coordinate system.



Modify Settings

Note: All length values are in meters. All angles are in degrees. Tag coordinate system is analogue with the object coordinate system where the dimensions are defined. Consider this while placing the tag on the object. Tag code has 9 digits.

Object Information: (Notice the object dimensions while entering the tag location.)

OBJECT_NAME: CABINET 1
OBJECT_TYPE: Moving Object
DESCRIPTION: Klaus Cabinet
DIMENSION_X: 1.19
DIMENSION_Y: 1.54
DIMENSION_Z: 0.44

TAGCODE 516 - 001000000100

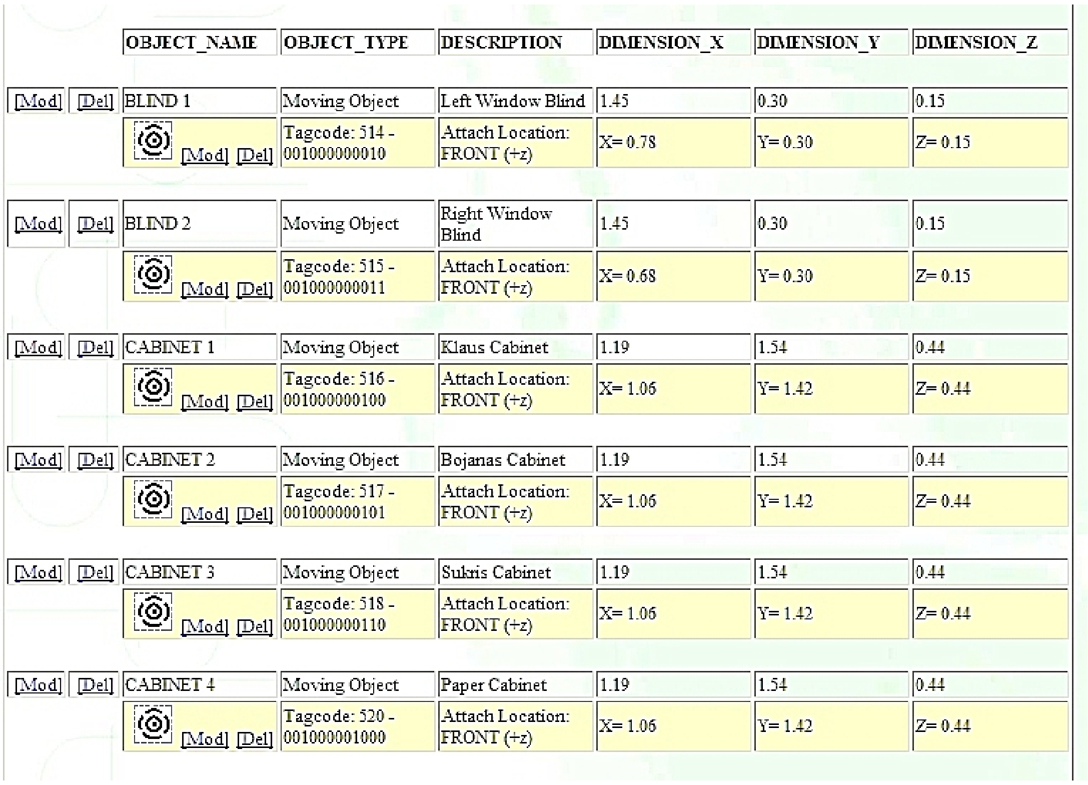
POSITION FRONT (+z)

LOCATION_X 1.06

LOCATION_Y 1.42

LOCATION_Z 0.44

Figure 61. Object setup.



		OBJECT_NAME	OBJECT_TYPE	DESCRIPTION	DIMENSION_X	DIMENSION_Y	DIMENSION_Z
[Mod]	[Del]	BLIND 1	Moving Object	Left Window Blind	1.45	0.30	0.15
		[Mod] [Del]	Tagcode: 514 - 001000000010	Attach Location: FRONT (+z)	X= 0.78	Y= 0.30	Z= 0.15
[Mod]	[Del]	BLIND 2	Moving Object	Right Window Blind	1.45	0.30	0.15
		[Mod] [Del]	Tagcode: 515 - 001000000011	Attach Location: FRONT (+z)	X= 0.68	Y= 0.30	Z= 0.15
[Mod]	[Del]	CABINET 1	Moving Object	Klaus Cabinet	1.19	1.54	0.44
		[Mod] [Del]	Tagcode: 516 - 001000000100	Attach Location: FRONT (+z)	X= 1.06	Y= 1.42	Z= 0.44
[Mod]	[Del]	CABINET 2	Moving Object	Bojanas Cabinet	1.19	1.54	0.44
		[Mod] [Del]	Tagcode: 517 - 001000000101	Attach Location: FRONT (+z)	X= 1.06	Y= 1.42	Z= 0.44
[Mod]	[Del]	CABINET 3	Moving Object	Sukris Cabinet	1.19	1.54	0.44
		[Mod] [Del]	Tagcode: 518 - 001000000110	Attach Location: FRONT (+z)	X= 1.06	Y= 1.42	Z= 0.44
[Mod]	[Del]	CABINET 4	Moving Object	Paper Cabinet	1.19	1.54	0.44
		[Mod] [Del]	Tagcode: 520 - 001000001000	Attach Location: FRONT (+z)	X= 1.06	Y= 1.42	Z= 0.44

Figure 62. A snapshot of the object list.

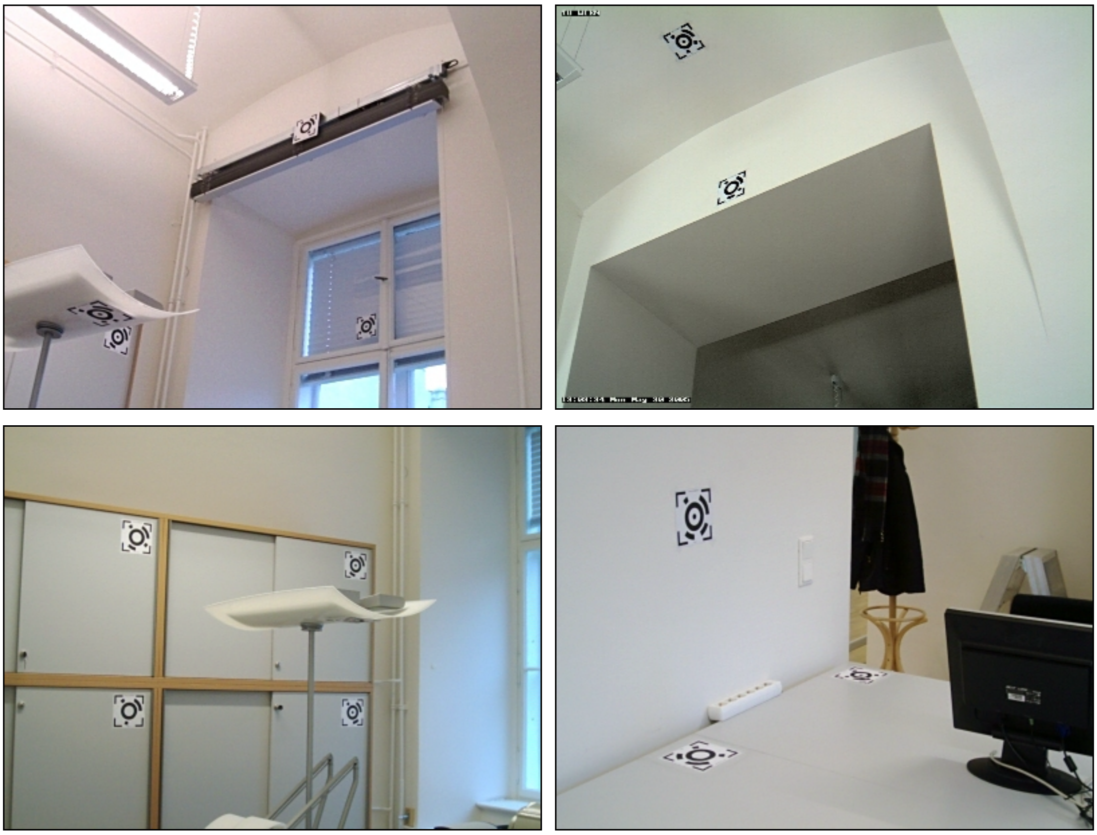


Figure 63. Some of the tagged objects in the test-bed.

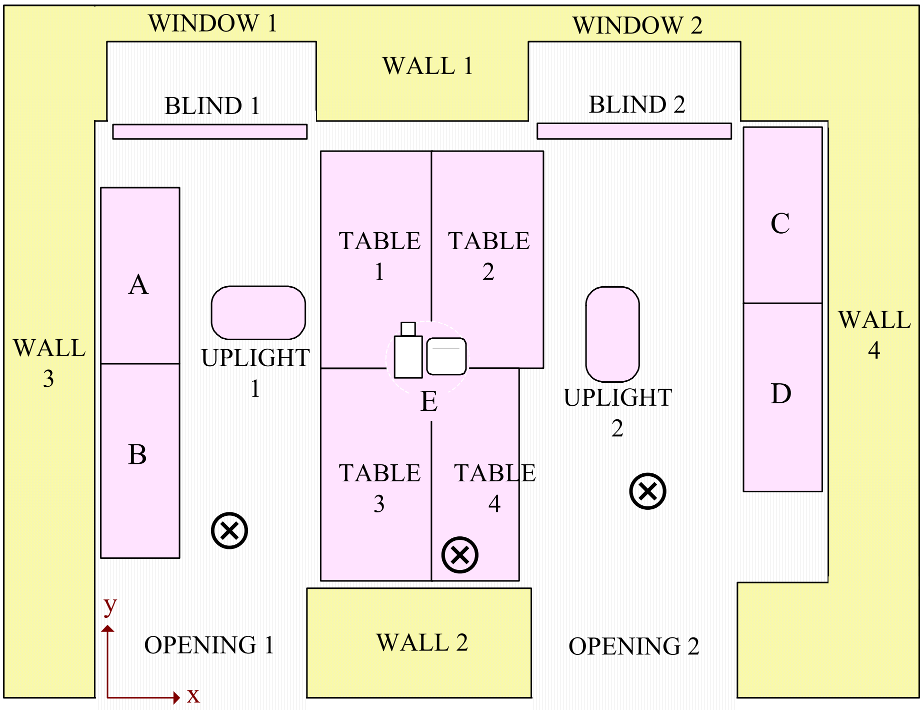


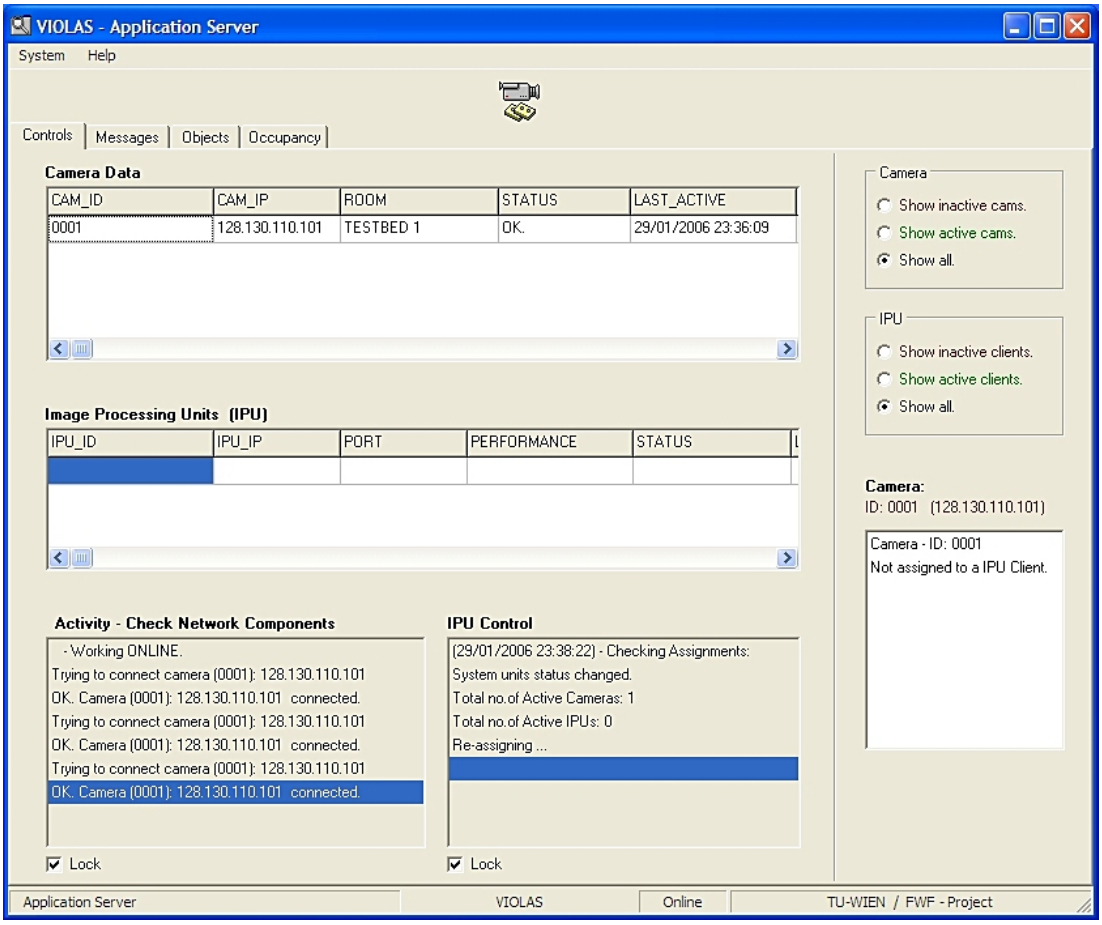
Figure 64. 2D sketch of the test-bed.
“A” refers to Cabinet-3 and Upper-Cabinet-2, “B” refers to Cabinet-4 and Upper-Cabinet-3, “C” refers to Cabinet-1 and Upper-Cabinet-1, “D” refers to Cabinet-2, “E” refers to Camera and P/T unit. The marks, ⊗, refer to the occupancies.

Table 3. Ground-truth data of the objects in the test-bed.
Orientation is denoted with the normal vectors of the tag planes. This vector represents the orientation of the tag expressed in the room coordinate system.

Object Name	Position (m)			Orientation		
	T_x	T_y	T_z	N_x	N_y	N_z
BLIND 1	1.04	4.25	3.38	0	-1	0
BLIND 2	4.03	4.25	3.38	0	-1	0
CABINET 1	5.16	3.1	1.42	-1	0	0
CABINET 2	5.16	1.9	1.42	-1	0	0
CABINET 3	0.44	3.69	1.42	1	0	0
CABINET 4	0.44	2.5	1.42	1	0	0
CEILING	4.1	1.25	4.05	0	0	-1
FLOOR	0.12	0.1	0	0	0	1
TABLE 1	1.95	2.55	0.73	0	0	1
TABLE 2	1.95	0.94	0.73	0	0	1
TABLE 3	3.35	3.94	0.73	0	0	1
TABLE 4	2.75	0.94	0.73	0	0	1
UPLIGHT 1	1.4	2.89	1.75	0	0	-1
UPLIGHT 2	4.24	2.44	1.75	0	0	-1
UPPER CABINET 1	5.16	3.1	2.16	-1	0	0
UPPER CABINET 2	0.44	3.69	2.16	1	0	0
UPPER CABINET 3	0.44	2.5	2.16	1	0	0
WALL 1	2.55	4.33	1.28	0	-1	0
WALL 2	2.55	0.8	1.28	0	1	0
WALL 3	0	1.43	3	1	0	0
WALL 4	5.6	2.8	1.89	-1	0	0
OPENING 1	0.85	0.8	3.5	0	1	0
OPENING 2	4.1	0.8	3.5	0	1	0
WINDOW 1	0.8	5.11	2.6	0	-1	0
WINDOW 2	3.95	5.11	2.6	0	-1	0

11.2 VIOLAS in Operation

Following the measurements, application server is initiated. Application server immediately starts checking the status of the cameras and the IPUs in the system, and finds the IQeye3 camera connected to the network. However, the camera is not assigned to an IPU, since none of them is activated so far (Figure 65). Therefore, an IPU is initiated from a computer in the department. After its initiation, the IPU client registers itself to the system, and responds to the application server's control request. Detecting an active IPU client, application server executes a resource management computation and assigns the netcam to the IPU. For testing the system's behavior with multiple clients, a second IPU is initiated from another computer. This IPU also registers itself successfully to the system, and responds the application server's control request. After the detection of a second active IPU, application server implements a reassignment, degrades the second IPU to slave, and shares the IQeye3 camera among the IPUs. Both of the IPUs successfully run in parallel, and scan the test-bed. They extract the object information in the test-bed, and transfer the results simultaneously to the VIOLAS database. Figure 66 demonstrates the execution of the master IPU driving the pan-tilt unit.



Camera Data

CAM_ID	CAM_IP	ROOM	STATUS	LAST_ACTIVE
0001	128.130.110.101	TESTBED 1	OK.	29/01/2006 23:36:09

Image Processing Units (IPU)

IPU_ID	IPU_IP	PORT	PERFORMANCE	STATUS

Activity - Check Network Components

```
 - Working ONLINE.
Trying to connect camera (0001): 128.130.110.101
OK. Camera (0001): 128.130.110.101  connected.
Trying to connect camera (0001): 128.130.110.101
OK. Camera (0001): 128.130.110.101  connected.
Trying to connect camera (0001): 128.130.110.101
OK. Camera (0001): 128.130.110.101  connected.
```

☑ Lock

IPU Control

```
(29/01/2006 23:38:22) - Checking Assignments:
System units status changed.
Total no.of Active Cameras: 1
Total no.of Active IPUs: 0
Re-assigning ...
```

☑ Lock

Camera
○ Show inactive cams.
○ Show active cams.
● Show all.

IPU
○ Show inactive clients.
○ Show active clients.
● Show all.

Camera:
ID: 0001 (128.130.110.101)

Camera - ID: 0001
Not assigned to a IPU Client.

Application Server | VIOLAS | Online | TU-WIEN / FWF - Project

Figure 65. Application server.

The outputs of the IPUs are fetched by the application server. The incoming results are fused by the internal data fusion unit, and subsequently recorded in the database. By the implementation of the test, system achieves a 100% identification performance, extracting all tag codes and recognizing all objects. The sensed location values are shown in Table 4.

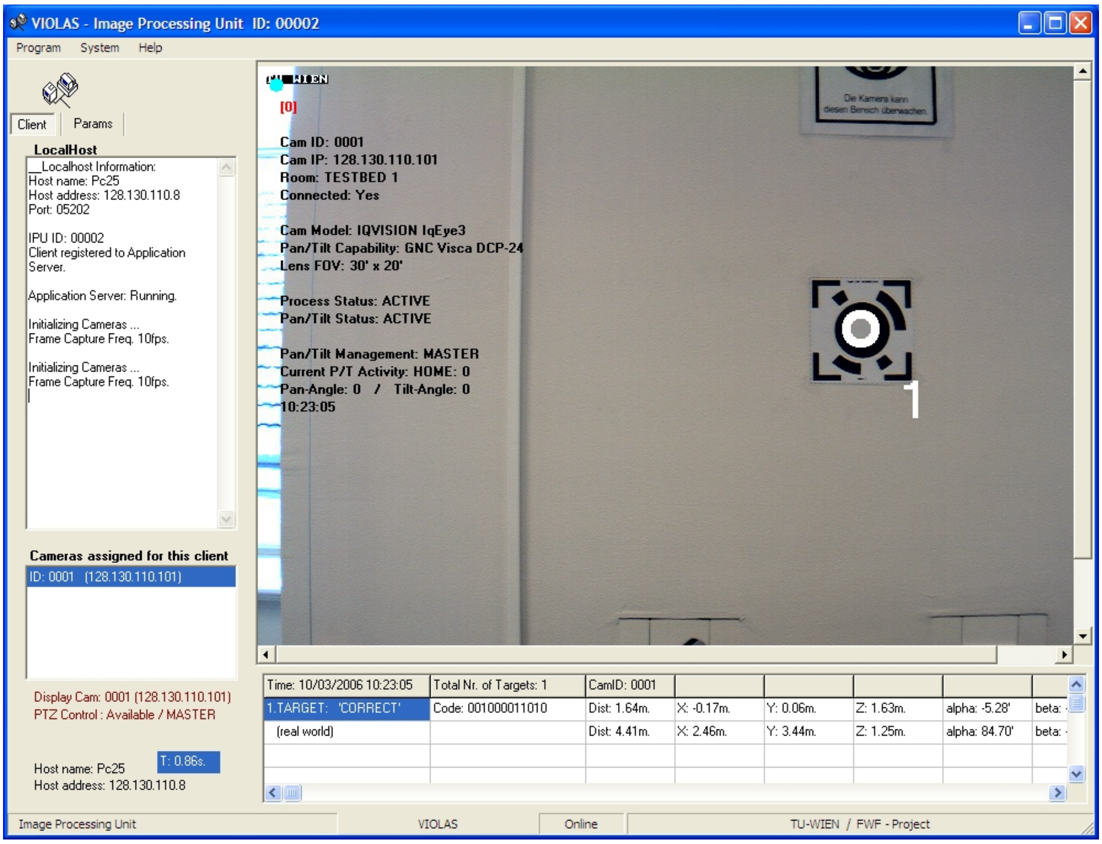
To evaluate the accuracy of location results, “position error” is defined as the distance between the ground-truth position and the sensed position of the tag. “Orientation error” is defined as the angle between the tag’s true surface normal and the sensed surface normal. Thus, the errors are calculated by:

$$Position\ \ Error = \left| \vec{T}' - \vec{T} \right| \qquad \text{Eq.77}$$

$$Orientation\ \ Error\ = \frac{\vec{N}'\cdot\vec{N}}{\left|\vec{N}'\right|.\left|\vec{N}\right|} \qquad \text{Eq.78}$$

where $\vec{T}'$ and $\vec{T}$ refer to the sensed and true positions of the tag, and the vectors, $\vec{N}'$ and $\vec{N}$, refer to the sensed and true orientations of the tag surface respectively. In Table 5, the resulting position and orientation errors are given for our test. The table also includes the position errors in relative terms, i.e., in percentage of camera-tag distance.

In addition to the resulting errors, Table 5 involves the two limitations observed in the system: (1) camera-tag distances, and (2) incidence angles (angles between the normals of the tag and image plane). The camera-tag distances are measured manually with a laser distance-measurer. The incidence angles are calculated with using the related pan and tilt angles as described in Appendix D. In order to observe the system performance with respect to these limitations, the average and maximum position and orientation errors are computed for different camera-tag distance and incidence angle bins, as given in Table 6 and Table 7 respectively.



LocalHost

__Localhost Information:
Host name: Pc25
Host address: 128.130.110.8
Port: 05202

IPU ID: 00002
Client registered to Application Server.

Application Server: Running.

Initializing Cameras ...
Frame Capture Freq. 10fps.

Initializing Cameras ...
Frame Capture Freq. 10fps.

Cam ID: 0001
Cam IP: 128.130.110.101
Room: TESTBED 1
Connected: Yes

Cam Model: IQVISION IqEye3
Pan/Tilt Capability: GNC Visca DCP-24
Lens FOV: 30' x 20'

Process Status: ACTIVE
Pan/Tilt Status: ACTIVE

Pan/Tilt Management: MASTER
Current P/T Activity: HOME: 0
Pan-Angle: 0 / Tilt-Angle: 0
10:23:05

Cameras assigned for this client

ID: 0001 (128.130.110.101)

Display Cam: 0001 (128.130.110.101)
PTZ Control : Available / MASTER

T: 0.86s.

Host name: Pc25
Host address: 128.130.110.8

Time: 10/03/2006 10:23:05	Total Nr. of Targets: 1	CamID: 0001					
1.TARGET: 'CORRECT'	Code: 001000011010	Dist: 1.64m.	X: -0.17m.	Y: 0.06m.	Z: 1.63m.	alpha: -5.28'	beta:
(real world)		Dist: 4.41m.	X: 2.46m.	Y: 3.44m.	Z: 1.25m.	alpha: 84.70'	beta:

Figure 66. Image processing unit

Table 4. Sensed location values of the objects in the test-bed recorded by VIOLAS.

Object Name	Position (m)			Orientation		
	T_x	T_y	T_z	N_x	N_y	N_z
BLIND 1	0.94	4.13	3.32	0.03	-1	-0.01
BLIND 2	4.01	4.28	3.26	0.07	-0.99	0.08
CABINET 1	5.08	3.22	1.31	-1	0.05	0.05
CABINET 2	5.2	2.04	1.31	-0.99	-0.12	-0.02
CABINET 3	0.5	3.6	1.38	1	0.01	0.02
CABINET 4	0.53	2.43	1.36	1	0.02	0.07
CEILING	4.38	1.25	3.79	-0.21	0.12	-0.97
FLOOR	0.26	0.09	-0.07	-0.01	-0.02	1
TABLE 1	1.97	2.54	0.75	0.01	0.05	1
TABLE 2	2.04	1.02	0.72	-0.01	-0.02	1
TABLE 3	3.27	3.9	0.73	0.03	0	1
TABLE 4	2.82	1.07	0.74	-0.01	-0.02	1
UPLIGHT 1	1.51	2.82	1.67	0.04	0.05	-1
UPLIGHT 2	4.19	2.5	1.64	-0.05	-0.03	-1
UPPER CABINET 1	5.17	3.22	1.77	-0.97	0	0.25
UPPER CABINET 2	0.47	3.59	2.07	1	-0.01	0.03
UPPER CABINET 3	0.49	2.41	2.06	1	0.02	0.03
WALL 1	2.49	4.23	1.24	0.06	-1	0.05
WALL 2	2.65	0.91	1.2	0	1	0.07
WALL 3	0.04	1.22	2.91	1	0.1	0.02
WALL 4	5.6	2.97	1.75	-0.99	-0.02	0.11
OPENING 1	1.02	0.65	3.34	-0.05	1	0.06
OPENING 2	4.33	0.9	3.22	-0.03	1	0
WINDOW 1	0.82	4.96	2.47	0.08	-1	0.03
WINDOW 2	3.9	5.13	2.47	0.02	-1	0.03

Table 5. Position and orientation errors of the objects with respective camera distances and incidence angles.

Object Name	Distance to Camera (m)	Incidence Angle (°)	Position Error (m)	(%)*	Orientation Error (°)
BLIND 1	3.01	67	0.17	5.6	1.81
BLIND 2	3.05	48	0.13	4.1	6.13
CABINET 1	2.68	0	0.18	6.8	4.04
CABINET 2	2.50	30	0.18	7.3	7.01
CABINET 3	2.32	30	0.12	5.0	1.28
CABINET 4	2.08	0	0.13	6.2	4.16
CEILING	3.49	56	0.38	10.9	14.00
FLOOR	3.69	72	0.16	4.3	1.28
TABLE 1	0.70	50	0.03	4.3	2.92
TABLE 2	1.76	72	0.12	6.9	1.28
TABLE 3	1.65	72	0.09	5.4	1.72
TABLE 4	1.69	70	0.15	8.8	1.28
UPLIGHT 1	1.21	50	0.15	12.6	3.66
UPLIGHT 2	1.78	50	0.13	7.6	3.34
UPPER CABINET 1	2.83	20	0.41	14.4	14.45
UPPER CABINET 2	2.49	35	0.14	5.5	1.81
UPPER CABINET 3	2.26	20	0.14	6.4	2.06
WALL 1	1.75	0	0.12	7.0	4.47
WALL 2	1.82	0	0.17	9.3	4.00
WALL 3	3.28	48	0.23	7.1	5.82
WALL 4	3.18	20	0.22	6.9	6.44
OPENING 1	3.35	48	0.28	8.3	4.47
OPENING 2	3.34	48	0.38	11.3	1.72
WINDOW 1	3.3	35	0.20	6.0	4.88
WINDOW 2	3.23	35	0.14	4.4	2.06

* Position error in percentage of camera-tag distance.

Table 6. The average and maximum position and orientation errors for different camera-tag distance bins.

Error values	Camera-tag distances (m)				*All*
	0..1	1..2	2..3	3..4	*distances*
Position Error (m)					
AVERAGE	0.03	0.13	0.19	0.23	*0.18*
MAXIMUM	0.03	0.17	0.41	0.38	*0.41*
Position Error (%)					
AVERAGE	4.3	8.2	7.4	6.9	*7.3*
MAXIMUM	4.3	12.6	14.4	11.3	*14.4*
Orientation Error (°)					
AVERAGE	2.9	2.8	5.0	4.9	*4.2*
MAXIMUM	2.9	4.5	14.5	14	*14.5*

Table 7. The average and maximum position and orientation errors for different incidence angle bins.

Error values	Incidence Angles (°)				*All*
	0..19	20..39	40..59	60..79	*angles*
Position Error (m)					
AVERAGE	0.15	0.19	0.21	0.14	*0.18*
MAXIMUM	0.18	0.41	0.38	0.17	*0.41*
Position Error (%)					
AVERAGE	7.3	7.0	8.3	6.2	*7.3*
MAXIMUM	9.3	14.4	12.6	8.8	*14.4*
Orientation Error (°)					
AVERAGE	4.2	5.0	5.3	1.5	*4.2*
MAXIMUM	4.5	14.5	14.0	1.8	*14.5*

As mentioned in the design scheme of VIOLAS, data fusion unit combines the reiterated results by selecting the most up-to-date and certain data as the final, unique location information, and the uncertainty is determined with the camera-tag distance (section 8.4). This judgment is based on the results acquired from Table 6, where the effect of this limitation can be seen on the location accuracy: the error levels increase with the camera distance. Similarly, Table 7 depicts the effects of the incidence angle. However, increase in the incidence angle does not similarly degrade the location accuracy. On the contrary, the system can fit ellipses more properly to the projection of reference circles generated by high incidence angles. This results in the ascension of deviations to some extend, but then causes a discernible decline. The impact of this limitation is observed substantially in the identification accuracy as explained in the following paragraphs. Eventually, based on the test, the system possesses an average position error of 0.18 meters and orientation error of 4.2 degrees on aggregate. The position error percentage has a mean value of 7.3%.

The occupancies in the test-bed are also sensed without any miss and without any false detections. The system can detect the motion of a hand wave within the location sensing range. Additionally, it is robust against illumination changes that occur due to opening or closing of the aperture in the camera mechanism. Likewise, slow changes in the illumination levels of the light sources also do not mislead the system to false occupancies. It must be noted, however, that it takes approximately 2.5 seconds for the system to process one image in the scene. The motions taking place faster than this duration possess the risk of not being able to generate any change in the two successive images, thus they may not be detected. As mentioned before, it is not possible for the system to give an exact location for the occupancy, however, it is possible to state the region within which the motion takes place. A graphical representation of the test-bed, as generated and displayed by the user interface, is illustrated in Figure 67. The object location results can be seen together with the sensed occupancies in the figure.

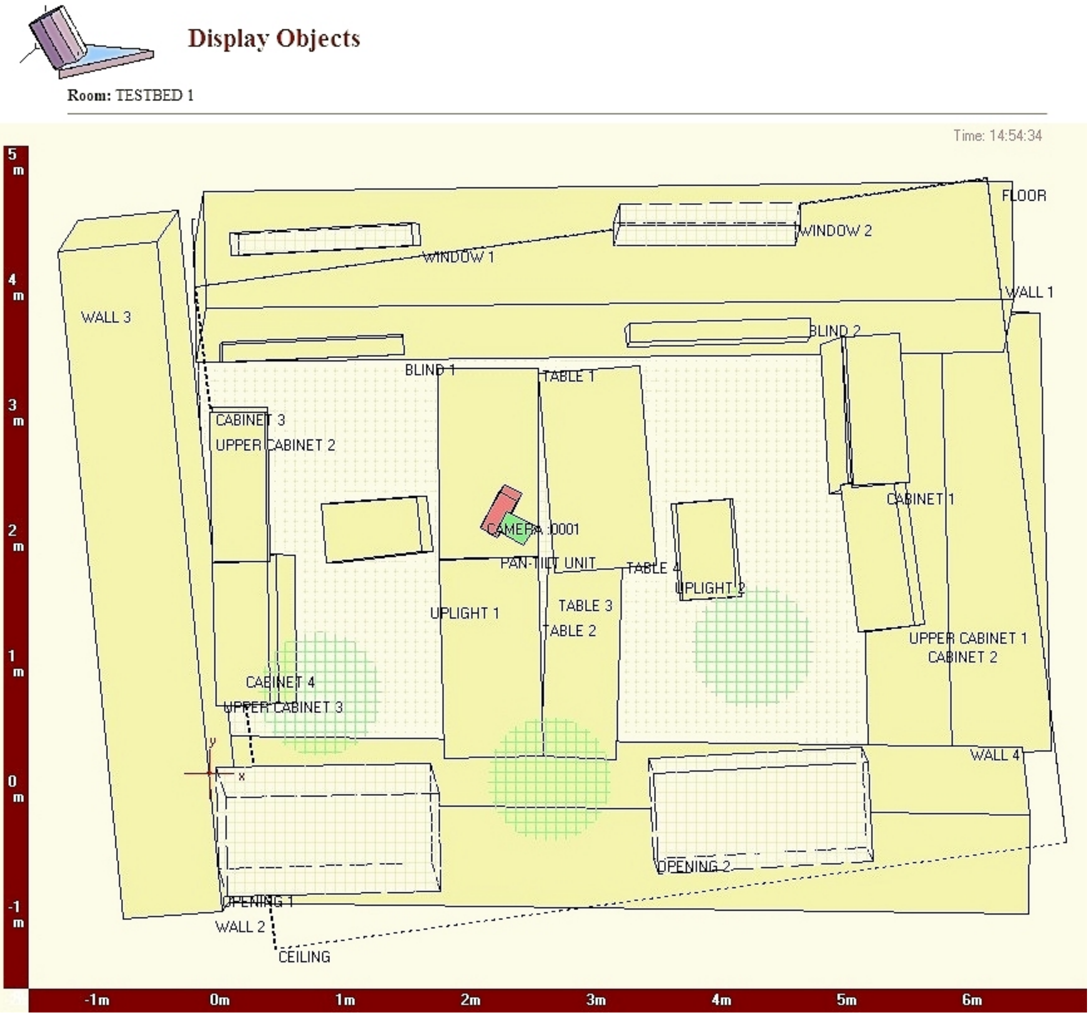


Figure 67. Graphical representation of the test-bed generated by the user interface server. The objects are drawn with the extracted locations after the execution of the test. The slides are caused by orientation errors that have an average value of ~5°. The detected occupancies are also depicted with (green) circles.

Afterwards, the test is re-implemented without the employment of enhancement methods in order to observe their impact. Within this condition, the system is not able to identify some of the tags in the test-bed. The identified and unidentified objects are given together with respect to their corresponding camera-tag distances and incidence angles in Table 8. As seen from the table, the impact of the incidence angle on the identification performance is predominant over the camera-tag distance limitation. The un-enhanced system cannot bring off the identification task especially for the incidence angles beyond 60 degrees. This demonstrates the identification performance augmentation of the enhancement methods. The sensed location results are also shown in Table 9. Based on the values given in the table, the average position error is calculated as 0.20 meters, and orientation error is calculated as 5.1 degrees. The position error percentage has a mean value of 8.4%. The acquired results depict that the enhancement methods also provide a slight positive impact on the location sensing accuracy.

Table 8. Identification results of the test implemented without enhancement methods. The unidentified objects are stroke through in the table.

Incidence	Camera-tag distances (m)				*All*
Angle (°)	0..1	1..2	2..3	3..4	*distances**
0 ..19	—	WALL 1 WALL 2	CABINET 1 CABINET 4	—	*100%*
20..39	—	—	CABINET 2 CABINET 3 UP. CAB. 1 UP. CAB. 2 UP. CAB. 3	WALL 4 ~~*WINDOW 1*~~ WINDOW 2	*88%*
40..59	TABLE 1	UPLIGHT 1 UPLIGHT 2	—	BLIND 2 CEILING WALL 3 OPENING 1 OPENING 2	*100%*
60..79	—	~~*TABLE 2*~~ ~~*TABLE 3*~~ TABLE 4	—	~~*BLIND 1*~~ ~~*FLOOR*~~	*20%*

* The identification performance given in percentage of identified objects for each angle bin.

The specifications of the sensors employed in the test can be given as follows: The IQeye3 netcam is adjusted to 800×600 resolution. The camera possesses 10 mm (36° FOV) lens, f1.6 aperture and 6×6 µm effective pixel size. As mentioned before, the pan-tilt unit's head part is a Mustang P25. This device possesses 0.2 degrees backlash deviation as announced by the manufacturer. Additionally, a Visca DCP-24 is used as the P/T controller. Controller devices also involve errors in positioning the head parts because of some internal mechanisms. A detailed evaluation of the pan-tilt unit used within this test is given in Appendix E. Based on this evaluation, the total maximum rotation error of the pan-tilt unit is measured as 0.8 degrees that generates roughly 4.5 centimeters deviation in 3 meters distance.

Table 9. Location values of the identified objects sensed without enhancement methods.

Object Name	Position (m)			Orientation		
	T_x	T_y	T_z	N_x	N_y	N_z
BLIND 2	4.01	3.5	3.32	0.04	-0.99	0.13
CABINET 1	5.07	2.42	1.34	-0.99	-0.02	0.13
CABINET 2	5.18	1.25	1.31	-0.99	-0.12	-0.03
CABINET 3	0.45	2.79	1.37	1	-0.01	0.01
CABINET 4	0.52	1.62	1.35	0.99	0	0.17
CEILING	4.43	0.41	3.84	-0.14	0.09	-0.99
TABLE 1	1.98	1.74	0.76	0	0.04	1
TABLE 4	2.79	0.21	0.73	-0.01	-0.03	1
UPLIGHT 1	1.53	1.97	1.67	0.04	0	-1
UPLIGHT 2	3.94	2.01	1.64	0	0	-1
UPPER CABINET 1	5.1	2.43	2.03	-1	-0.06	0.07
UPPER CABINET 2	0.44	2.76	2.05	1	0.02	0.03
UPPER CABINET 3	0.49	1.6	2.05	1	0.02	0.06
WALL 1	2.47	3.44	1.25	-0.03	-0.99	0.11
WALL 2	2.63	0.1	1.23	0.02	0.99	0.16
WALL 3	0.06	0.41	2.87	1	0.09	0.03
WALL 4	5.5	2.13	1.78	-0.99	-0.02	0.17
OPENING 1	1.05	-0.11	3.31	-0.03	1	0.09
OPENING 2	4.39	0.04	3.27	-0.05	1	0.04
WINDOW 2	3.83	4.31	2.49	0	-1	0.04

12 CONCLUSIONS

As mentioned in the beginning of the thesis, the implementation of simulation-based control mechanisms is important towards the realization of sentient buildings, and consequently towards the improvement of quality and effectiveness in building services. To provide a proof for the concept, a simulation-based lighting control system is developed in our department. The key point in fitting such control systems in the sentient-buildings concept is to provide self-updating space models with enabling a context awareness capability. For our lighting control system, the main issues of the context awareness are comprised of identifying the objects, sensing their location and detecting the available occupancies. In our efforts to meet these requirements, VIOLAS is developed as described throughout the thesis. The essential points in the development can be concisely summarized as: the selection of proper sensors, i.e., network cameras, the implementation of the sensing algorithms, and the design of the system software architecture.

As given in the previous chapter, software organization and sensing functionalities of VIOLAS are fully tested in an office environment. The natural drawbacks of vision-based solutions are the poor lighting conditions and occlusions that deform the first-hand image information, and prevent the acquisition of a processable visual data. The test platform is constructed without such inconvenient situations. Nevertheless, two additional limitations are observed: the camera-tag distances and incidence angles. Without the employment of image enhancement modules designed to compensate the netcam drawbacks, the system identifies and locates the objects effectively within 3 meters range and 60 degrees incidence angle, and leaves some of the distant objects in the test-bed unidentified. With the addition of these modules, the sensing performance is augmented to 4 meters range and 75 degrees incidence angle, enabling the identification of all objects in the test-bed, and locating them with an accuracy of 0.18 meters average position and 4.2 degrees average orientation error. Eventually, with a single netcam and pan-tilt unit, VIOLAS possesses an effective scanning area of approximately 50 m^2, within which it can simultaneously detect and roughly locate the occupancies as well.

As mentioned in the technology review chapter of the thesis, there are various systems that perform distinct sensing activities in different performance levels. Even among the vision-based solutions, there are different available visual sensors: cheap devices like web cameras, or more expensive and sophisticated solutions like CCTV or network cameras. Within all of these possibilities, the results obtained from VIOLAS suggest that vision-based sensing, when enhanced computationally and integrated with appropriate hardware, is also a promising technology for spatial domains such as facilities and buildings.

In addition to the sensing results, several consequences are inferred from the implementation and overall execution of the system. Based on the software design, the application server of VIOLAS controls the status of the distributed components. It dynamically assigns active netcams to active IPUs in such a manner that the workload is well distributed within the system. This arrangement minimizes the operator overhead, and offers a kind of self-organizing capability by automatically providing a balanced and constant system operation.

Moreover, the distributed structure of VIOLAS provides a configurable system. It enables the utilization of multiple cameras by a single IPU, or a single camera by multiple IPUs. A camera-rich configuration can increase the coverage area of the system, whereas an IPU-rich configuration can augment the system speed. Consequently, the system can be utilized in various configurations, depending on the environmental conditions, response time brevity and financial capabilities.

Thus, it can be concluded that VIOLAS wraps the assorted sensing solutions under a common self-updating platform representing a scalable and configurable structure. We believe this provides a flexible and adaptive system that is highly suited to the requirements of indoor-environmental control applications in the built environment. The self-updating building model, as generated by VIOLAS, can provide, thus, the core of the prototypical implementation of the simulation-based control strategies in sentient buildings.

As for the concurrently ongoing studies, the location results of VIOLAS are processed by spatial reasoning methods towards the rectification of the position and orientation data. This allows for the reconstruction of space models within the lighting simulation context, and the comparison of performance between the as-built and rectified models (Suter et al. 2005). The resulting models are utilized in the studies performed towards the realization of a simulation-assisted lighting control application, where the application can dynamically adjust the position of window blinds and the status of room uplights to achieve user-specific performance levels (Mahdavi et al. 2005).

REFERENCES

ArcSecond. 2004. ArcSecond Inc., http://www.constellation3di.com/

Battiato, S., Castorina, A., Guarnera, M., Vivirito, P. 2003. An adaptive global enhancement pipeline for low cost imaging sensors. IEEE International conference on consumer electronics, Los Angeles, June 2003. 398-399.

Battiato, S., Gallo, G., Stanco, F. 2000. A new edge-adaptive algorithm for zooming of digital images. IASTED Signal Processing and Communications, Marbella, September 2000. 144-149.

Bewator, 2006. Bewator Group, http://www.bewator.com/

Bookstein, F. L. 1979. Fitting conic sections to scattered data. Computer Graphics and Image Processing (9), pp.56-71. 1979.

Brunner, K. A. 2006. Phd dissertation, Vienna University of Technology.

DCOM. 2004. Microsoft COM technologies - Information and resources for the component object model-based technologies, http://www.microsoft.com/com/

Ekahau. 2004. Ekahau Inc., http://www.ekahau.com/

Farin G., Hansford D. 1997. The geometry toolbox for graphics and modelling. A.K. Peters Publishing, ISBN 1-56881-074-1.

Faugeras, O. D. 1993. Three-dimensional computer vision: a geometric viewpoint. Artificial Intelligence Series. MIT Press, Cambridge, USA. ISBN 0-26206-158-9.

Forsyth, D., Mundy, J.L., Zisserman, A., Coelho, C., Heller, A., Rothwell, C. 1991. Invariant descriptors for 3-D object recognition and pose. IEEE Transactions on Pattern Analysis and Machine Intelligence, October 1991; 13(10): 971-991.

Geodata. 2004. GeoData GmbH, http://www.geodata.at/

GNT, 2006. GNT Gumprecht Nachrichtentechnik, http://www.gnt.biz/

Hightower, J., Borriello, G., Want, R. 2000. SpotON: An indoor 3D location sensing technology based on RF signal strength. UW CSE Technical Report 2000-02-02, February 2000

Horn, B. K. P. 1986. Robot vision, Reflectance Map: Photometric Stereo. The MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, USA. ISBN 0-26208-159-8, 1986. pp 209-213.

İçoğlu, O., Mahdavi, A. 2005. A vision-based sensing system for sentient building models, 22nd CIB-W78 Conference – Information Technology in Construction, Dresden, Germany.

Iqinvision, 2006. IQinVision Inc., http://www.iqeye.com/

Krumm, J., Harris, S., Meyers, B. 2000. Multi-camera multi-person tracking for EasyLiving. Proceedings of 3rd IEEE International Workshop on Visual Surveillance, July 1, 2000, Dublin, Ireland

Lòpez de Ipina, D. 2002. Visual sensing and middleware support for sentient computing. PhD dissertation, University of Cambridge.

Lòpez de Ipina, D., Mendonca, P. S., Hopper, A. 2002. Visual sensing and middleware support for sentient computing. Personal and Ubiquitous Computing. Volume 6, Issue 3, ISSN: 1617-4909. pp. 206–219.

Mahdavi, A. 2001a. Aspects of self-aware buildings. International Journal of Design Sciences and Technology. Europia: Paris, France.Volume 9, Number 1. ISSN 1630 - 7267. pp. 35 - 52.

Mahdavi, A. 2001b. Simulation-based control of building systems operation. Building and Environment. Volume 36, Issue 6, ISSN: 0360-1323. pp. 789-796.

Mahdavi, A. 2003a. Computational building models: Theme and four variations (Keynote). Proceedings of the Eight International IBPSA Conference, Eindhoven, Netherlands, 2003. Augenbroe, G. - Hensen, J. (eds). ISBN 90 386 1566 3. Vol. 1. pp. 3-18.

Mahdavi, A. 2003b. Modell-basierte Steuerungsstrategien für selbstbewusste Gebäude. Gesundheits-Ingenieur. Oldenbourg Industrieverlag. Munich, Germany. Heft 5 2003. ISSN 0932-6200. pp. 234 - 244.

Mahdavi, A., Suter, G. 2002. Sensorgestützte Modelle für verbesserte Gebaeudeservices. FWF proposal.

Mahdavi, A., Spasojević, B., Brunner, K. A. 2005. Elements of a simulation-based daylight responsive illumination systems control in buildings. Building Simulation '05, International Building Performance Simulation Assosication, IBPSA, Montreal, Canada.

Mitsubishi. 2004. Mitsubishi Electric Research Laboratories, http://www.merl.com/projects/visual-tags/

Ni, L. M., Liu, Y., Lau, Y. C., Patil, A. P. 2003. LANDMARC: indoor location sensing using active RFID. Proceedings of the 2003 IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2003), Dallas, Texas, USA.

Pal, V., Mahdavi, A. 1999. A comprehensive approach to modeling and evaluating the visual environment in buildings. Building Simulation. International IBPSA Conference. Kyoto, Japan. Vol. II. ISBN 4-931416-02-0. 1999. pp. 579 - 586.

Pentax, 2006. Pentax Imaging, http://www.pentaxtech.com/

Pilu M, Fitzgibbon A, Fisher R. 1996. Ellipse-specific direct least-square fitting. IEEE International Conference on Image Processing.

Polhemus. 2004. Polhemus Inc., http://www.polhemus.com/

Priyantha, N. B., Chakraborty, A., Balakrishnan, H. 2000. The Cricket location-support system. Proc. of the Sixth Annual ACM International Conference on Mobile Computing and Networking (MOBICOM).

Ryburg, J. 1996. New churn rates: people, walls, and furniture in restructuring companies. Facility Performance Group, Inc.

Sonitor. 2004. Sonitor Technologies AS, http://www.sonitor.com/

Spasojević, B., Mahdavi, A. 2005. Sky luminance mapping for computational daylight modeling. Building Simulation '05, International Building Performance Simulation Assosication, IBPSA, Montreal, Canada.

Suter, G., İçoğlu, O., Mahdavi, A., Spasojević, B. 2005. Position uncertainty in space scene reconstruction for simulation-based lighting control. Building Simulation '05, International Building Performance Simulation Assosication, IBPSA, Montreal, Canada.

Tanenbaum, A. S. 1992. Modern operating systems, Interprocess communication. New Jersey: Prentice Hall Inc., ISBN 0-13595-752-4, 1992. pp. 100-123.

Tanenbaum, A. S. 2002. Computer networks. New Jersey: Prentice Hall Inc., ISBN 0-13-038488-7.

Toth, D., Aach, T., Metzler, V. 2000. Illumination-invariant change detection, 4th IEEE Southwest Symposium on Image Analysis and Interpretation.

Trucco, E., Verri, A. 1998. Introductory techniques for 3-D computer vision. New Jersey: Prentice Hall Inc. ISBN 0-13-261108-2.

Ward, A. 1998. Sensor-driven computing. PhD dissertation, University of Cambridge.

Werb, J., Lanzl, C. 1998. Designing a positioning system for finding things and people indoors. IEEE Spectrum, Vol. 5, Issue 9, Sep. 1998, pp. 71–78

Wellner P. 1993. Interacting with paper on the DigitalDesk. Communications of the ACM, August 1993; 36(7): 87-96

WhereNet. 2004. WhereNet, http://www.wherenet.com/

Wildes, R. P. 1998. A measure of motion salience for surveillance applications. In Proceedings of the IEEE International Conference on Image Processing, 183-187.

Appendix A

CONVERTING IMAGES FROM COLOR TO GRAY-SCALE

All of the image processing methods described in this dissertation (identification and location sensings, image enhancements, occupancy sensing) are applied to gray scale images. The intensity (brightness) values of the pixels in the gray scale images are represented with 8–bits, ranging from 0 to 255. The colors demonstrate a transition from black to white, where the 0 value corresponds the black, the 128 value corresponds to absolute gray and 255 corresponds to white.

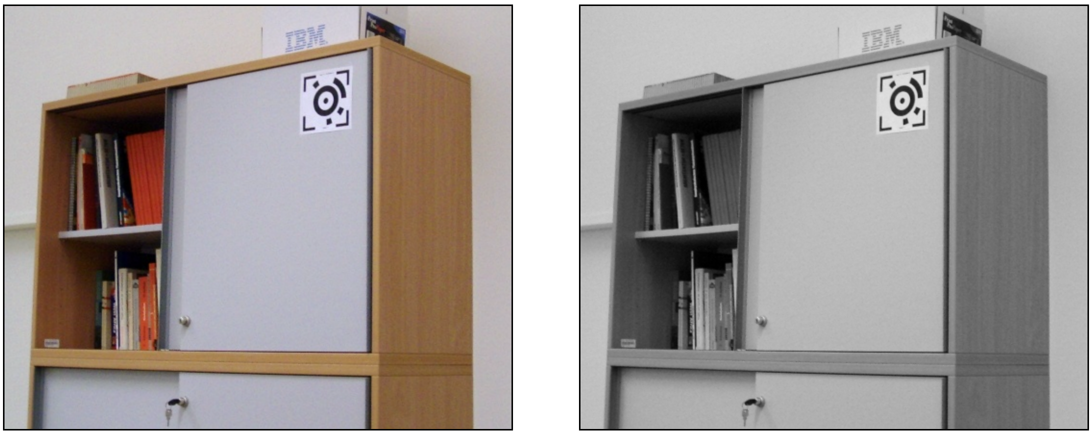
However, the images captured from the cameras are represented with a RGB color space. Color spaces are a way of orginizing the colors perceived by the human beings. The RGB color space consists of the three additive primaries: red, green, and blue. Spectral components of these colors combine additively to produce a resultant color. The pixel values are represented with 24–bits; 8–bits per color channel. Red is defined by $(255,0,0)$, green by $(0,255,0)$, and blue by $(0,0,255)$. The combination of these components are sufficient to define almost all colors we perceive, such as $(255,255,0)$ that gives the yellow color.

Since the color is a perceived phenomenon, there is no closed form conversion from RGB space to gray scale. However, there are some predefined standards. The method used in this research is the standard set by the American National Television Systems Committee (NTSC). With respect to this standard, the RGB values of an image are converted to gray scale as follows:

$$Gray\ scale\ \ intensity\ =\ 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \qquad \text{Eq.79}$$

Please note that the shooting camera's being color or black-and-white does not affect the conversion. In a black-and-white camera, the pixel values are still defined in RGB space. These values are given by (x,x,x), where $x = 0 \ldots 255$. The result of the conversion, in this case, will be x for each pixel. The conversion does not create any difference for the viewing eye, but the pixels' representation in the image changes.

In Figure 68, a sample conversion is illustrated. Figure 68a shows an image captured by the network camera used in the project. The pixels of the image are defined in RGB space. Each pixel is represented with 24–bits. Since the network camera is a color one, the different RGB combinations can be seen in the captured image. Figure 68b illustrates the result of the conversion. The pixels are defined in gray scale, and represented by 8–bits in this image.



(a) (b)

Figure 68. Converting RGB images to gray scale.
(a) Pixels in the image are defined in RGB space. Each pixel is represented by 24-bits. (b) Pixels are defined in gray scale. Each pixel is represented by 8-bits.

Appendix B

DIRECT LEAST SQUARES ELLIPSE FITTING

The direct least squares method fits an ellipse to at least six given points, and always yields to *one and only one* elliptical solution. The method is described in Pilu 1996 in detail. In this section, its implementation is explained concisely.

The general ellipse equation defined as a second order polynomial is given by:

$$F(A, X) = A \cdot X \; = \; ax^2 + bxy + cy^2 + dx + ey + f \; = \; 0 \qquad \text{Eq.80}$$

where A is a 6×1 matrix, $A = [a \quad b \quad c \quad d \quad e \quad f]^T$, and X is a 1×6 matrix, $X = [x^2 \quad xy \quad y^2 \quad x \quad y \quad 1]$. The algebraic distance of a point X_i to the ellipse $F(A, X) = 0$ is given by:

$$F(A, X_i) = d \qquad \text{Eq.81}$$

In the least square fitting method, the algebraic distances over the set of N data points have to be minimized:

$$A = \min_A = \left[\sum_{i=1}^{N} F(A, X_i)^2 \right] \qquad \text{Eq.82}$$

This minimization problem can be transformed into the following form, and solved by applying a quadratic constraint on the ellipse parameters (Bookstein 1979):

$$D^T \cdot D \cdot A \; = \; S \cdot A \; = \lambda \cdot C \cdot A \qquad \text{Eq.83}$$

where $D = [x_1 \quad x_2 \quad x_3 \quad \ldots \quad x_n]^T$ is called the design matrix ($N \times 6$). Indices of the design matrix are the known edge points, on which we want to fit the ellipse. $S = D^T \cdot D$ is the known scatter matrix (6×6) formed by the design matrix. C is a constant matrix (6×6) (determined by Pilu 1996) that gives *one and only one* elliptical solution when applied to Eq83:

$$A^T \cdot C \cdot A \; = b^2 - 4ac \; = -1 \qquad \text{Eq.84}$$

So, the constraint matrix, C, can be defined as:

$$C = \begin{bmatrix} 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Eq.85}$$

Now, Eq83, $S \cdot A = \lambda \cdot C \cdot A$, can be solved with these known values. Scatter matrix, S, is a symmetric matrix, and can be decomposed as given by:

$$S = L \cdot L^T \qquad \text{Eq.86}$$

Thus, Eq83 becomes:

$$L \cdot L^T \cdot A = \lambda \cdot C \cdot A \qquad \text{Eq.87}$$

If we name $L^T \cdot A = V$, then $A = \left(L^T\right)^{-1} \cdot V$

$$L \cdot V = \lambda \cdot C \cdot \left(L^T\right)^{-1} \cdot V$$

$$V = \lambda \cdot L^{-1} \cdot C \cdot \left(L^T\right)^{-1} \cdot V$$

If we name $L^{-1} \cdot C \cdot \left(L^T\right)^{-1} = M$

$$V = \lambda \cdot M \cdot V$$

$$M \cdot V = (1/\lambda) \cdot V \qquad \text{Eq.88}$$

We can proceed with naming $1/\lambda$ as the new scalar, λ:

$$M \cdot V = \lambda \cdot V \qquad \text{Eq.89}$$

Now, V and λ become eigenvectors and eigenvalues of the known 6×6 matrix, M. Matrix A can be computed from $A = \left(L^T\right)^{-1} \cdot V$. However, this equation has six eigenvalue-eigenvector pairs. The one, (λ_i, A_i), that solves Eq83 must have the same sign with the constraint (Pilu 1996):

$$sign(\lambda_i) = sign\left(A^T \cdot C \cdot A\right) \qquad \text{Eq.90}$$

So, the eigenvalue-eigenvector pair for $\lambda_i < 0$ gives the parameters of the fitting ellipse: $A = [a \quad b \quad c \quad d \quad e \quad f]^T$

From the general ellipse equation, we can reach to parametric ellipse equation:

$$\begin{aligned} x &= x_0 + a.\cos(\theta).\cos(t) + b.\sin(\theta).\sin(t) \\ y &= y_0 - a.\sin(\theta).\cos(t) + b.\cos(\theta).\sin(t) \end{aligned} \qquad \text{Eq.91}$$

where (x_0, y_0) is the centre point, a, b are the axes and θ is the orientation with respect to x-axis in clockwise direction.

Appendix C

3D TRANSFORMATIONS

3D transformations map each point in 3D space to a potentially different point in the same 3D space. Some type of transformations may be rotation, translation, scaling, shearing and reflection. In some situations, several types of transformations may be restricted, for example, when modeling a solid object, it can move (translate) and rotate, but scaling, shearing or reflection may not be valid.

When simulating solid 3D objects, some means of specifying, storing and calculating the orientation and subsequent rotations of the object are needed. Rotational quantities are more difficult to represent than linear quantities. One method of holding this information is not suitable for all needs, therefore there are different ways to specify and perform this rotation. These methods include defining Euler angles, axis and angle, quaternions and matrices. Representing transformations with matrices allows to define rotation, translation and scaling. In transforming tag coordinates to camera coordinates, or camera coordinates to room coordinates, translation and rotation are the transformations that have to be considered. Towards this end, the matrix representation of the transformations is utilized within the calculations in this thesis.

Translation in 3D

A vector with three dimensions (in other words, a 3×1 matrix) can represent a physical quantity, which is directional, such as position, velocity, acceleration, force, or momentum. If the vector represents a point in space, these three numbers represent the position in the x, y and z coordinates, where x, y and z are mutually perpendicular axes in some agreed direction and units.

A three-dimensional vector may also represent a displacement in space, such as a translation in some direction. Translation is actually the shifting of the coordinates with given vector values (Figure 69).

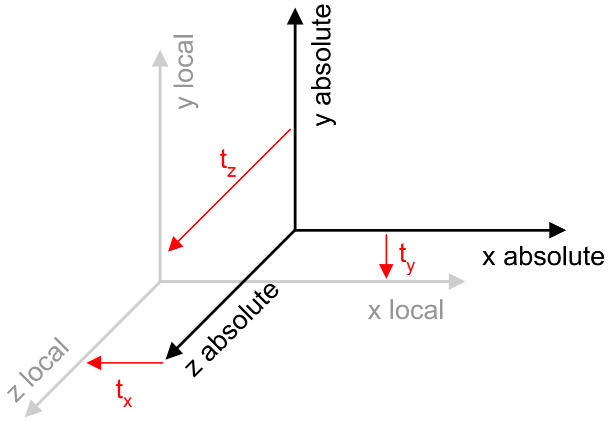


Figure 69. Translation in 3D.

Therefore, translation is applied by the addition of the translation vector to each point on the solid object:

$$P' = P + \vec{T} \qquad \text{Eq.92}$$

where $P = [p_x \quad p_y \quad p_z]^T$ is a point on the solid object represented by a vector in the object's local coordinates (^{T} represents the matrix transpose), $\vec{T} = [t_x \quad t_y \quad t_z]^T$ is the translation vector, and $P' = [p'_x \quad p'_y \quad p'_z]^T$ is the new location of the point after the translation represented by a vector in absolute coordinates.

Combining Translations

Two successive translations, $\vec{T_1}$ and $\vec{T_2}$, can be combined by adding their vectors, $\vec{T} = \vec{T_1} + \vec{T_2}$. So,

$$\vec{T} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} t_{1_x} + t_{2_x} \\ t_{1_y} + t_{2_y} \\ t_{1_z} + t_{2_z} \end{bmatrix} \qquad \text{Eq.93}$$

The resulting vector, $\vec{T}$, can be used as the final translation vector.

Rotation in 3D

Rotations can be represented with 3×3 orthogonal matrices. The matrix, A, is orthogonal, if:

$$A \cdot A^T = 1 \qquad \text{Eq.94}$$

In 3D rotation, there are three degrees of freedom: azimuth, elevation, and tilt (Figure 70). In some areas, different terms may be used such as roll, pitch and yaw.

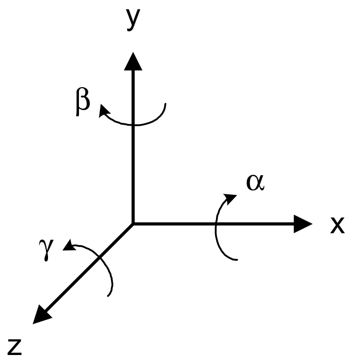


Figure 70. Rotation in 3D.

Azimuth, α, is the rotation around x–axis, elevation, β, is the rotation around y–axis, and tilt, γ, is the rotation around z–axis. The positive values of the angles represent a rotation direction performed by the right hand while closing the fingers inside the palm. The negative values represent a rotation in the opposite direction. This notation is called the *right-hand* rule.

In general, rotation around x–axis with α degree, R_x, rotation around y–axis with β degree, R_y, and rotation around the z–axis with γ degree, R_z, are given by the following matrices:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.95}$$

The rotation around an axis is applied by the multiplication of the corresponding rotation matrix to each point on the solid object:

$$P' = R(\theta) \cdot P \qquad \text{Eq.96}$$

where, $P = [x \quad y \quad z]^T$ is a point on the solid object represented by a vector in the object's local coordinates, $R(\theta)$ is a rotation matrix which is a function of angle θ, and P' is a point moving with respect to the angle θ, and represented by a vector in absolute coordinates.

Combining Rotations

Successive rotations can be calculated by multiplying together the matrices representing the individual rotations. In the same way that the order of rotations is important, the order of matrix multiplications is important.

In other words, the rotation operation is not commutative. The order of successive rotations is significant. For example: (1) rotation around x–axis with 90 degrees, (2) rotation around y–axis with 90 degrees, and (3) rotation around x–axis with -90 degrees eventually gives 90 degree rotation around z–axis. However, keeping the first rotation but replacing the second rotation with the third, gives 90 degrees rotation around y–axis (first two rotations cancel out).

A 3D rotation is defined by the sequential implementation of the rotations around each axis given above: R_x, R_y, and R_z. So, $R = R_x \cdot R_y \cdot R_z$ is a three-dimensional rotation ordered by firstly the z–axis, γ, then y–axis, β, and finally x–axis, α.

$$R = \begin{bmatrix} \cos\beta.\cos\gamma & -\cos\beta.\sin\gamma & \sin\beta \\ \sin\alpha.\sin\beta.\cos\gamma + \cos\alpha.\sin\gamma & -\sin\alpha.\sin\beta.\sin\gamma + \cos\alpha.\cos\gamma & -\sin\alpha.\cos\beta \\ -\cos\alpha.\sin\beta.\cos\gamma + \sin\alpha.\sin\gamma & \cos\alpha.\sin\beta.\sin\gamma + \sin\alpha.\cos\gamma & \cos\alpha.\cos\beta \end{bmatrix} \qquad \text{Eq.97}$$

There are three degrees of freedom, and three axis values which should make 3×3 different combinations to move one point to another location. However, because of the orthogonality constraint, there six different combinations (six different α, β, γ values) to rotate one point to another in 3D.

Coordinate Transformation

As mentioned above, 3D transformations map each point in 3D space to a potentially different point in 3D space defined with the same coordinate system. However, in the coordinate transformation, the situation is vice versa; same point is mapped by a different 3D space. The purpose is to figure out the coordinate values of the "same point" in the "new coordinate system". In order to achieve this, the translation and rotation values between the two coordinate systems must be known.

As an example, it can be assumed that a 3D point defined in a camera coordinate system is being transformed to the harboring room's coordinate system. The 3D transformation between the coordinate systems is as follows: the camera can be moved onto the origin of the room coordinate system (with the corresponding axes overlapping each other) by rotating it γ, β, α degrees (order of rotation is important), and translating it T_x, T_y, T_z meters. Please note that the rotation and translation values are defined with respect to the camera's coordinate system.

In this case, a point's camera-based coordinates, P_{cam}, can be transformed into the room-based coordinates, P_{room} with the following equation:

$$P_{room} = R_{cam \to room}{}^{-1} \cdot P_{cam} - \vec{T}_{cam \to room} \qquad \text{Eq.98}$$

where $R_{cam \to room} = R(\alpha, \beta, \gamma)$, and $\vec{T}_{cam \to room} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T$. The equation means that, the points (defined by the camera's system) must be rotated and translated in the inverse of the direction that is used to bring the camera back to the origin of the room, so that new coordinates of the point can be found.

Thus, the equation can be reformed in a more understandable way by defining a new rotation, $R_{room \to cam} = R_{cam \to room}{}^{-1}$, and a new translation, $\vec{T}_{room \to cam} = -\vec{T}_{cam \to room}$. Finally, the equation becomes:

$$P_{room} = R_{room \to cam} \cdot P_{cam} + \vec{T}_{room \to cam} \qquad \text{Eq.99}$$

where the rotation and translation are defined from room to camera, rather than the one used in the initial 3D transformation, from camera to room. Therefore, the 3D transformation, i.e., translation and rotation, that brings the room reference frame onto the camera, performs the coordinate transformation from camera to room reference frames.

All the methods described above can also be applied in 2D space. Towards this end, the transformations must be reduced to two dimensions with removing the absent axis. For translation, two dimensional vectors are used, whose elements correspond to the relevant axes of the coordinate system. Similarly, for rotation, 2×2 matrices are utilized. The rotation is applied with respect to the axis orthogonal to the 2D frame.

Appendix D

EXTRACTING INCIDENCE ANGLES

Incidence angle is the angle between the normals of the target (tag) and image (camera) plane. This value is important for the evaluation of VIOLAS performance, since it is one of the main limitations in object identification. Towards the evaluation of the system, a demonstrative test is implemented as given in chapter 11. This section describes how the incidence angles are computed for the tags used in the test. A sample tag viewed by the camera from a particular pan-tilt angle is demonstrated in Figure 71.

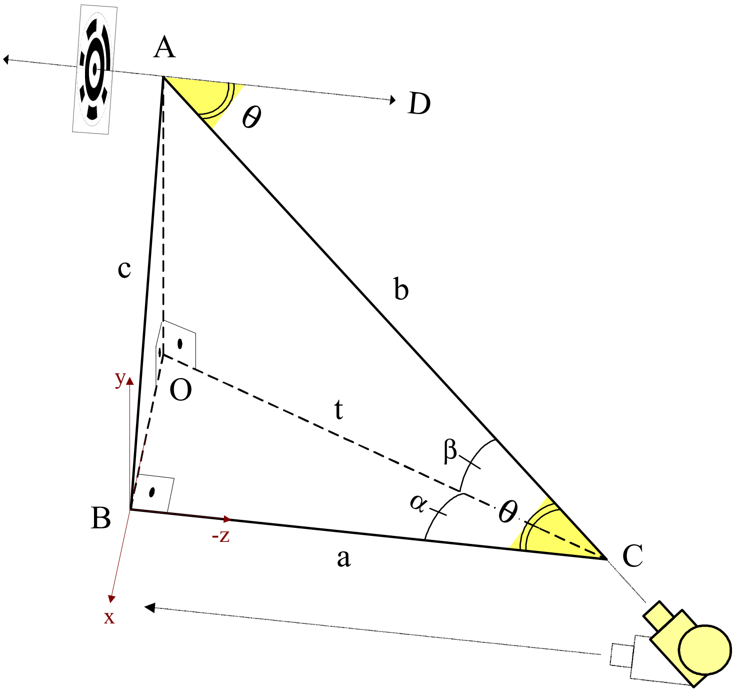


Figure 71. Extracting the incidence angle.

As given in the figure, the lines, $|CA|$ and $|AD|$, represent the normals of the camera and tag plane respectively. The incidence angle represented by θ lies on the corner, DAC, and it is identical with the angle lying on the C corner of the triangle, ABC. This triangle is a facet of the pyramid $OABC$ formed by the pan and tilt angles as described in the following. The pan rotation performed around y–axis is represented by angle α, and the tilt rotation performed around x–axis is represented by angle β in the figure. The right triangle, OBC, is generated by the pan angle, and lies on the xz–plane. The right triangle, OAC, is generated by the subsequent tilt angle, and lies on the plane that is formed with rotating the yz–plane by α degrees around y–axis. Taking the OBC triangle as basis and OAC triangle as a facet, the triangular-pyramid, $OABC$ can be formed.

In order to find the θ angle, the law of cosines is applied to the ABC triangle:

$$\cos\theta = \frac{a^2 + b^2 - c^2}{2ab} \tag{Eq.100}$$

The values, a, b and c represent the corresponding edges of the triangle as also stated in Figure 71. These values are computed with using the pan and tilt angles, α and β, as follows:

In OBC right triangle:

$$a = t \cdot \cos\alpha \tag{Eq.101}$$

$$|OB| = t \cdot \sin\alpha \tag{Eq.102}$$

In OAC right triangle:

$$b = t/\cos\beta \tag{Eq.103}$$

$$|OA| = t \cdot \tan\beta \tag{Eq.104}$$

By using the hypotenuse law in AOB right triangle:

$$c^2 = |OA|^2 + |OB|^2 \tag{Eq.105}$$

Thus, combining Eq105 with Eq102 and Eq104:

$$c = t \cdot \sqrt{\sin^2\alpha + \tan^2\beta} \tag{Eq.106}$$

If we apply the results obtained in Eq101, Eq103 and Eq106 to Eq100:

$$\cos\theta = \frac{t^2 \cdot \cos^2\alpha + \dfrac{t^2}{\cos^2\beta} - t^2 \cdot \sin^2 - t^2 \cdot \tan^2\beta}{2 \cdot t^2 \cdot \dfrac{\cos\alpha}{\cos\beta}} \tag{Eq.107}$$

$$\cos\theta = \frac{t^2 \cdot \left(\cos^2\alpha + \dfrac{1}{\cos^2\beta} - \sin^2\alpha - \dfrac{\sin^2\beta}{\cos^2\beta}\right)}{2 \cdot t^2 \cdot \dfrac{\cos\alpha}{\cos\beta}} \qquad \text{Eq.108}$$

The unknown t values cancel out each other:

$$\cos\theta = \frac{\left(\cos^2\alpha - \sin^2\alpha\right) + \dfrac{1-\sin^2\beta}{\cos^2\beta}}{2 \cdot \dfrac{\cos\alpha}{\cos\beta}} \qquad \text{Eq.109}$$

$$\cos\theta = \frac{\left(2 \cdot \cos^2\alpha - 1\right) + 1}{2 \cdot \dfrac{\cos\alpha}{\cos\beta}} \qquad \text{Eq.110}$$

$$\theta = \arccos\left(\cos\alpha \cdot \cos\beta\right) \qquad \text{Eq.111}$$

Thus, the incidence angle, θ, is acquired in terms of pan and tilt angles, α and β. For all the tags in the test-bed, the incidence angles are computed similarly, by using the corresponding pan and tilt values of the viewing camera.

Appendix E

RESOLUTION AND ERROR ESTIMATION FOR PAN-TILT UNITS

Pan-tilt units manage the rotation control with a special type of embedded sensors: potentiometers. Potentiometer is an electromechanical device that converts mechanical information into an electrical signal. It is commonly used as a position sensor in servo systems, having a terminal connected to each end of a restrictive element, and a third connected to a wiper contact. The output is a voltage that is variable depending upon the position of the wiper contact (Figure 72).

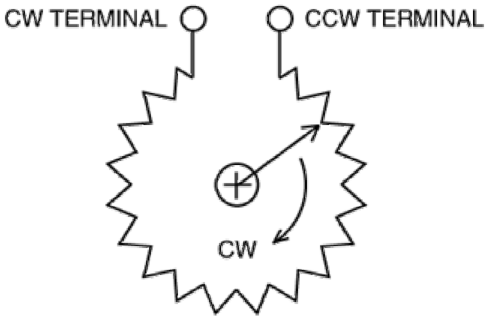


Figure 72. A simple potentiometer.

In Visca DCP-24 controller, signals of the potentiometer are digitized by the $12-$bit A/D converter embedded inside the unit. Theoretically, for a range of $360°$, a mechanical resolution of $360°/4096 = 0.088°$ should be achieved ($2^{12} = 4096$ steps). However, in practice, the entire area of the potentiometers cannot be used. Therefore, the maximum and the minimum voltage levels (0 and 5 volts) at the A/D converter cannot be reached. As an example, for a voltage deviation of 0.5 to 4.5 volts, a loss of 1 volt ($1/5$ of the total voltage range) occurs, and this results in the loss of $4096/5 = 819$ steps. Thus, a mechanical resolution of $360°/(4096-819) = 0.11°$ can be achieved in this example.

In the Visca controller, 3168 steps are reserved for pan positioning, and 2160 steps are reserved for tilt positioning. Even though the P/T head part has an announced swiveling capability of $365°$ pan and $180°$ tilt, its potentiometers' angular lengths are measured as $368°$ for pan positioning, and $188°$ for tilt positioning. This results in a pan resolution of $368°/3168 = 0.116°$, and a tilt resolution of $188°/2160 = 0.087°$. These values are important, since the positioning commands are not given in degrees, but in number of steps that have to be performed to reach the relevant angles.

In addition to the A/D converter resolution, measuring errors due to mechanical and electrical reasons develop. Gear backlash is the mechanical error generated by the P/T head. Noise in the environment, on the other hand, generates electrical errors. It affects the wires that carry the position data from the potentiometers. This error is mostly apparent for long wiring.

It is important to know the error level of the pan-tilt unit, since it directly affects the accuracy of location results. Towards this end, the rotation error is measured. As announced by their manufacturers; Bewator P25, the P/T head utilized in the project, involves $0.2°$ gear backlash. Together with the resolution precision, this makes a total theoretical error range of:

$$\begin{aligned} \sum pan\ \ rotation\ \ error &= 0.316° \\ \sum tilt\ rotation\ \ error &= 0.287° \end{aligned} \qquad \text{Eq.112}$$

After the rotation accuracy test, a total error range of $0.8°$ is measured including the external errors in addition to the theoretically calculated ones. $0.8°$ deviation generates a 4.2 centimeters shift at 3 meters distance.

$0.8°$ is the maximum amount of deviation measured in one positioning step. Even though this looks like a negligible value, the sequential relative positioning of the pan-tilt unit increments the error, and substantial deviations may occur in the final phases of the pan-tilt unit's scan-path. Towards this end, inside the scan-path, four absolute positioning commands are used. These commands align the pan-tilt unit to an absolute value at the locations: $pan = \pm 60°,\ tilt = 0°$, and $pan = \pm 120°,\ tilt = 0°$. Actually, the "return home" command used in the middle of the scan-path is also an absolute positioning performed at $pan = 0°,\ tilt = 0°$. This makes totally five alignments that rectify the angle values. The reader may be curious at this point about the reason for the preference of relative positioning, even though a more accurate alternative, absolute-positioning, exists. The reason lies in our efforts to make the system compatible with different netcam models. As mentioned in section 4.4, some network cameras enable the employment of a limited number of P/T control commands. It is possible to comply with this structure by using relative positioning, since absolute positioning requires a distinct command for each position, therefore, requires unlimited access to the pan-tilt unit.

CURRICULUM VITAE

Personal information

Surname / First name	**İÇOĞLU, Oğuz**
Address	Abteilung für Bauphysik und Bauökologie, TU-WIEN. Karlsplatz 13 (259.3) A1040, Wien, Austria.
E-mail	oguz.icoglu@tuwien.ac.at
Date of birth	26 November 1975
Place of birth	Istanbul, Türkei

Work experience

Dates	May 2003 – November 2005
Occupation or position held	Research assistant
Main activities and responsibilities	Development of a vision-based object sensing system in a FWF (Fonds zur Förderung der wissenschaftlichen Forschung) project (project number: P15998-N07).
Name and address of employer	Univ. Prof. Ardeshir Mahdavi. Department of Building Physics and Building Ecology, Vienna University of Technology, Vienna, Austria.
Type of business or sector	Education / Research
Dates	July 2000 – April 2003
Occupation or position held	Researcher
Main activities and responsibilities	Several projects in Multimedia group, covering topics in: MPEG-7, image-processing and user-interface development.
Name and address of employer	Institute of Information Technologies, Marmara Research Center, Kocaeli, Turkey
Type of business or sector	Research and development
Dates	November 1997 – November 1999
Occupation or position held	Software developer / Application specialist
Main activities and responsibilities	Design and implementation of systems for logistics and export-sales.
Name and address of employer	Department of Information Technologies, Arçelik A.Ş., Istanbul, Turkey
Type of business or sector	Manufacturing

Education and training

Dates	May 2003 – present
Title of qualification awarded	Doctor of Philosophy Doctorate degree in Architecture
Name and type of organisation providing education and training	Department of Building Physics and Building Ecology, Faculty of Architecture and Space Planning, Vienna University of Technology.
Level in national or international classification	Level 8 in draft framework of qualifications (set by Turkish Ministery of Education).
Dates	October 1998 – June 2001
Title of qualification awarded	Master of Science Master diploma in Computer Engineering
Name and type of organisation providing education and training	Department of Computer Engineering, Faculty of Electric and Electronics, Istanbul Technical University.
Level in national or international classification	Level 7 in draft framework of qualifications (set by Turkish Ministery of Education).
Dates	October 1993 – February 1998
Title of qualification awarded	Bachelor of Science Diploma in Computer Engineering
Name and type of organisation providing education and training	Department of Computer Engineering, Faculty of Electric and Electronics, Istanbul Technical University.
Level in national or international classification	Level 6 in draft framework of qualifications (set by Turkish Ministery of Education).