Automated Camera Placement using Hybrid Particle Swarm Optimization

Mohammad Reza Shams Amiri
Sarmad Rohani

This thesis is presented as part of Degree of
European Master in Computer Science

Blekinge Institute of Technology

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.
Contact Information:

Authors:
Sarmad Rohani  
E-mail: saro11@student.bth.se,  
sarmad.rohani@gmail.com

Mohammad Reza Shams Amiri  
E-mail: mosb11@student.bth.se,  
existme@gmail.com

University advisor:  
Johan Holmgren, Ph.D.  
Senior Lecturer

School of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona  
Sweden

Internet :  
www.bth.se/com  
Phone :  
+46 455 38 50 00  
Fax :  
+46 455 38 50 57

Industry advisor:  
Göran Sandberg  
Project Manager

Academy Department  
Axis Communications AB  
Emdalavägen 14 SE-223 69 Lund  
Sweden

Email :  
Goran.Sandberg@axis.com  
Phone :  
+46 46 272 18 00  
Fax :  
+46 46 13 61 30
Abstract

Context. Automatic placement of surveillance cameras’ 3D models in an arbitrary floor plan containing obstacles is a challenging task. The problem becomes more complex when different types of region of interest (RoI) and minimum resolution are considered. An automatic camera placement decision support system (ACP-DSS) integrated into a 3D CAD environment could assist the surveillance system designers with the process of finding good camera settings considering multiple constraints.

Objectives. In this study we designed and implemented two subsystems: a camera toolset in SketchUp (CTSS) and a decision support system using an enhanced Particle Swarm Optimization (PSO) algorithm (HPSO-DSS). The objective for the proposed algorithm was to have a good computational performance in order to quickly generate a solution for the automatic camera placement (ACP) problem. The new algorithm benefited from different aspects of other heuristics such as hill-climbing and greedy algorithms as well as a number of new enhancements.

Methods. Both CTSS and ACP-DSS were designed and constructed using the information technology (IT) research framework. A state-of-the-art evolutionary optimization method, Hybrid PSO (HPSO), implemented to solve the ACP problem, was the core of our decision support system.

Results. The evaluation of CTSS confirmed an outstanding satisfactory level of the respondents. Various aspects of the HPSO algorithm were compared to two other algorithms (PSO and Genetic Algorithm), all implemented to solve our ACP problem. The HPSO algorithm outperformed the other two algorithms in terms of reaching better and more accurate solutions in less iterations.

Conclusions. The HPSO algorithm provided an efficient mechanism to solve the ACP problem in a timely manner. The integration of ACP-DSS into CTSS might aid the surveillance designers to adequately and more easily plan and validate the design of their security systems. The quality of CTSS as well as the solutions offered by ACP-DSS were confirmed by a number of field experts.

Keywords: 3D Modelling, Particle Swarm Optimization (PSO), Hybrid PSO (HPSO), Automatic Camera Placement (ACP), Region of Interest (RoI), Visibility Detection.
This thesis would have never been finished without the inspiration and support of our families, supervisors, and all the people we met along the way.

A sincere thanks to Göran Sandberg and Dan Eriksson for initializing the thesis, sharing their knowledge in worthwhile discussions and comments, and giving us the opportunity of presenting our work to its potential users.

We would like to notably thank our supervisors Göran Sandberg and Johan Holmgren for their continuous commitment and support, invaluable guidance, and their patience all the way from the start to finish of our thesis.

Finally, we are grateful to all the teachers and employees at BTH as well as our colleagues at Axis Communications AB whom we met and worked with during the thesis. We are specially thankful to the teachers of the COM department at BTH and the employees of the Academy and Global Sales departments at Axis, for their support and priceless feedback.
## List of Figures

2.1 Research process flow diagram ........................................... 6

3.1 FoV visualization ............................................................... 13
3.2 Generic camera types ......................................................... 14
3.3 Camera mounts ................................................................. 15
3.4 Polygons with (B) and without (A) holes ............................... 15
3.5 Visibility polygon .............................................................. 16

4.1 Conceptual model of camera parts ........................................ 32
4.2 Camera settings panel ......................................................... 33
4.3 Updated CTSS ................................................................. 38

5.1 Lens formula ................................................................. 42
5.2 Projected FoV cone ............................................................ 43
5.3 Dead zone coordinates ........................................................ 43
5.4 Dead zone calculation ........................................................ 44
5.5 Isosceles trapezoid polygon .................................................. 46
5.6 Event type 1 - Based on Erdem and Sclaroff (2006) .................. 50
5.7 Event type 2 - Based on Erdem and Sclaroff (2006) ................. 50
5.8 Event type 3 - Based on Erdem and Sclaroff (2006) ................. 51

6.1 Particle encoding ............................................................. 54
6.2 Emitting a semi-infinite horizontal ray from a test point (Hit-Test) 54
6.3 Star and Singly-linked Topologies ......................................... 58
6.4 Area calculation .............................................................. 59
6.5 Double Sided Door RoIs ...................................................... 62
6.6 Visualization of iterative optimization ................................... 64

7.1 Scenario 1 .......................................................................... 75
7.2 Scenario 2 .......................................................................... 76
7.3 Scenario 3 .......................................................................... 76
7.4 Deviation from best found solution - Scenario 2 ......................... 78
7.5 Scenario 1 - Final result: A- Optimizer B- SketchUp ............... 80
7.6  Scenario 1 - Final result seen by: A- Camera 1 B- Camera 2
7.7  Scenario 2 - Final result: A- Optimizer B- SketchUp
7.8  Scenario 2 - Final result seen by: A- Camera 1 B- Camera 2
7.9  Scenario 3 - Final result: A- Optimizer B- SketchUp
7.10 Scenario 3 - Final result seen by: A- Camera 1 B- Camera 2
7.11 Scenario 3 - Final result seen by: A- Camera 3 B- Camera 4
7.12 Average fitness values - Scenario 1
7.13 Average fitness values - Scenario 2
7.14 Average fitness values - Scenario 3
7.15 Scenario 1 - DBFS comparison: HPSO vs. GA and PSO
7.16 Scenario 2 - DBFS comparison: HPSO vs. GA and PSO
7.17 Scenario 3 - DBFS comparison: HPSO vs. GA and PSO
7.18 Sorted $G_{Best}$ fitness values for each run - Scenario 1
7.19 Sorted $G_{Best}$ fitness values for each run - Scenario 2
7.20 Sorted $G_{Best}$ fitness values for each run - Scenario 3
7.21 Sample particle exploration movement path
7.22 $G_{Best}$ Movement path in a sample scenario
7.23 Particles’ convergence - PSO vs. HPSO
List of Tables

2.1 Data collection methods ................................................. 11
7.1 Experimental settings for all scenarios .......................... 77
7.2 Particles Vs. Iterations - Scenario 2 ............................. 77
7.3 Base configuration - All scenarios ................................. 79
7.4 Experiments’ results .................................................. 79
7.5 Average DBFS comparison of PSO, HPSO, and GA for all scenarios ................................................. 85
7.6 Best found solutions and 95% threshold for all scenarios ........ 86
7.7 Fraction of 95% threshold fitness values in each algorithm for all scenarios ................................................. 86
7.8 Time consumption of different modules in C# implementation of HPSO ................................................. 91
# List of Algorithms

1. Main PSO algorithm 
3. Handle event points - Based on Erdem and Sclaroff (2006) 
4. Hit-Test algorithm 
5. Calculating the area of a polygon 
6. Update particle’s fitness values based on its genes 
7. Particle hill-climbing 
8. Greedy PSO iteration 
9. HPSO algorithm 
10. Elite selection 
11. Crossover operator 
12. Genetic algorithm
# Contents

**Abstract**

*Abstract*  

1  **Introduction**  
1.1  Scope and Purpose of the Study  
1.2  Contributions  
1.3  Outline

2  **Research Methodology**
2.1  Research Questions  
2.2  Research Design  
  2.2.1  Data Collection Methods

3  **Background**
3.1  Definitions  
3.2  Related Work  
  3.2.1  Coverage and Visibility  
  3.2.2  Automatic Camera Placement (ACP)  
3.3  Particle Swarm Optimization  
  3.3.1  PSO: A Part of Swarm Intelligence  
  3.3.2  PSO Basics  
  3.3.3  PSO Parameters  
  3.3.4  Improving Convergence Speed

4  **Camera Toolset Modelling in SketchUp**
4.1  Introduction  
4.2  Requirements Analysis  
  4.2.1  Already Existing Tools  
  4.2.2  The Requirements  
4.3  CTSS Implementation  
4.4  CTSS Evaluation: Axis Global Sales Engineers Conference  
  4.4.1  Online Questionnaire Analysis  
4.5  CTSS: Updated Features
6.10.4 The implemented Genetic Algorithm ........................................ 72

7 Experimental Results and Discussions  ........................................ 74
  7.1 Experiments .................................................................................... 74
  7.1.1 Experimental Settings ................................................................. 74
  7.1.2 Swarm Size vs. Iterations ............................................................. 77
  7.1.3 Configurations ............................................................................. 78
  7.1.4 General Results of the Experiments .............................................. 79
  7.2 Comparing HPSO with PSO and GA .................................................. 83
    7.2.1 Average Fitness Values .............................................................. 83
    7.2.2 Deviation from the Best Found Solution (DBFS) ....................... 84
    7.2.3 Fitness Values ........................................................................... 85
    7.2.4 Exploration and Exploitation Comparison between HPSO and PSO .......................................................... 88
    7.2.5 Particles’ Convergence ............................................................... 89
    7.2.6 Computation Time ..................................................................... 90
  7.3 Validity Threats ............................................................................... 91

8 Discussion and Concluding Remarks ................................................... 94
  8.1 Future Work ................................................................................... 95

A Appendix ............................................................................................. 102
  A.1 Figures ....................................................................................... 102
  A.2 Source Codes ............................................................................... 102
    A.2.1 Face Extractor (Ruby) ............................................................... 102
    A.2.2 Velocity Update (C#) ............................................................... 104
    A.2.3 Velocity Clamping (C#) ........................................................... 104
    A.2.4 Particle Movement (C#) .......................................................... 105
    A.2.5 Neighbourhood Selection (C#) ................................................. 106
    A.2.6 Particle::UpdateFitness(C#) .................................................... 106
  A.3 Cameras used in the experiments ..................................................... 107
  A.4 Questionnaire ................................................................................. 107
    A.4.1 Questions .............................................................................. 107
    A.4.2 Result Statistics ..................................................................... 108

ix
Chapter 1

Introduction

This Master’s thesis is the result of a cooperation between Blekinge Tekniska Högskola (BTH) and Axis Communications AB (Axis). This section introduces our thesis. It begins with a background, purpose, and discussion of the research problem. It then continues with addressing the scope of the problem. Finally, the layout of this thesis is presented.

Video surveillance is nowadays a fundamental part of the design process of new and especially large buildings. Each building usually requires some key areas, such as entrance doors or windows, to be monitored. It is essential for surveillance system designers to decide where exactly to place the video cameras and what technical specifications to consider in order to obtain optimal coverage. Satisfying end users’ expectations regarding the actual visible areas seen through camera eye is another aspect to consider when designing such systems.

In practice, planning the cameras’ arrangement requires the designers to first determine their customers’ needs. Some common requirements in all projects, like budget, apply here as well. Other needs that are more specific to the planning of surveillance systems include installing proper types of cameras with right intrinsic parameters on right places while capturing the project’s specific key areas. Traditionally, all these decisions needed time-consuming and difficult manual calculations. Integrating 3D visualization with surveillance system design support could assist the designers do more accurate and cost-effective planning.

This thesis contributes to the improvement of surveillance system planning process by designing an enhanced camera toolset in SketchUp (a 3D CAD tool) as well as an automatic camera placement optimizer using a state-of-the-art Hybrid Particle Swarm Optimization (HPSO) method. Integration of the implemented camera toolset in SketchUp and a decision support system provides a full design and development life-cycle in the planning process for surveillance systems. Such an integrated approach enables the system designers to model and review the environment while benefiting from the assisted automatic camera placement, something that could serve them greatly.
1.1 Scope and Purpose of the Study

The main purpose of this thesis work was to improve the surveillance system design process for the designers of such systems in SketchUp. We found out that in practice, there was a need for improving two main activities in the design process. The first improvement concerned the modelling part of surveillance system design. The modelling support was provided through building a camera toolset (which is referred to as CTSS, standing for Camera ToolSet in SketchUp). It contains nearly all Axis cameras’ 3D representations with interactive features. The second improvement concerned finding an efficient arrangement of cameras in a given scene considering various constraints. A camera placement optimizer was designed and implemented to address this need (which is referred to as HPSO-DSS, standing for Hybrid Particle Swarm Optimization Decision Support System). Moreover, providing an integration mechanism between these two areas could enrich the design process by equipping it with visualization and validation means. These two subsystems are described in detail in the following chapters.

1.2 Contributions

Investigation and development of a camera modelling support in SketchUp (CTSS) as well as design and construction of a novel optimization algorithm to solve the automatic camera placement (ACP) problem in an arbitrary polygonal area were our two main contributions.

HPSO-DSS resulted in an improved automatic suggestion of a heuristic solution for a determined number of cameras considering scene occlusions and a maximum coverage of regions of interest (RoIs). In HPSO-DSS, compared to other implemented optimization models, the solution quality was improved in terms of covered areas with regard to the RoIs’ specific constraints in fewer number of iterations.

1.3 Outline

In Chap. 2, the research questions along with the methods used to address these questions are discussed. Chap. 3 contains the background material on the existing approaches for the ACP optimization problem as well as other related work. Further in Chap. 4, we explain the modelling part of the thesis. Chap. 5 includes the camera and environment model as well as the algorithm used to detect the visibility polygons. In Chap. 6, the Particle Swarm Optimization technique and later the HPSO method in the application of the ACP problem domain is outlined. Chap. 7 contains the analysis of the experimental results and a discussion on the
presented optimization models. Finally, Chap. 8 concludes the thesis and suggests some future work.
Chapter 2

Research Methodology

One attempts to address the aims, objectives, intentions, and plans in a formal manner when designing research. It is important to consider all available resources and to make sure that there is a clear line between the traditional and novel ideas (Hakim, 1987). The goal is, hence, to choose the appropriate strategies and methods to answer the research questions (Robson, 2002). In this chapter, the research questions as well as the research methodology used to address these questions are presented.

2.1 Research Questions

As stated earlier, the main purpose of this thesis is to provide the surveillance system designers with an interactive and integrated design tool. A decision support system to assist the designers in modelling their desired systems and planning the camera arrangement in given floor layouts is the other purpose. We, thus, defined our main research question as follows:

How can the design process of surveillance systems be improved by introducing an interactive and automatic design support system?

The main research question is divided into three sub-questions, as follows:

- **RQ1**: How can a camera toolset in SketchUp be designed to enhance user experience in surveillance system design process?
- **RQ2**: How can Particle Swarm Optimization (PSO) be applied to automatic camera placement in a continuous problem space considering obstacles and RoIs?
- **RQ3**: How does HPSO perform compared to PSO and Genetic Algorithm (GA)?
RQ1 corresponds to the modelling part of the surveillance systems’ design. RQ2 and RQ3 correspond to the part associated with planning camera arrangements in a given floor layout. These questions are addressed by the different steps described in the following section.

2.2 Research Design

In order to answer our main research question, two separate, yet related, subsystems were developed. The development of these subsystems were addressed by the information technology (IT) research framework introduced by March and Smith (1995). According to this framework, it is of great importance for the IT researchers to take into account both utility, as a design science, and theory, as a natural science. Our work of designing and constructing the two subsystems, with regard to the research framework of March and Smith (1995), is finalized in several steps as shown in Fig. 2.1.

**Figure 2.1:** Research process flow diagram

**Requirement Analysis - CTSS:** Prior to building our first subsystem, a requirement analysis was conducted by studying the technical documentations regarding camera installation planning, interviewing the domain experts and practitioners, and reviewing the existing camera toolsets in other system design tools (i.e. Revit and Visio). The reason for reviewing other similar camera toolsets was to be able to construct a toolset that offers improved functionalities, compared to the other toolsets, in terms of user experience. The requirements are listed in Sec. 4.2.2.
Build and Evaluate CTSS: According to March and Smith (1995), when conducting an IT research, four types of artefacts exist: constructs, models, methods, and instantiations. An instantiation, meaning a working computer-based subsystem (i.e. CTSS), was built based on the gathered requirements on the previous step. Implementation of the camera toolset started by investigating SketchUp’s environment and its underlying Ruby API in order to get familiarized with what it provides. Further, the required camera mechanics and some of its interactive features were added to the design by integrating SketchUp features and its provided Ruby API. A rapid prototype was then constructed for two main camera series, namely, Fixed and Fixed Dome. After the prototype’s accuracy was confirmed by a number of field experts, the implementation of the real toolset was started. The development process continued iteratively by: a) comparing the implemented system to other existing camera toolsets and formally\(^1\) or informally\(^2\) evaluating it by field experts and potential users and b) updating the functionalities accordingly. This iterative process continued until reaching to a point where the desired user experience and usability was achieved\(^3\). The result was a complete toolset in SketchUp, with enhanced user experience, containing all camera series while satisfying their required functionalities. Here, RQ1 could be answered.

Since we were focusing on designing and implementing a camera toolset with an enhanced user experience, CTSS was evaluated based on the categorization done by Oates (2005). The author introduced the three following evaluation types:

- **Proof of concept** corresponds to the representation of a small (or even an incomplete) prototype with the purpose of demonstrating the feasibility of a researcher’s method or theory under specific circumstances.
- **Proof by demonstration** suggests an evaluation of the product in practice, though not in the real-world situation. Here, the context of the research is limited and, therefore, the method or theory is only applied to a restricted context.
- **Real-world evaluation** evaluates the product in real-world settings.

The evaluation of CTSS was done using the first and second approaches mentioned above. A prototype as well as the final working system were evaluated by the field experts using proof of concept and proof by demonstration approaches respectively. We took the opportunity of evaluating CTSS in various occasions, including informal sessions with the field experts and two different conferences held at Axis. The participants of both conferences were part of CTSS’ potential users, a fact that legitimised using their feedback for the evaluation purposes. In

\(^1\) By conducting an online questionnaire.
\(^2\) By casually asking for feedback from field experts.
\(^3\) The whole process is further described in Chap. 4.
order to evaluate CTSS in terms of user experience, an online questionnaire was designed and sent to the participants with a 53% response rate. The results were considered as great evaluation and validation means as well as a useful data resource for improving CTSS.

**Literature Review - ACP Optimization:** Another requirement of this thesis (to answer RQ2 and RQ3) was a need for a computer decision support system that could help the surveillance system designers to automatically place the cameras into a given floor plan. This resulted in the second part of our thesis: investigating the effects of integrating an optimization method into a computer decision support system on improving surveillance design process. This part of the investigation began by studying the existing approaches for automatic camera placement optimization by reviewing the current literature. It helped us with understanding the context and conditions under which the surveillance system design operators took decisions about placing the cameras in a given scene. We conducted literature review in ACP domain.

Our literature review started by searching in databases containing mostly peer-reviewed studies. The following databases were selected:

- ACM Digital Library
- Engineering Village (Inspec, Compendex)
- Google Scholar
- IEEE Xplore

At our first attempt, the following keywords were mainly searched for: \textit{automatic+camera+placement}.

The search resulted in more than 49,000 hits from just Google Scholar. Duplicated results in the Engineering Village database were excluded from the search results in order to find unique papers. Around 20,000 papers were remained at this point. We then chose the following inclusion areas:

- The paper should be peer-reviewed and in the computer science field.
- The paper should be written in English and available in full-text.
- The paper should be related solely to the arrangement of cameras for surveillance tasks rather than the placement of cameras with regard to other criteria like light source, vision, etc.
Chapter 2. Research Methodology

The first and second inclusion areas resulted in excluding most of the found papers and limited the number of articles to less than 1,000. The papers that did not match the last inclusion criterion were excluded, resulting in 212 papers. At this stage, two other sets of keyword were added to the search domain: art gallery problem and visibility detection. Moreover, the keyword optimization was added to the first set of keywords (automatic + camera + placement + optimization). The same inclusion criteria was considered in order to discard irrelevant papers. The remaining papers were divided into two parts, each part to be reviewed by one of the authors of this thesis. The articles whose title and abstract did not seem to be relevant were discarded in the next step. The excluded articles of each party were then reviewed by the other party to make sure that no important paper was excluded. The remaining and most relevant papers (37 papers) were fully read.

Identify Research Gap: After reviewing all relevant papers, we found out that among the optimization techniques used in the camera placement domain, many were based on Binary Integer Linear Programming. The optimality of these methods were assumed to be dependent on the scene’s sampling rate (grid size). Therefore, having search spaces with high sampling rates, which can result in more accurate solutions, consumed a lot of time and memory resources.

Suggest a Solution: Since our aim was to integrate the optimizer in CTSS, we could not afford deploying a time-consuming and restricted optimization technique. Hence, we decided to choose an evolutionary optimization technique. We chose Particle Swarm Optimization as our approach because of its effectiveness in solving combinatorial optimization problems and its simplicity. As a result, we added the last keyword set to our search domain: Particle Swarm Optimization. We searched for both automatic+camera+placement+PSO (or Particle Swarm Optimization) as well as for PSO alone in order to find the state-of-the-art in this optimization technique.

Build and Evaluate ACP-DSS: After reviewing the literature and getting familiar with the advantages and shortcomings of different methods that were already used in the ACP domain, we decided to apply the PSO algorithm to the ACP problem. In order to be able to design and construct the proposed method, an intuitive platform that could be easily used by the surveillance design experts was needed. This infrastructure should have as well been suitable for modelling the design environment including the scene to be optimized, barriers or occlusions, and different types of RoIs. Naturally, SketchUp, armed with CTSS, was picked as our modelling and visualization platform. The detailed implementation of the camera placement optimizer is explained in Chap. 6.
Another instantiation, meaning another working computer-based subsystem (i.e. ACP-DSS), was built at this stage. ACP-DSS was evaluated iteratively based on theoretical analysis and identified attributes of the problem characteristics. The prototype as well as the final working ACP-DSS were evaluated by the field experts using proof of concept and proof by demonstration approaches respectively. **RQ2** could be answered at this step.

**Integrate CTSS and ACP-DSS:** In this stage we integrated the final ACP-DSS into CTSS for which we added a few features into CTSS. After updating CTSS, the users could select an arbitrary floor plan in SketchUp, pass it to ACP-DSS, and visualize the optimizer’s end result in their building model.

**Theorize and Justify HPSO:** After successfully applying PSO in ACP domain, we tried to enhance the efficiency and effectiveness of the PSO optimization model by iteratively trying different operators and concepts to the standard PSO algorithm. The results were evaluated in our visualization tool and a decision was made to whether or not add the tested aspect to the new optimization model. We called the new and state-of-the-art algorithm, Hybrid PSO (HPSO). Proposal of HPSO can result in **RQ3** to be partially answered.

**Build and Evaluate HPSO-DSS:** A third computer-based subsystem (i.e. HPSO-DSS), was built afterwards. This subsystem used the HPSO algorithm in the ACP domain. HPSO-DSS was again evaluated iteratively based on theoretical analysis and identified attributes of the problem characteristics.

**Evaluation and Validation - Experiment:** After finalizing HPSO-DSS, we integrated it into CTSS (this time with no effort since we already had the necessary added features in CTSS). We then performed a series of experiments in order to evaluate and validate the whole integrated system. **RQ3** could be completely answered by the results collected from the conducted experiments. The experimental results and some interpretations of these results are presented in Chap. 7.

**2.2.1 Data Collection Methods**

According to **Locke et al. (2009)**, researchers could choose from three different data collection strategies when conducting their qualitative research: interviews, observations, and review of documents. As stated earlier, in order to answer our research questions, we used different data collection methods. The data collection resources used to address each research question is summarised in Table 2.1. In this table we considered two columns for review of documents since we had two types of documentation to review in our thesis: technical documents and research
documents published by various researchers. We also added questionnaire as a resource to collect data due to the fact that we used the data gathered from our questionnaire to evaluate and update CTSS.

**Table 2.1:** Data collection methods

<table>
<thead>
<tr>
<th></th>
<th>Interviews</th>
<th>Technical</th>
<th>Research</th>
<th>Observations</th>
<th>Questionnaire</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>RQ2</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ3</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

Background

Traditional manners of designing the placement of surveillance cameras in a given space rely, to a large extent, on human experts (security operators, camera placement designers, etc.). This could result in a reasonable optimality of camera arrangement when having a network of tens of cameras or less (van den Hengel et al., 2009). However, the practical requirements in today’s security world might ask for larger spaces (say an airport, a prison, or a shopping mall) to be monitored, leading to a need for more than tens of cameras. Aside from the size of surveillance layouts, and, hence, more required video sensors, there are various camera types with large number of adjustable parameters as well as many coverage constraints that should be met in order to satisfy security needs. Automatic surveillance camera placement using computing intelligence could, therefore, be a solution for this challenging task. Further in this chapter, the definitions of the necessary phrases that are used in the whole thesis as well as previous research in the area of the automatic camera placement and related subjects are discussed.

3.1 Definitions

Before proceeding to the next parts of this thesis, some concepts related to the application domain are first defined.

Field of View (FoV) is an intrinsic parameter that indicates the parts of the scene that is visible through the camera’s lens and is measured in degrees. FoV actually represents the area that could be monitored by the camera. The FoV for a rectilinear lens\(^1\), can be calculated based on its focal length and a chosen dimension (as shown in Fig. 3.1):

\[
\alpha = 2\arctan \frac{d}{2f}
\]

where \(\alpha\) is the FoV, \(d\) is the chosen dimension (horizontal/vertical/diagonal), and \(f\) is the camera’s focal length.

\(^1\) A lens that does not distort an image’s straight features, such as walls of the buildings.
Chapter 3. Background

Figure 3.1: FoV visualization

Depending on the height and width of the camera’s image sensor, both horizontal \((\theta_h)\) and vertical \((\theta_v)\) fields of view are mentioned in the camera’s specifications.

Visible FoV refers to the region that the camera can observe and is not occluded by obstacles. The visible field of view is limited to the effective surveillance distance and the camera’s FoV.

Overlapping FoVs: Visible FoVs shared by different cameras are called the overlapping fields of view.

Depth of Field (DoF): The distance between the nearest and the farthest objects, that appear acceptably sharp in the image provided by the camera, is called depth of field (Erdem and Sclaroff, 2006).

Sensor Size: In a camera, the actual device that captures the image data is the sensor. The sensor’s size is one of the important parameters that determines the FoV of a camera. Moreover, the sensor’s pixel density affects the resolution of the camera (Cohen et al., 2009).

Pan and Tilt Angles: Pan is the angle with which the camera is rotated in a horizontal plane and tilt is the angle with which the camera rotates in a vertical plane. The combination of pan and tilt angles represents the orientation of a camera.

Focal Length of a camera determines its angle of view and specifies the resolution of a camera. It is defined as the distance from the optical centre of the camera’s lens to its focal point. Focal length is measured in millimetres.

Resolution: In order to view a region properly, it is required for the image resolution in that region to be greater than a certain amount. In this thesis, by resolution we mean a constraint on an object’s or a region’s minimum required resolution that is a type of quality measure.
Chapter 3. Background

**Camera Types:** Many types of video surveillance cameras are available in the market. These different camera types mainly differ in their sensor element types, lens types, servo capabilities, and so forth. However, the following three camera types are commonly used in the relevant research applications (Erdem and Sclaroff, 2006): **Fixed Perspective Cameras** (also known as **Fixed Cameras**), **PTZ (Pan-Tilt-Zoom) Cameras**, and **Omnidirectional Cameras**. Once fixed cameras are mounted in place, they will have a fixed position, orientation, and focal length. PTZ cameras are mounted on a fixed position but can rotate around their horizontal (tilt) and vertical (pan) axes using remotely controlled servos. Some PTZ cameras also have an adjustable focal length (zoom). The omnidirectional cameras have horizontal FoV angle of 360 degrees but might get affected by lens aberration because of their small focal length and the convex mirrors used in their setup (Batista et al., 1998). The cameras that are not omnidirectional (i.e. fixed and PTZ cameras) can be also called **Directional Cameras**. Besides this general and common categorization in the relevant research applications, we divided the camera models, from modelling perspective, into two major types of **Dome** and **none-Dome** cameras based on their appearance and how they interact with their environment (Fig. 3.2). This new categorization was done because we found many similarities among the cameras in these two groups.

![Figure 3.2: Generic camera types](image_url)

**Camera Mounts:** In order to plan a complete security solution, the surveillance system designers require variety of accessories that could assist them in the installation process. Among these accessories, the camera mounts are studied in this thesis. With mounting accessories, the cameras could be installed on various locations and positions. Fig. 3.3 shows five camera mounts used to install a camera on a parapet\(^3\) (A), ceiling (B), corner (C), wall (D), and pole (E).

**Occlusions** or obstacles are parts of a floor plan that limit the line of sight for

\(^2\) Also unofficially called *Boxed* cameras at Axis.

\(^3\) A low protective wall along the edge of a roof, bridge, or balcony.
each camera. Such occlusions like walls, columns, etc. are considered static in this thesis and are dealt with in the visibility detection part.

The following definitions concerning the polygons and the visibility are borrowed from Ghosh (2007):

**Polygon:** “A polygon $P$ can be defined as a closed region $R$ in a plane bounded by a finite set of line segments (called *edges* of $P$) such that there exists a path between any two points of $R$ that does not intersect any edge of $P$.” In a polygon $P$, the endpoints of an edge are called *vertices* of $P$.

**Cycles:** $P$’s boundary consists of a *cycle* or *cycles* of $P$’s edges (with a shared vertex for two adjacent edges), since $P$ is a bounded and closed region. $P$ is called a *polygon with holes* in case of having two or more cycles in the boundary of $P$ (Fig. 3.4-B), and a *simple polygon* or a *polygon without holes* (Fig. 3.4-A), otherwise.
Visible Points: Two points $p$ and $q$ are said to be visible to each other, if the line drawn from $p$ to $q$ (segment $pq$) totally resides inside $P$. It can also be said that $p$ sees $q$ or, obviously, $q$ sees $p$.

Visibility Polygon: The visibility polygon $V(q)$ for a point $q$ is constructed by all points of $P$ that are visible to $q$; or $V(q) = \{p \in P | q \text{ sees } p \}$.

![Visibility polygon](image)

Figure 3.5: Visibility polygon

3.2 Related Work

A lot of research has been done in the areas of *optimal camera placement* as well as *camera coverage*. The optimal camera placement problem deals with the arrangement of multi-camera systems in which the main objective is to maximize the total coverage of the system while satisfying different scene-related constraints (Mavrinac and Chen, 2012). In order to have a proper understanding of previous conducted studies in this area, we review the related work from two separate perspectives. First, we explain the literature regarding visibility algorithms. Detecting the visibility is essential for calculating camera coverage since we need to be able to calculate what parts of the scene are visible to the camera and what parts are not. Afterwards, we will look at the studies that deal with the optimization part.

3.2.1 Coverage and Visibility

As stated earlier, one of main factors in deciding where to put different types of cameras in a scene, was the degree to which the scene is visible to each camera. In this thesis, the aim was to optimize the arrangement of the cameras in a 2D projected view of a 3D scene that resulted in a closed polygonal floor plan. The coverage problem for each camera could be, hence, reduced to the calculation of
the visible regions in that polygon. As a result, the problem could generally fall in the realm of computational geometry.

The visibility for two points in a polygon can be simply defined as: one point is visible to the other if no barrier intersects line segment between those two points. In the literature, the initial progresses in the notion of visibility were in the area of Computational Geometry by the introduction of Art Gallery Problem (AGP) (Chvátal, 1975; O’rourke, 1987) and later its variants like Watchmen Tour Problem (Carlsson et al., 1993).

In AGP, we aim to find the minimum number of guards needed to be placed inside the interior of an art gallery in a way that the entire workspace boundary is visible to them (i.e. guarded). In order to address this problem, Chvátal (1975) established a theorem (known as Art Gallery Theorem) and its associated algorithm, stating that in order to guard a gallery in form of a simple polygon with \( n \) vertices, at most \( \lceil n/3 \rceil \) number of guards were needed. The theorem led to the discussions regarding triangulation and a re-examination of AGP could bring us to the concept of convex partitioning (for more information in this regard please refer to O’rourke (1987)). In many visibility algorithms, an initial triangulation of a polygon is needed. Variety of algorithms introduced for triangulation later formed the basis for the methods developed to find the visibility polygon of a source in a polygon.

Davis and Benedikt (1979) presented the theoretical framework of solving the problem of finding the visibility polygon \( (V(q)) \) for a simple polygon in an \( O(n^2) \) time algorithm. Later, in order to compute \( V(q) \), El Gindy and Avis (1981) and Lee (1983) introduced an \( O(n) \) time algorithm. Joe and Simpson (1987) showed that both of these algorithms might fail on polygons with sufficient winding number\(^4\) (i.e. at least two) and, thus, suggested that the polygon should be pruned before using any of the two mentioned algorithms. For computing \( V(q) \) in a polygon \( P \) with \( h \) number of holes and \( n \) vertices, Asano (1985) presented an \( O(n \log h) \) time algorithm.

Joe and Simpson (1985) introduced a reorganized algorithm of Lee (1983) allowing simpler representations and correcting where his method failed. This algorithm carried out a monotone radial sweep scan of the polygon while manipulating a stack of vertices that ultimately constructed the \( V(q) \). The organization of this algorithm was decomposed into three procedures of ADVANCE, RETARD, and SCAN in perusing the current active edge. In their approach, an active edge was the edge that was currently visible to the point. ADVANCE and RETARD determined the next segment that should be processed by SCAN, while SCAN proceeded with finding the invisible edges of the polygon.

\(^4\) or the revolution number is the total number of times that the polygon’s edges turns 360° CCW around a point in a polygon.
Erdem and Sclaroff (2006) presented an algorithm for computing $V(q)$ in the context of automatic camera layout planning for polygons with holes (cavities). Their approach was very similar to what Joe and Simpson (1987) have provided in pursuing an active edge. We found the approach of Erdem and Sclaroff (2006) to be very close to what was suitable for our problem. Hence, we used their method to compute the visibility polygon of each camera. However, our way of constructing the final visibility polygon was slightly different, due to the calculation of dead zones (Sec. 5.1.2). The detailed description of our approach is described in Sec. 5.2.

### 3.2.2 Automatic Camera Placement (ACP)

There exists a close resemblance among the camera placement, sensor placement, and alarm placement problems. In the sensor placement problem, the goal is to arrange sensors in a way that a complete coverage of a given scene is acquired while guaranteeing the objects’ visibility to the sensors with a minimum resolution and adequate sampling frequency. In the alarm placement problem, we aim to place the alarms on the nodes of a graph $G$ in a way that every single fault in the system could be identified and examined (Rao, 1993). The camera placement problem, however, differs in three ways with the other two problems (1) in the restriction of the camera’s field of view due to the resolution and the sensor properties; (2) in the required projections of the FoV cone due to the panning and tilting of the cameras; and (3) in the overall required objective. In our ACP problem, the objective was not only obtaining the maximum total coverage but was more importantly satisfying certain coverage requirements of different RoIs.

### Integer Linear Programming

Horster and Lienhart (2006) formulated the problem of maximal coverage of camera sensors’ array using linear programming. The authors divided the scene into a set of grid points and then derived an integer linear programming (ILP) model using binary variables. They assumed that the scene was a simple rectangular polygon and ignored the presence of any occlusion or barrier in the scene. Moreover, as mentioned in their study, the ability to handle larger number of grid points, when dealing with bigger scenes, in order to offer solutions with higher precision should be investigated.

Erdem and Sclaroff (2006) solved the ACP problem for the planar regions, that are typical for the building floor plans, via binary optimization and over a discrete problem space. To do that, they mapped the feasible regions and the area to be covered into a grid. They thus reduced the original problem to a Set Covering Problem and then solved it using 0-1 programming. To the best of our
knowledge, they were the only ones that actually incorporated polygons with holes and visibility obstructions in ACP. As stated earlier, we used part of their work for the formulation of our problem. However, their approach still used sampling and mapping for the optimization and did not provide a solution in a continuous space in which the cameras could move anywhere in the space and not just on the grid points.

Yabuta and Kitazawa (2008) proposed a scene segmentation approach to solve ACP for the security cameras. Their segmentation approach involved dividing the observed scene into rectangular regions by extending peripheral lines of input blocks (rectangular obstacles) or merging the regions together. In this way, they simplified their visibility test procedure and continued to turn the problem into a set covering problem that then was solved using linear programming. They suggested that for efficient security monitoring of a region, there was a need to only observe selected areas i.e. “essential regions”. Hence, they used an approach called Weighted Region Observation (WRO) in which they monitored essential regions against all regions of the scene. Their approach not only oversimplifies the problem, but also was unable to cover other constraints that were necessary in handling different types of RoIs. Because of the mentioned shortcomings and the fact that this algorithm obviously performed in a discrete space, it could not be used to solve our problem.

Gonzalez-Barbosa et al. (2009) studied ACP when placing a mixture of directional and omnidirectional cameras in a given scene. They used an Occupancy Grid Map to represent landmarks and spatial representation of the environment and modelled both directional and omnidirectional types of cameras. They then, like previously mentioned studies, formulated the problem using a linear integer programming and solved it in a discrete space.

**Evolutionary Algorithms**

All the above studies used linear integer programming to solve ACP. They mainly followed the approach suggested by Horster and Lienhart (2006) to model the optimization problem. All of them believed that the solution’s optimality depended on the scene’s sampling rate. They also all agreed that it should be further investigated how to handle large number of grid points for the larger spaces or for having more precision. However, in order to satisfy the solutions’ optimality, when having different RoI constraints in the scene, we needed to be much more precise and provide a realistic and practical solution in placing the cameras. This led us to investigate the problem using other approaches and thus moving our attention from discrete space to continuous space using methods such as evolutionary algorithms.

Among evolutionary algorithms, swarm intelligence, notably Particle Swarm
Optimization (PSO), have proven to be effective in solving the optimization problems (Subrata and Zomaya, 2003) while keeping the implementation simple by having few parameters to adjust. These algorithms are good candidates to be used in solving the camera placement problem because of their effectiveness in combinatorial and multi-objective optimization problems. We specifically chose to employ PSO for solving the ACP problem, more importantly because performance plays a major role in the modelling tools. Apart from that, having realistic assumptions about the environment (i.e. occlusions and holes) was a critical issue in the design process that was easier to be addressed when using PSO.

PSO (Kennedy and Eberhart, 1995) has its roots in swarm intelligence in general, and evolutionary computations like Genetic Algorithm (GA) in particular. In this approach, different candidate solutions coexist and cooperate together simultaneously to solve an optimization problem. Each candidate is called a particle that flies in the problem search space (like the search process of a bird swarm for food) to find a good solution to the problem. A particle, during its quest, adjusts its position based on its own experience, while considering the experiences of its neighbour particles through communicating with them (Eberhart and Kennedy, 1995).

Currently, there exist few evolutionary techniques in the literature that use GA or PSO in the ACP problem area. In Xu and Lei (2009), the improvement of a camera network’s FoV using PSO was studied. In their study, the cameras were randomly spread over an area and their locations were considered fixed rather than changeable. Hence, only the cameras’ orientation, \((\theta_1, \theta_2, ..., \theta_N)\), were the subject of optimization. A practical example of their algorithm could be when the cameras were randomly scattered over an area for military purposes where the cameras’ positions were needed to be fixed.

The work of Xu and Lei (2009) was extended in Xu et al. (2010) by adding location to the optimization problem. They assumed that each camera was installed on a mobile robot giving it the possibility to move. However, in order to limit the energy consumption, the robot’s movement was limited to a pre-defined distance. Therefore, their constrained optimization problem was defined as \(\max(f(x)) = C(x)\), where \(C\) is the coverage and \(x\) is composed of the locations and orientations of \(N\) cameras: \(x = (x_1, y_1, \theta_1, x_2, y_2, \theta_2, ..., x_N, y_N, \theta_N)\). They applied three types of operators in the particle movement, namely, penalty, absorbing, and reflecting. These operators differed in how they handle the particle movement constraints when it went out of the assigned energy radius. Even though their study was close to our problem definition, they had oversimplified assumptions for a general ACP problem domain. Some examples of these assumptions included the surveillance area’s definition (mainly rectangular), the coverage model, the fact that the cameras were all of the same type, and not considering the projection
effect of the FoV cone. Moreover, their unclear and vague approach for handling the visibility in the presence of occlusions, which was very important in our case, made their method not applicable to ours.

A standard PSO algorithm is incorporated in Zhou and Long (2011) for optimizing the placement of \( N \) static cameras that try to monitor \( M \) static targets in an obstacle-free area. Moreover, each camera could only be oriented towards a discrete set of \( D \) directions. The authors addressed the problem using Binary Integer Programming, hierarchical approach (greedy search), and PSO. Again their assumptions were oversimplified with regard to our problem. They neither considered the occlusions and the surveillance area’s complexity (they only addressed simple rectangular areas), nor provided a realistic model for calculating the coverage. In addition, assuming discrete set of directions for cameras was not something that would fit into our problem.

In Morsly et al. (2012), the Binary PSO Inspired Probability (BPSO-IP) algorithm was used to solve the camera placement problem. The authors simplified the problem by letting each camera to be only placed on discrete grid points. They, therefore, turned the problem into a grid coverage problem. With this assumption, they represented each particle with \( f_x \times f_y \times f_\phi \) bits (grid and orientation sampling frequencies respectively), in which each bit represented a particular camera. A bit was, thus, 1 if there was a camera on a grid and 0, otherwise. The authors claimed that their approach could solve the ACP problem in a reasonable amount of time for even large spaces. But as it could be observed from their results and also by analysing their mapping approach, it was clear to us that their method did not provide the required optimality in a timely fashion with the desired precision for our problem definition. The reason is explained in the following example. Consider a grid of \( 8 \times 8 \) (\( f_x = 8 \) and \( f_y = 8 \)) and only eight different orientations with steps of 45 degrees (\( f_\phi = 8 \)). According to the introduced particle mapping, the search space will have \( 2^{(3+3+3)} = 2^{512} \) combinations that is a large number to explore even using a guided search. Although, we believe that a more precise solution space (much bigger than an \( 8 \times 8 \) grid mapping with only 8 possible directions) is needed. Moreover, they again solved the problem in the discrete space without considering the occlusions, barriers, or RoIs requirements.

The methods described above did not consider a continuous problem domain, complex or realistic regions of interest (RoIs), or obstacles and occlusions at the same time. Their assumptions and, hence, their methods were not exactly applicable to our case. A combination of certain aspects of the studied approaches while considering occlusions in the continuous space using PSO was something that was missing in previous research. Furthermore, by taking into account the RoIs, the problem could be modelled in a closer to reality manner. As a result, we introduced a new algorithm with a combination of different methods integrated
in a PSO algorithm and, hence, we called it a Hybrid PSO (HPSO) algorithm. In our algorithm, we took into account the realistic RoIs as well as the obstacles and occlusions to solve the camera placement optimization problem while improving the general performance of the standard PSO.

### 3.3 Particle Swarm Optimization

In this section, a brief review of the PSO algorithm and the selection of its parameters is discussed.

#### 3.3.1 PSO: A Part of Swarm Intelligence

Swarm Intelligence (SI) inspired optimization techniques have become largely popular in the last decade. Inspired from the collective behaviour of social insects (such as ants, bees, and wasps as well as animal societies such as flocks of birds or schools of fishes), SI disciplines try to design an intelligent system by mimicking these behaviours. Although single individuals of these colonies are non-sophisticated, the complex tasks could be achieved in the shadow of the cooperation and interactions between the individual members in a decentralized and self-organized manner (Blum and Li, 2008).

In both scientific and industrial worlds, the optimization problems are of high importance. An important class of the optimization problems, known as combinatorial optimization (CO), deals with finding an optimal object from a finite set of objects. A general linear CO can be defined as follows: Given a finite ground set \( E \), a subset \( S \subseteq 2^E \) (the set of feasible solutions), and a weight function \( w : E \rightarrow R \), the aim is to find \( S^* \subseteq S \) such that \( w(S^*) := \sum_{e \in S^*} w(e) \) is maximal or minimal (Bixby, 1987).

The examples of the CO problems include travelling salesman problem, vehicle routing, minimum spanning tree, and many more. The algorithms that have been developed to solve the CO problems could be categorized as either complete or approximate algorithms (Blum and Li, 2008). Complete algorithms guarantee to find the optimal solution whereas in approximate methods, the goal is to find a good enough solution in a significantly reduced amount of time. The use of approximate methods have gained attention of a lot of researchers in recent years since they are easier to implement compared to complete algorithms (Blum and Li, 2008). Particle Swarm Optimization is one of the most notable SI techniques for finding approximate solutions to the NP-hard CO problems and is the focus of this thesis that will be discussed further.
3.3.2 PSO Basics

Kennedy and Eberhart (1995) introduced Particle Swarm Optimization for the first time as a new optimization method for the continuous non-linear functions that has its roots in two main component methodologies: the artificial life in general and bird flocking, fish schooling, and swarm theory in particular. The PSO concept was originated by the simulation of a simplified social system when searching for food. Assuming each bird as a particle, the individual particles move towards a position based on their current velocity, their own previous experience, and the best previous experience of the group that was known so far.

As described in Kennedy et al. (2001), PSO follows the adaptive culture model in which “given a large space of possibilities, the population is able to find multivariate solutions, patterns that solve problems, through a stripped-down form of social interaction.” The simple socio-cognitive theory that underlies PSO is based on three principles: evaluate (the tendency to evaluate stimuli, learning can not occur if the organism cannot evaluate or distinguish the features of the environment that attracts or repels), compare (individuals compare themselves with their neighbours on certain measures and imitate and follow those who are superior to others), and imitate (taking the perspective of other individuals that actually means not to blindly imitate a behaviour but rather to realize its purpose and execute the behaviour only when it is appropriate).

PSO, as it was originally proposed by Kennedy and Eberhart (1995), was about a swarm of particles each representing a potential solution, flying iteratively through the problem space in search of the best fitted solution. Like the genetic algorithm (GA), the system was initialized with a population of random solutions (particles). However, unlike GA, each particle was also assigned with a random velocity that guided the particle through its navigation in the problem space. The optimization took place by changing the velocity (accelerating in a specified direction) of each particle towards its personal best position ($P_{Best}$) and the global best position ($G_{Best}$) of the swarm.

The implementations of PSO usually follows one of the two sociometric principles, known as Global Best ($G_{Best}$) and Local Best ($L_{Best}$) PSOs. These two types are discussed briefly in the following.

Global Best PSO

In the global best PSO, each particle is conceptually connected to every other particle and it could be said that the neighbourhood of each particle is the entire 5 “The simple socio-cognitive theory describes integrated cognitive and social properties of systems, processes, functions, and models. [...] This term is especially used when complex cognitive and social properties are mutually connected and essential for a given problem.” (Wikipedia)
swarm. The effect of using this topology is that each particle is influenced by any member of the entire swarm population that performs best (Kennedy et al., 2001).

**Local Best PSO**

In the local best PSO, a neighbourhood is created for each individual and its $k$ nearest neighbours. A simple example of this topology would be with $k = 2$ (ring topology) in which particle $i$ is affected by the best particle among particles $i - 1$, $i$, $i + 1$. In other words, each particle is influenced by the local knowledge of the environment (Kennedy et al., 2001).

**Velocity Update**

Since the individual’s disposition should be adjusted towards the success of both the individual and the community (Kennedy et al., 2001), the presence of $P_{Best}$ and $G_{Best}$ is necessary in the update velocity equation. From the sociological point of view, $P_{Best}$ resembles autobiographical memory and the velocity adjustment associated with it is called “simple nostalgia”, in which an individual seeks to return to the best place that they have already experienced in the past. Alternatively, $G_{Best}$ is similar to publicized knowledge or group norm that an individual seeks to imitate (Kennedy and Eberhart, 1995). In the literature, the group norm is also referred to as the “social knowledge.” The velocity update is calculated as follows.

$$v_{id}^{t+1} = v_{id}^{t} + c_1 r_1 (P_{Best,i}^t - x_{id}^t) + c_2 r_2 (G_{Best} - x_{id}^t),$$  \hspace{1cm} (3.1)$$

where

- $v_{id}^t$ is the velocity of particle $i$ in dimension $d$ at time step $t$,
- $x_{id}^t$ is the position of particle $i$ in dimension $d$ at time step $t$,
- $P_{Best,i}$ is the personal best position of particle $i$ found so far,
- $G_{Best}$ is the best fitted particle found so far by the swarm,
- $c_1$ is the acceleration constant that determines nostalgia component contribution,
- $c_2$ is the acceleration constant that determines social component contribution, and
- $r_1$ and $r_2$ are random numbers from uniform distribution between 0.0 and 1.0.

For local best PSO (Sec. 3.3.2), we just use $L_{Best}$ instead of $G_{Best}$ in the social component of Eq. 3.1.
Position Adjustment

In each iteration, after updating the particle’s velocity, each particle will adjust its position based on its previous location in the problem space and the newly updated velocity using the following equation.

\[ x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1}, \]  

(3.2)

where

- \( x_{id}^{t+1} \) denotes the new position of particle \( i \) in dimension \( d \) (at time \( t + 1 \)),
- \( x_{id}^t \) is the previous position of particle \( i \) in dimension \( d \) (at time \( t \)), and
- \( v_{id}^{t+1} \) denotes the newly updated velocity of particle \( i \) in dimension \( d \) (at time \( t + 1 \)).

Main Algorithm

Here the algorithm used to implement the global PSO is described (Shi et al., 2001).

**Algorithm 1** Main PSO algorithm

1: Initialize a population of particles with random velocities and positions on \( d \) dimension in the problem space.
2: \textbf{while} maxIteration reached \textbf{do}
3: \quad Evaluate the fitness function for each particle.
4: \quad Compare particle’s fitness with particle’s \( P_{Best} \). If the current value is better than \( P_{Best} \), set \( P_{Best} \) to the current particle in \( d \)-dimensional space.
5: \quad Compare fitness evaluation with the population’s overall previous best. If the current value is better than \( G_{Best} \), set \( G_{Best} \) to the current particle.
6: \quad Change the velocity and position of the particle according to Eqs. 3.1 and 3.2.
7: \textbf{end while}
8: The \( G_{Best} \) is the best found particle, in terms of fitness value.

3.3.3 PSO Parameters

Swarm Size

The number of the particles in the swarm defines its initial diversity. While having a large number of particles helps to cover more subsections of the problem space, it also increases the computational complexity and degrades PSO to a random search
since the fitness function should be computed for the particles that are spread over nearly all search space (Engelbrecht, 2007). Empirical studies have shown that a small swarm size with the initial population between 10 to 30 particles is enough to find good solutions (Brits et al., 2002; Engelbrecht and van den Bergh, 2001). Although it should be noted that the optimal swarm size is problem-dependant and it is the roughness or smoothness of the search space that actually defines the number of particles in the swarm (Engelbrecht, 2007).

Neighbourhood Size

The size of the neighbourhood defines the degree to which the particles interact with each other. PSOs with larger neighbourhood sizes converge faster (in terms of the number of iterations). However, the neighbourhoods with smaller sizes are less likely to fall into local optima (Engelbrecht, 2007). Suganthan (1999) proposed an approach in which by starting with a small neighbourhood size and iteratively increasing the perimeter of the neighbourhood, an initial high diversity with faster convergence is ensured.

Acceleration Coefficients

In Eq. 3.1, the acceleration coefficients, $c_1$ and $c_2$, control the influences of nostalgia and social components of the particle’s velocity vector.

Usually the values of $c_1$ and $c_2$ are found empirically. Lower values of these coefficients result in a smooth particle flight while high values cause more acceleration and sudden movements towards other components (Engelbrecht, 2007). Kennedy (1998) extensively studied the tuning of the original PSO’s acceleration coefficients. While the default values of $c_1 = c_2 = 2$ were proposed to make the search cover all regions around the components of velocity equation, experiments indicated that setting $c_1$ and $c_2$ to 0.5 might obtain better results. In many recent studies based on the work of Clerc (1999), the value of 1.49445 is used for both $c_1$ and $c_2$ to ensure convergence using a constriction factor (Shi et al., 2001).

3.3.4 Improving Convergence Speed

One problem associated with PSO is its high convergence speed towards a local optimum. Although high convergence rate speeds up the optimization process, it might result in a not very thorough search that makes many areas of the problem space to remain unexplored.

---

A particular type of objective function used to summarise how close a given design solution is to achieving the objectives defined for the problem in question.
Two characteristics, that can define how well an optimization algorithm searches the problem space, are referred to as exploration and exploitation. The ability to search different regions of the problem space, seeking a better optima, is called exploration and the ability to focus on a promising area to refine a candidate solution is called exploitation. To provide a good optimization algorithm, there should be a well defined trade-off between these two objectives (Engelbrecht, 2007). Different factors that can be considered in PSO that help to balance this trade-off are brought in the following.

Velocity Clamping

Each particle flies through the problem space using its velocity vector. The PSO algorithm adjusts the velocity in a way that the particle could move into every dimensions of the problem space. The velocity equation discussed before (Eq. 3.1) has a tendency to result in a phenomenon called Swarm Explosion that is when the oscillations around an optimum become wider and wider in all dimensions. One method that is usually employed to damp the velocity is to prevent it from exceeding on each dimension \( d \) for the individual \( i \) (Hu et al., 2004; Kennedy et al., 2001).

\[
\begin{align*}
\text{if } v_{id} > V_{\text{max}} & \text{ then } v_{id} = V_{\text{max}} \\
\text{else if } v_{id} < -V_{\text{max}} & \text{ then } v_{id} = -V_{\text{max}},
\end{align*}
\] (3.3)

where \( V_{\text{max}} \) is usually selected empirically based on the problem characteristics.

Inertia Weight

Shi and Eberhart (1998) introduced a version of PSO that used an inertia weight as a mechanism to enable a control on the exploration and exploitation of the swarm as well as to help eliminating the need for velocity clamping (Shi et al., 2001). The inertia weight, \( \omega \), controls how much of the flight direction’s previous memory controls the particle’s new velocity (Engelbrecht, 2007). The velocity update equation (Eq. 3.1) will be, as a result, changed into:

\[
v_{id}^{t+1} = \omega v_{id}^{t} + c_1 r_1 (P_{\text{best},i} - x_{id}^{t}) + c_2 r_2 (G_{\text{Best}} - x_{id}^{t})
\] (3.4)

Setting \( \omega < 1 \) will cause the particle’s velocity to decelerate over time. Larger values for \( \omega \) help the particle to explore the problem space more easily and improve the diversity of solutions, while smaller values for \( \omega \) help the algorithm to search local neighbourhoods more thoroughly. However, too small values disable the swarm exploration ability (Engelbrecht, 2007).
Chapter 4
Camera Toolset Modelling in SketchUp

As stated earlier, improving user experience in surveillance system design process and as a result, implementing a camera toolset in SketchUp (CTSS) was one main part of this thesis that aimed to address RQ1. CTSS was built and integrated into SketchUp through a modelling process that is described in this section. The feedback received from CTSS’ potential users in two different occasions is also presented here. In addition, corresponding updates of the toolset as a result of that feedback is brought.

4.1 Introduction

The security camera designers use CAD tools to be able to visualize the cameras’ arrangement in the virtual model of the installation setting. As mentioned earlier, SketchUp was selected as our design platform mainly due to its high accessibility (it is available for free) and intuitiveness (it has a very smooth learning curve). Additionally, SketchUp is very extensible because of its powerful embedded Ruby API. It is also available on the two most popular operating systems (Windows and OS X).

With the 3D models of Axis cameras in SketchUp, integrated in CTSS, we aimed to assist security system designers to select the appropriate surveillance cameras, interactively visualize the coverage of each camera, and optimize the layout of the system in SketchUp even before the real world construction was started or was complete. The security system designers should be able to place the 3D models of different cameras directly into the CAD building floor plans in order to visualize the real look and feel of the camera setup. The areas covered by the surveillance system could as well be shown to the designers. This visualization could help the users to:

- Realize whether or not the camera setup was feasible in a real-world scenario
- Decide if the arrangement of security cameras correspond to their business and surveillance needs
• Reduce the risk of unexpected blind spots
• Identify the potential design or operational problems

Therefore, developing and utilizing the camera models in the simulated environment (of which the security cameras will be installed in reality) could result in savings in cost and time. In addition, using exactly the same CAD tool (SketchUp) by both security system designers as well as architects and engineers, enabled a complete integration of the security system’s planning with the building design.

4.2 Requirements Analysis

To be able to collect the stakeholders’ requirements and to build CTSS, we organized several brainstorming sessions with a number of domain experts at Axis. We also reviewed the existing camera toolsets in other system design tools (i.e. Visio Coverage Shapes\(^1\) and Camera Families for Autodesk Revit\(^2\)) in order to get familiarized with their characteristics and, therefore, to be able to construct a toolset that offers improved functionalities in terms of user experience in SketchUp.

4.2.1 Already Existing Tools

As mentioned earlier, we analysed Axis’ two existing security system design tools in two CAD environments namely, “Visio Coverage Shapes” and “Camera Families for Autodesk Revit”. These two products help system designers with planning and layout of video surveillance systems and can usually replace the manual performing of some complex calculations. There is also an emphasis on user-friendliness in the design of these tools to allow system designers doing their job more conveniently. In addition to reviewing the technical documentations, we also interviewed several users of these tools and asked their opinion about the advantages and disadvantages of the tools when using them for surveillance design purposes.

Visio Coverage Shapes

Axis Visio Coverage Shapes is a library of simple 2D shapes for Microsoft Visio that helps the designers to decide on the position of each camera by simulating the camera coverage based on its FoV and the required resolution. The designers can import the layout of their installation area into Visio and then choose where to place cameras in the floor plan in order to visualize each camera’s covered area. Visio offers a 2D drawing environment with Visio Coverage Shapes providing

interactive FoV adjustments and simple drag-drop features for the cameras. Our interactive pan, tilt, and zoom features were inspired by the Visio Coverage Shapes. So did the calculations for modelling a realistic FoV considering the dead zones.

Camera Families for Revit

Architects and security system designers who work with Autodesk Revit can use Camera Families for Revit to place the security cameras in the building layout’s 3D model. This toolset also provides required meta-data for Building Information Modelling (BIM). Moreover, it helps the users visualize each camera’s covered area. This lets them verify if the scene’s critical parts are covered and/or if the unwanted spots are blocked from the camera eye. We were inspired by the Revit Camera Families’ user interface in the way the cameras’ parameters were shown to the users.

4.2.2 The Requirements

The requirements for building CTSS were extracted after collecting all information gathered in the informal face-to-face interviews with the domain experts, as well as reviewing the provided features in Visio Coverage Shapes and Camera Families for Revit. The main requirements are listed in the following.

- The ability to easily place camera models in a scene in SketchUp.
- Visualizing a realistic FoV considering relevant camera parameters.
- Manipulating all camera’s intrinsic parameters and visualizing any changes in the model.
- Demonstrating the scene in the exact way that is seen through the camera eye.
- Visualization of the interactive camera rotation.

After extracting the requirements and realizing the actual stakeholders’ needs regarding CTSS, we analysed SketchUp’s environment and its underlying Ruby API’s object model. The overall modelling process was then divided into two separate parts using the two techniques of: geometric modelling and behaviour modelling. While the geometric modelling deals with the problem of graphical data representation based on geometric shapes, the behaviour modelling studies the movements and behaviours of the objects in the scene that might include animation capabilities (Shen and Zeng, 2011). In order to build CTSS, the following tasks were to be accomplished.
- Modelling two different series of camera models: fixed and fixed dome
  Modelling separate parts of the camera (body, attach, mount (Fig. 4.1))
  Implementing mechanics of the camera (pan and tilt functionalities)
  Modelling camera’s FoV Cone

- Building the toolset (extension)
  Building the user interface to let users place cameras into the scene
  Building an object model to manage newly placed cameras in the scene
  Managing attributes of each camera through a web panel
  Implementing see through camera eye feature
  Extracting the selected floor plan (polygon) for the optimization part
  Importing the optimizer’s solution to SketchUp

### 4.3 CTSS Implementation

In the following parts, the actual process of building the camera toolset and realizing the discussed requirements will be described and presented.

**Dynamic Components in SketchUp**

In SketchUp, complex models can be treated as either components or groups. While these two can be treated the same way at some points, the components are used for creating reusable scene objects that can be placed within other models, while the groups are mainly used for combining entities in order to perform quick operations (like copy or move). The components are programmed to know what they are, using their embedded meta-data. Another main difference between the components and the groups is that the copies of components (that are called instances) relate together, so changes in one can affect the other. In addition, the components provide advanced features like glue$^3$ and cut through$^4$.

Considering the mentioned facts regarding the components, each camera was designed as a component in SketchUp, and its different intrinsic parameters were added to its definition through component attributes so that they could be modifiable.

---

$^3$ Glue is the component’s ability to get aligned to a specified surface.

$^4$ Cut through is when we want a component to cut through a surface like when having a window that cut open the wall so that the inside of the building can be viewed.
Chapter 4. Camera Toolset Modelling in SketchUp

**Camera Structure**

In order to be able to add the main mechanics (pan and tilt functionalities) of the camera to the model, each camera component was divided into three separate parts, namely, *Camera Body*, *Camera Attach*, and *Camera Mount* as shown in Fig. 4.1.

Each of these subcomponents were designed to have different rotation axes calculated based on the camera’s real model. Using dynamic component attributes, we formulated different rotation angles to each part for simulating the desired behaviour in case of pan or tilt. Moreover, the FoV cone of each camera was considered as a subcomponent of the camera body in order to inherit the camera model’s dynamics (e.g., when it is panned or tilted). To enable each camera to pan and tilt, angular variables were attached to the main component and then the corresponding rotations of the desired axis was assigned to different involved parts.

**Modelling Cameras’ FoV**

In order to visualize each camera’s FoV and DoF, two separate proportional cones with semi-transparent materials were designed. The cones became dynamic by adding the size constraints (LenX, LenY, LenZ) to their definition. The formulas to calculate these sizes based on the camera sensors where added to the main camera component and are described in Eq. 5.1c.

**Integrating Camera Models with SketchUp’s Object Model**

To be able to control camera models in the scene using Ruby API, we needed to track each inserted model and keep their references somewhere in the memory. One main requirement of our project was to provide a way to let the users modify the cameras’ parameters and then see the effect of the changes in the corresponding model. For this reason, we introduced an in-memory map between components in the scene and our Ruby camera objects. Hence, the information regarding the cameras’ parameters could be extracted and shown to the user. By using the SketchUp APIs, the information can be presented and acquired in the web panels using HTML documents. Thus, an HTML form representation was used to show the camera’s intrinsic parameters to the users. Moreover, the users can manipulate the fields and post back the form. After performing the required validation on the manipulated data, they will be reflected on the scene’s selected component and the user can visualize the changes in the model. An overview of
Chapter 4. Camera Toolset Modelling in SketchUp

the object model used for this integration is brought in Fig. A.2.

Modifying Cameras’ Parameters

Axis has more than a hundred different camera models that the users should be able to use in the scene. It is confusing to provide separate icons and modelling representations for each of these cameras. Additionally, all models in each family series look exactly the same. Therefore, we used camera family series to categorize them into 25 different group models. By placing each group model in the scene, all related cameras to that specific group model will be filtered and presented to the user. The user can then select the exact camera model in the web panel and all camera parameters (such as horizontal/vertical resolutions, minimum/maximum focal length, desired focal length, etc.) will be applied to the model (Fig. 4.2).

Camera View

In addition to modifying a camera’s parameters in CTSS, the users can also see through the camera’s eye to figure out how the scene is viewed by the camera. This functionality is called Camera View. To implement this feature, we defined a hidden straight line from the centre of the camera’s lens to the centre of the Depth of Field’s cross section. SketchUp’s built-in camera object was set to look towards this line with its FoV being the same as the selected camera’s.

Cameras’ Alignment in the Scene

The camera models should be able to align themselves to the walls or on the ceilings or any other place that they should be installed. The alignment problem for the camera models could be solved easily with help of a built-in feature in SketchUp with which any component could be glued to a sloped, horizontal, vertical, or any face on the scene. The definition of the component’s axes determine the place and orientation of the alignment. When aligning a component to a face, there is also a possibility (using SketchUp’s “cut through” feature) for cutting open the surface and put the component in a way that some part of it can be placed at the other side of the face. That feature is used in the modelling of the recessed cameras. These cameras are placed in a protection box that can be installed in the ceiling and therefore should cut through it.

Interactive Camera Motions’ Visualization

One built-in feature in SketchUp, namely, Interact Tool is employed in CTSS in
order to visualize the rotation of the camera’s FoV coverage in the scene. With this feature, the users can define a specific pan angle $P$ for the camera and then see how the camera will view the scene when it is panned $P$ degrees.

**Floor Plan Extraction (For Optimization)**

One of our main requirements in building CTSS, which also formed the basis of our work flow in the optimization part, was the need to completely integrate the optimizer into SketchUp. Using this feature, we not only can feed the floor plan polygon and the regions of interest directly (interactively) to the optimizer, but also can visualize the optimizer’s result in SketchUp for further validation.

SketchUp offers a very helpful method corresponding this need of ours. The way SketchUp treats faces is a unique way compared to other CAD tools. Here, when one draws some other geometries on top of a polygon’s face, SketchUp automatically decompose the face into several separate faces. For instance, consider having a face of a floor plan that on top of which we draw a number of columns. SketchUp decompose the face and subtract the columns’ faces from the floor plan. Hence, selecting the face of that floor plan actually results in one with a number of circular holes in it. Each hole is obviously there showing its corresponding column on the scene. The implementation of face decomposer is featured in A.2.1.

**Importing Solution (For Optimization)**

In order to integrate the solution made by the optimizer into SketchUp, we added a functionality to CTSS with which the position, orientation, focal length, and type of the cameras that were determined by the optimizer could be imported in the SketchUp’s current scene. To do so, an XML file is generated with the required information for each camera and will then be parsed by a Ruby script. The corresponding cameras’ components will be loaded into SketchUp and their specifications will be assigned accordingly. Then the cameras will be placed and aligned in the scene based on their exact supplied position and orientation information. This functionality can serve the designers as a means for evaluating the solution generated by the optimizer.

**4.4 CTSS Evaluation: Axis Global Sales Engineers Conference**

As mentioned earlier, the first version of CTSS (Fig. A.1) was presented and later evaluated by over 100 users in Global Sales Conference 2013 held at Axis. The subsystem was used by these potential users during the training sessions organized for all of the Conference’s participants. The participants were asked to complete
some tasks in SketchUp using CTSS. We designed the tasks very carefully so that they resemble a normal surveillance system design task. An online questionnaire\textsuperscript{5} was then sent to each participant. The results of the answered questionnaires (Sec. A.4) were gathered and are shown and analysed in this section.

Our objective in conducting a questionnaire survey was to have a means of evaluating CTSS. We were then able to improve the user experience when using the system. This was possible by updating this subsystem according to the feedback of the potential users and other stakeholders. The questions asked in the questionnaire corresponded to the user experience aspects of CTSS.

When analysing the user experience, we looked at the users’ entire interaction with the system in question including their thoughts, feelings, and perceptions while interacting with it (Tullis and Albert, 2010). However, the users should first be able to use the system and successfully perform the tasks that were supposed to be accomplished by employing it (i.e. usability).

### 4.4.1 Online Questionnaire Analysis

In this section, the questions asked in the questionnaire together with the analysis of their answers is brought. For statistics of all the questions, please see Sec. A.4.

58 complete responses were collected from the conference’s attendees out of totally 102 participants. In Question 2, we asked the participants their degree of familiarity with surveillance system design in the already existing tools. 81% of them were at least ”a little” familiar with one or both of these tools that made them being reasonable judges for CTSS. Most of the participants were satisfied with the amount of time it took for them to finish their assigned tasks (nearly 70% - Question 6). This answer implies two meanings: first, it could be derived that most of the users were effectively able to use CTSS and finish a task that the subsystem was supposed to carry out (i.e. usability). Second, the users could finish the task in a satisfactory amount of time (i.e. efficiency).

One of the most important questions in terms of user experience was one that asked the users about the difficulty of aiming the cameras at the targeted point or area (Question 4). Even though most attendees (around 70%) answered that it was easy to do so, we got almost a unanimous request (formal and informal) to change the way this was done. In the first version of CTSS, the users were able to change the cameras’ pan, tilt, and focal length parameters solely by entering the desired values in the attributes panel. The manual manipulation of these parameters proved to be tedious and unsatisfactory. As a result, we decided to improve this part of CTSS by enabling the users to interactively change the cameras’ pan, tilt, and zoom. This is explained further in Sec. 4.5.1.

\textsuperscript{5} http://fluidsurveys.com/surveys/sarmad-rohani/axis-toolbar-questionnaire/
Chapter 4. Camera Toolset Modelling in SketchUp

Another important question, leading to an updated feature, was one that questioned if it was difficult for the users to work with two different panels when trying to manipulate the cameras’ intuitive parameters (Question 5). More than 30% found it difficult and we again received several formal and informal request to integrate these two panels, something that is done in the updated version (see Fig. 4.3).

Questions 7 to 13 addressed usability aspects of CTSS corresponding to its learnability. For these questions we received an average value of 80%. This value was calculated by averaging the answers that corresponded to a good enough value. For instance for question 7, we added the values of “Somewhat clear”, “Clear and understandable”, and “Extremely clear and understandable” responses. For question 8, sum of the values of “Slightly flexible”, “Flexible”, and “Extremely flexible” answers was calculated. We then did the same for the other five questions. Finally, we calculated the average of all these values in order to reach to one means of measuring the learnability level of CTSS.

Questions 14 to 16 helped us in identifying the strengths and weaknesses of Visio and Revit design tools as well as our proposed toolset. Most of the feedback regarding Visio was related to its 2D nature and work flow problems such as importing and scaling floor layouts. Issues regarding Revit related to its complexity and its non-interactiveness. CTSS, on the other hand, received very positive feedback regarding its intuitiveness as well as some of its features such as camera view. Moreover, thanks to these answers we became aware of some of the problems that designers had when working with CTSS. Some of these problems concerned its user interface while some were related to some of the toolset’s missing features where were addressed in the updated version.

The responses to Question 17 assisted us in figuring out some faults of the toolset. Some issues regarding sudden crashes and wrong refreshments of the user interface were reported. In addition, some feedback regarding miscalculations in lens formula and misbehaving parts was as well collected.

Questions 18 to 20 revealed more feedback regarding what really needs to be added to, updated, or removed from CTSS. Features such as localization, instant FoV toggles, more camera models, camera mounts, and supporting corridor format were demanded. These features were included in the updated version of CTSS.

4.5 CTSS: Updated Features

Further to our discussions in the previous section, the most important features of CTSS updated version is described in this section.
4.5.1 Interactive Pan, Tilt, and Zoom

As stated earlier, when working with the first version of CTSS, around half of the users complained that it was rather hard to aim the cameras to a part of the scene by manually entering the pan, tilt, and focal length values in the provided fields. We, therefore, assigned the highest priority to address this need and started to find a compatible solution considering our 3D environment.

To solve this problem, we provided an on-screen gauge that can interact with the user through mouse movements and clicks. We extended the gauge to support interactive adjustments of pan, tilt, focal length, and the distance to object parameters for each camera. Later we got very positive feedback regarding these added features.

4.5.2 Management Panel

A problem that we observed during the training sessions at the Global Sales Engineers Conference, and that was also stated by a few other users, was regarding having a big installation site that required several video cameras on the scene. The problem occurred mainly because the cameras’ FoV cones might intersect and, thus, it was hard to manage them or even to show/hide their FoVs. Hence, we decided to add a management panel in which all the cameras that exist on the scene are listed and the user have the ability to select or manipulate them instantly using the panel. This was done by enumerating all in-memory camera objects and extracting the required information in the form of a list in a separate web panel than the attribute panel. The interaction was implemented through a combination of JavaScript and Ruby APIs.

4.5.3 Integrated Panel

Again, as mentioned in the previous section, another feedback that we got during the training sessions, and also from the answers to the questionnaire (Question 5), was that using both attribute web panel and the built-in SketchUp panel for editing camera parameters was confusing for the users (Fig. A.1). We, therefore, decided to remove the dependency of using the built-in SketchUp panel by enriching the attribute web panel with original functionalities offered by the built-in panel and add even more features to the new integrated panel (Fig. 4.3). We did this by completely redesigning the attribute web panel, through for instance adding some missing functionalities such as numeric validations and in-place unit conversion for
each field.

![Figure 4.3: Updated CTSS](image)

### 4.5.4 Mount Models

One feature requested by many of the field experts after evaluating CTSS, was to integrate the mount models and their pendant kits. “Camera Families for Autodesk Revit” already offers this feature that is implemented by embedding all the mounts in their corresponding camera series. The toolset includes five main types of mounts, namely, parapet, ceiling, corner, wall, and pole mounts (Fig. 3.3). Axis provides three types of pendant kits that could be used together with any of these mounts.

Unlike Revit models, which embeds the mounts in the camera models, we decided to provide five separate mount models and embed three different pendant kits into each mount. In this way, smart alignment feature of SketchUp for placing mounts and camera models on the scene was kept. Also, the file size of each mount model was also reduced in this way. It was as well much easier to extend the extension (CTSS) when new mounts or pendant kits were introduced and, therefore, needed to be modelled. In order to ease the installation of cameras on mounts, a functionality was added to the mounts. This new functionality let eligible cameras of each mount be automatically placed on mount models.
4.5.5 User Interface

According to some of the received feedback, changing camera models using the settings panel were confusing for many of the users. In order to provide more accessibility and better user experience in selecting different camera models, the cameras were regrouped based on their family series into separate buttons. In addition, the toolset was divided into three toolbars: main toolbar, camera toolbar, and accessories toolbar. The main toolbar provides different actions for cameras or mounts such as pan, tilt, zoom, rotation, or activating different panels. The camera models toolbar contains all the camera families including generic dome and generic boxed camera models. Finally, the accessories toolbar let the users place different mount models into the scene.

4.5.6 Additional Improvements

Based on some received feedback from the questionnaire as well as a number of personal face-to-face and email requests, the following features were also added to CTSS.

- **Corridor Format Support**: In some cases (like monitoring a staircase or a hallway) the camera housing should be rotated 90 degrees in order to obtain a portrait view of the scene. The corridor format support was, hence, added to let the designers capture their area of interest in this mode.

- **Automatic FoV Level**: What usually surveillance system designers want is having the camera’s FoV to be aligned with the ground level. A functionality was added to the camera models that made sure that their body (and subsequently their FoV cone) could be automatically rotated (levelled) to be aligned with the ground when installed on an inclined surface.

- **Localization**: Many people asked for translated version of the user interface. Therefore, a functionality for using the system’s resources and translating the whole user interface into French and Swedish languages was added to CTSS. The toolset will be offered in other languages in the near future.

4.6 CTSS Evaluation: Axis Global A&E Conference

The updated version of CTSS was presented at the Axis Global Architecture & Engineering Conference. Nearly half of the conference’s participants were the same people who were initially consulted in the validation of the previous version of
CTSS. The updated version of CTSS received very good feedback. The feedback were gathered informally this time by mainly having discussions with the participants asking them to compare the updated version with the first version of CTSS. Overall, they were all very satisfied with this new version and we did not receive any new requests of changing or updating features\textsuperscript{6}.

\footnote{CTSS is officially released by Axis and is freely accessible from \url{http://www.axis.com/techsup/system_design_tools/camera_extension/index.htm}}
In this chapter, the prerequisite concepts that need to be explained in order to make the rest of the thesis comprehensible is presented. We, thus, start by showing how the camera-related concepts were modelled and implemented. Furthermore, our visibility detection approach is described in detail.

5.1 Camera Model

5.1.1 FoV Cone

The simplest representation of a camera’s FoV could be described through an angle that is equal to its horizontal FoV ($\phi_h$) and an origin that corresponds to the camera’s centre ($a_x,y$). A camera is not theoretically bound to any upper bounds with regard to its depth of field, but obviously the objects that are too far look very small for the surveillance tasks. In order to capture the required constraints of surveillance tasks, in this thesis, an upper bound is considered for the depth of field, based on the minimum required resolution. These parameters construct a 2D representation of the camera FoV cone for us.

The calculation of camera FoV cone will be described further in this section. But since we are projecting a 3D model onto a 2D plane, it should be mentioned that each camera has also a tilt angle, which impacts the coordinates of the 2D projected cone, and therefore, bring some complications to this calculation. In Sec. 5.1.2, the effect of the tilt angle on the coverage area and its reflection on the camera FoV cone formula will be presented. Here, we describe the calculation of the projected camera FoV cone on the 2D plane (X-Y) based on the camera’s intrinsic parameters.

---

1 The reason behind using the projected FoV cones instead of their actual 3D representations is because in modelling surveillance system design we are dealing with empty floor layouts with static obstacles that can mainly be captured and represented using a 2D projection. With this assumption, performing optimization using 3D volumes of FoV is redundant, more complex, and time consuming without having any added value to the problem definition and the solution.
According to the lens formula (Eq. 5.1a), the image’s width, called FoV width ($\chi_{\text{width}}$), is calculated considering the distance, focal length ($f$), and sensor size ($X_{\text{Eff}}$).

$$FoV_{\text{width}} = \frac{\text{Distance} \times \text{Sensor Size}}{\text{Focal Lenght}}$$  (5.1a)

In reality, the cameras with variable focal lengths, in certain positions, produce an effect that is called lens distortion. Lens distortion is a deviation from the rectilinear projection so that the straight lines of the scene do not remain straight on the image. To address this problem, the camera producers provide maximum and minimum values for the focal length ($f_{\text{min}}$ and $f_{\text{max}}$) as well as maximum and minimum values for the horizontal viewing angle ($\phi_{h_{\text{min}}}$ and $\phi_{h_{\text{max}}}$). Using these provided specifications, a variable called effective sensor size ($X_{\text{Eff}}$) is calculated that can be used in the lens formula (Eq. 5.1a).

When the camera’s intrinsic parameters are given (i.e. the minimum focal length ($f_{\text{min}}$), the maximum focal length ($f_{\text{max}}$), and the desired focal length ($f$)), the calculation of the camera’s effective sensor size could be achieved using the following equations:

$$X_{\text{Eff}}_{\text{max}} = 2f_{\text{max}}\tan\left(\frac{\phi_{h_{\text{max}}}}{2}\right)$$

$$X_{\text{Eff}}_{\text{min}} = 2f_{\text{min}}\tan\left(\frac{\phi_{h_{\text{min}}}}{2}\right)$$  (5.1b)

$$X_{\text{Eff}} = (X_{\text{Eff}}_{\text{max}} - X_{\text{Eff}}_{\text{min}}) \frac{(f - f_{\text{min}})}{f_{\text{max}} - f_{\text{min}}} + X_{\text{Eff}}_{\text{min}}$$

By using the lens formula (Eq. 5.1a) and employing the calculated $X_{\text{Eff}}$, the FoV cone size for an area that satisfies a given resolution constraint ($\rho_{\text{target}}$)
Chapter 5. Problem Model

could be achieved in a straight forward manner. Assuming that $\chi$ represents the FoV cone of the required resolution and $\rho$ represents the resolution constraints, the width ($\chi_{width}$) and height ($\chi_{height}$) of the FoV cone can be calculated according to Eq. 5.1c.

\[
\begin{align*}
\chi_{width} &= \frac{\rho_{horizontal}}{\rho_{target}} \times 1000\text{mm} \\
\chi_{height} &= \frac{\rho_{vertical}}{\rho_{horizontal}} \\
\chi_{depth} &= \frac{f}{X_E} \\
\end{align*}
\]

**Figure 5.2:** Projected FoV cone

5.1.2 Dead Zones

When a real camera is placed on a scene, a fan shaped projected FoV cone (Fig. 5.2) does not provide the accurate information regarding the actual visible region of the camera. That is mainly because the camera is usually placed at a certain height ($H$) and is tilted to better capture the objects of the scene. As a result, some parts of the scene that are closer to the camera, might not be fully visible (say an object placed on the floor and very close to the camera). In the surveillance systems, what is usually important to be captured by the camera is a specific part of an object. In case of a person as the object, what is important to be captured is from the waist to the top of his/her head.

Assuming $H$ being the height on which the camera is mounted, and $N$ and $F$ respectively being the maximum and minimum heights required to be captured, certain parts of the previously modelled FoV cone (Eq. 5.1c) will not be seen by the camera. The values for $H$, $N$, and $F$ can be set by the user. Assuming that $H$, $N$, and $F$ are given, to be able to calculate the visible region, we need to define two planes that can help to identify these dead zones: plane $N$, which is closer to the camera, corresponds to the maximum height and plane $F$, which is farther from the camera, corresponds to the minimum height. The area that lies between the planes $N$ and $F$ is, hence, the visible region.

**Figure 5.3:** Dead zone coordinates
Chapter 5. Problem Model

The planes $N$ and $F$ (illustrated in Fig. 5.4) intersect with the camera’s FoV cone at 4 points (Fig 5.3). By calculating these points, the visible region could be identified based on the minimum visible areas that are identifiable by the camera. In Fig. 5.4, the calculation of two sample points of these planes are provided. As it can be seen in this figure, $\phi_h$ is the camera’s horizontal field of view and $\phi_v$ is its vertical field of view.

![Figure 5.4: Dead zone calculation](image)

To calculate each vertex of the area shown in Fig. 5.3, the following triangular functions could be used:

$$ON = \text{Min} \left( \text{abs} \left( \frac{H - H_{\text{far}}}{\sin(tilt + \phi_v)} \right), \text{abs} \left( \frac{H - H_{\text{near}}}{\sin(tilt + \phi_v)} \right) \right)$$

$\rightarrow A_{x,y} = (ON \times \cos(tilt + \phi_v), ON \times \tan(\phi_v))$

$\rightarrow B_{x,y} = (A_x, -A_y)$

$$OF = Abs \left( \frac{H - H_{\text{near}}}{\sin(tilt - \phi_v)} \right)$$

$\rightarrow C_{x,y} = (OF \times \cos(tilt - \phi_v), -OF \times \tan(\phi_v))$

$\rightarrow D_{x,y} = (C_x, -C_y)$

Finally, the coordinates should be rotated according to the camera’s pan angle ($\beta$) using the rotation matrix, $R = \begin{pmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{pmatrix}$, that will result in the
following equations:

\[
\begin{align*}
A'_{x,y} &= (A_x \cos(\beta) - A_y \sin(\beta) + o_x, A_x \sin(\beta) + A_y \cos(\beta) + o_y), \\
B'_{x,y} &= (B_x \cos(\beta) - B_y \sin(\beta) + o_x, B_x \sin(\beta) + B_y \cos(\beta) + o_y), \quad (5.3) \\
C'_{x,y} &= (C_x \cos(\beta) - C_y \sin(\beta) + o_x, C_x \sin(\beta) + C_y \cos(\beta) + o_y), \text{ and} \\
D'_{x,y} &= (D_x \cos(\beta) - D_y \sin(\beta) + o_x, D_x \sin(\beta) + D_y \cos(\beta) + o_y),
\end{align*}
\]

where \( A', B', C', \) and \( D' \) are the final coordinates of the FoV cone’s vertices. \((o_x, o_y)\) represents the camera’s location and \( \text{pan} \) is its rotation angle.

### 5.1.3 Environment Model

To model the environment, a 2D projection of the floor plan from a top-view is chosen. Using 2D projection is well suited for our problem because in this way not only the essential parameters of the coverage problem could be acquired, but also it lets us to simplify the problem from both geometrical and computational points of view, without ignoring the required details.

We used SketchUp, without losing generality, as our platform in planning an interactive camera placement. In our approach, the users are enabled to extract the selected desired face of the model (as the floor layout), with its embedded occlusions along with the given regions of interests, and pass it to the main optimization environment. The extracted layout is assumed to be a closed simple polygon that might also have cavities (holes).

The main face is treated as the outer polygon and the holes within it are identified as inner polygons. Each closed polygon (outer and inner polygons) is passed to the environment as a series of connected points with their corresponding \((x, y)\) coordinates. Additionally, different kinds of RoI are as well passed to the optimizer with regard to their specific behaviour and attributes.

In this thesis, each camera within the scene is rendered by an *isosceles trapezoid*\(^2\) polygon (Fig. 5.5). In the studies that we have reviewed, the camera rendering problem was simplified by modelling the FoV as a 2D fan shape or a simple triangle. This simplification ignores the fact that cameras are mounted at a certain height. Obviously there are determined positions (specially locations that are very near to the camera’s origin, a.k.a. dead zones) in which the coverage is partial and could not be considered visible to the camera. In the real camera placement surveillance tasks, these areas are important, accounted, and taken care of. Hence, we decided to use isosceles trapezoid polygons instead of conventional 2D fan shapes for representing the camera FoV cones. This let us capture the dead

\(^2\) A trapezoid with equal diagonal lengths and supplementary opposite angles.
zones, as described in Sec. 5.1.2, and provided a more realistic camera placement planning optimization.

\[\text{Figure 5.5: Isosceles trapezoid polygon}\]

5.1.4 Regions of Interest

Camera placement is a key element in designing a video surveillance system, although often the complexity of operational requirements makes this process difficult. This is mainly because the main intend of a surveillance design is not only to maximize the coverage, but also to capture information from various areas of the scene that are relatively important. In order to enable the designers to collect the operational requirements, employ them into such a system, and have a more clear definition of the problem, we defined the concept of *regions of interest* as follows.

RoIs are parts of the scene that are more important to the surveillance designers than the other parts. For instance, there are certain areas in the scene that must be monitored at a certain resolution to satisfy a safety constraint, or there are parts of the scene that no camera should capture (say some workstations in an organization). As mentioned earlier, achieving the maximum coverage of the whole scene is usually not the main and the only intention of a surveillance system designer. Satisfying the requirements such as identifying the persons passing a door (*identification*) or detecting the persons moving within a certain area (*detection*) can be of more priority. In order to satisfy these requirements and provide a more practical definition of the ACP problem, we formulated four categories of RoI (three of which were implemented). This categorization was done by consulting different surveillance system designers. In the following the four different types of RoI are explained.

**Essential RoIs**

The first kind of RoIs that we identified, were simple polygonal areas that should be monitored with a certain minimum resolution. These areas represent regions that are of great importance of monitoring (say the cashiers’ area at a bank, which may require more detailed coverage, or the gaming area in a casino). For these kinds of RoI, the region to be observed and the
resolution constraints should be defined by the user. The RoI and the resolution with which it should be captured by the cameras will then be passed from the modelling tool to the application as a series of points corresponding the floor layout.

**Blocked RoIs**

The blocked RoIs are used to determine areas that the cameras should not monitor at all. By putting a blocked RoI in an area, we guide the system to search through the arrangements that prevent partial or full coverage of these sections by any camera. Some examples of these areas are: employees’ workstations at an organization³, locker rooms, or other areas that violate people’s privacy and are forbidden to be recorded by the law.

**Double Sided Door RoIs**

Using these types of RoI, we express our interest in certain situations where the recognition and identification play the major role. A usage example of this RoI could be the entrance door of a building in which, for the identification purposes, we may want to recognize each person’s entrance or exit. In such cases, it is not only important for a camera to be able to cover the target area with a certain resolution, but also in order for the target to be identifiable, the area should be monitored with a specific angle. To satisfy this requirement, we defined this type of RoI with resolution and angle of view constraints for both sides of the target area (which in our case is a line).

---

³ According to the law of some countries, it is prohibited to record the activities of the employees at their workstations.
Emerald Form RoIs

In addition to the previously mentioned RoIs, there is a special case in which we want to monitor a certain point from at least three different angles. An example of such case could be a precious item at a museum in which the item should be monitored from different angles to avoid theft or other security reasons. We named this kind of RoIs as *Emerald Form* RoIs.

5.2 Visibility Detection Algorithm

The visibility has been used widely in the context of the art gallery problem. As stated before, the AGP deals with finding the number of guards needed to see every point in a closed polygon. The problem of camera placement at its core is very similar to the art gallery problem in which we want to find the minimum number of the cameras needed to cover a region. In the following, the algorithm we used to detect the visibility of a camera is brought.

5.2.1 Point visibility computation in polygons with holes

We present the algorithm of Joe and Simpson (1985) for computing the visibility polygon $V(q)$ from a point $q$ inside a polygon $P$ with $h$ holes and total number of vertices $n$ in here. Our implementation of this visibility model is based on the work of Erdem and Sclaroff (2006) with some minor modifications that will be described later.

The goal of visibility detection is to compute the polygon that contains all the visible points from a given point $q$ inside the closed polygon $P$. The main idea of this algorithm is to perform a radial sweep from point $q$ as the centre of the sweep and compute all visible segments between $[0, 2\pi]$ by keeping track of the current visible edge, which will be called $ActiveEdge$.

Assuming $P_e$ contains a list of all edges of $P$, it can be represented as $P_e = \{e_1, e_2, ..., e_i, ..., e_N\}$ where $N$ is the number of edges and $i$ is the edge index ordered CCW. Each edge $e$ consists of a start vertex $(v^s_e)$ and an end vertex $(v^e_e)$. Hence, $P_e$ can also be represented as $P_e = \{(v^s_e, v^s_1), (v^s_2, v^s_2), ..., (v^s_i, v^s_i), ..., (v^s_N, v^s_N)\}$ and since the polygon is connected, $v^s_i = v^s_{i+1}$. As the idea of the algorithm is to perform a radial sweep around $q$, first we need to convert $P_e$ into a list of polar edges, namely, $P_{ep}$. Each polar vertex could be represented as $v_i = \{\theta_i, r_i\}$. After converting each vertex and subsequently each segment to its polar form, $P_{ep}$ should be sorted based on the $\theta$ values of each segment. Obviously, the segments that do not satisfy $\theta^s_i < \theta^e_i$ or $\theta^s_i > \theta^e_i$ would not be visible from point $q$ and will be pruned. That is because we are performing
a radial sweep on a sorted list of $\theta$s (please refer to Erdem and Sclaroff (2006) for proof).

To be able to sweep the the polygon in CCW order, we need to construct the ordered list of vertex polar angles and their associated edges. Each vertex and its association to an edge can be represented as $(\theta_i, r_i, e_j)$ where $i$ is the vertex index and $j$ is the edge index. The edge list $Q = \{(\theta_1, r_1, e_1), \ldots, (\theta_i, r_i, e_j), \ldots, (\theta_M, r_M, e_N)\}$ can be constructed by sorting these triplets based on first, the polar angle $(\theta)$, second, radii $(r)$, and third, whether the vertex is the start vertex or the end vertex of the associated edge. This is a lexicographically ascending order sorting of $Q$. Alg. 2 constructs the visibility polygon by tracing the current visible edge that is stored in a variable named ActiveEdge. This algorithm iterates through all polygon edges, after we have performed pruning, and changes the ActiveEdge only at each polygon vertex point. Therefore, each vertex is an event point in which the important decisions regarding the visibility polygon takes place.

Algorithm 2 Make visibility polygon - Based on Erdem and Sclaroff (2006)

\[
\begin{align*}
P_{ep} & \leftarrow \text{MakePolarEdgeList}(P_e) \\
\text{PrunedList} & \leftarrow \text{PruneAll}(P_{ep}) \\
& \text{Sort PrunedList in lexicographically ascending order} \\
& \text{ActiveEdge} \leftarrow \text{Null} \\
& \text{while PrunedList} \neq \text{Null do} \\
& \quad \text{CurrentVertex} \leftarrow \text{PrunedList.Pop()} \\
& \quad \text{HandleEventPoint(ActiveEdge, CurrentVertex, PV, SL)} \\
& \text{end while}
\end{align*}
\]

5.2.2 Occurred events at each vertex

The heart of the visibility detection algorithm is the eventpoint handler part (Alg. 3). As we iterate through the edges of polygon, three types of events could happen at each vertex:

Type 1: Reaching to the end point of the ActiveEdge.
In this case, the current vertex should be added to the visibility polygon ($PV$), since it can be seen from point $q$ (Fig. 5.6). Subsequently, by finding the intersection between the line drawn from $q$ to the current vertex and the polygon, we will find the next visible edge. The polygon may intersect with this line in more than one point, but the one that is closer to $q$ (with smallest radii) is visible from $q$ and, hence, this point is a part of $PV$. Subsequently, the closest intersected edge will become the new ActiveEdge. This is accomplished by adding these intersections into a sorted list (sorted by radii) named $SL$. Fetching the head of $SL$ will result
in having the next ActiveEdge.

**Type 2: Reaching to the start vertex of an edge other than** ActiveEdge, while the vertex is farther away from point $q$ than ActiveEdge is. In this case, the associated edge of the current vertex will be added to the sorted list ($SL$) (see Fig. 5.7). That is because this edge can be considered as a candidate to be included in the visibility polygon ($PV$) in the future.

**Type 3: Reaching to the start vertex of an edge other than** ActiveEdge, while the vertex is closer to $q$ than ActiveEdge. In this case, the intersection point between the ray that is drawn from point $q$ to the CurrentVertex and the polygon is part of the visibility polygon ($PV$) (see Fig.
5.8). Also, the segment $\overrightarrow{k_j}$ should be added to $PV$ and ActiveEdge is changed to the associated edge of the CurrentVertex.

![Diagram](image)

**Figure 5.8:** Event type 3 - Based on Erdem and Sclaroff (2006)

### 5.2.3 Handling Polygons with Holes

To construct the visibility polygon in the polygons with holes (occlusions), the same algorithm as the one mentioned above could be applied in which we simply add the edges of each hole to the edge list $P_{ep}$. What should be noted is that in the presence of holes we sort the associated edges in CW (Clock Wise) order (in contrast to the main polygon in which the edges were sorted in CCW order). This is mainly because the interior part of the main polygon contributes to ($PV$) while in case of having holes, only the exterior regions can participate in the visibility polygon. One more thing that should be taken care of in this case, is when the ActiveEdge completely lies between the current segment (the associated edge of the current vertex) and the point of view. In this case, which occurs in type three of Alg. 3, we will add the ActiveEdge to the sorted list $SL$ for further processing.
Algorithm 3 Handle event points - Based on Erdem and Sclaroff (2006)

Input: ActiveEdge, CurrentVertex, PV, SL

if CurrentVertex.IsEndVertex and CurrentVertex.Edge = activeEdge
then
  PV.Push(CurrentVertex)
  while SL ≠ Null do
    IntersectSegment ← SL.Head
    if IntersectSegment.Radii > CurrentVertex.Radii then
      k ← Intersect(IntersectSegment.edge, CurrentVertex)
      PV.Push(k)
      ActiveEdge ← IntersectSegment.edge
      SL.PopReverse()
      break
    end if
    SL.PopReverse()
  end while
end if

if CurrentVertex.IsStartVertex then
  k ← Intersect(ActiveEdge, CurrentVertex)
end if

if k.Radii < CurrentVertex.Radii then
  SL.Push(CurrentVertex.Edge)
else
  PV.Push(k)
  PV.Push(CurrentVertex)
  pn ← CurrentVertex.Edge.End.GetRayFromOrigin()
  if ActiveEdge.Intersect(p) and ActiveEdge.Intersect(pn) then
    SL.Push(ActiveEdge)
  end if
  ActiveEdge ← CurrentVertex.edge
end if
Chapter 6

PSO for Automatic Camera Placement

In this chapter the application of the PSO algorithm to the problem of automatic camera placement is presented. With the provided approach, we were able to construct and develop the optimization toolset as well as answering RQ2. Initially, the representation of each particle and the relevant aspects according to the problem is discussed. The formulation of camera coverage with regard to different kinds of RoI is then identified. Additionally, an iterative visualization of solving a sample problem is illustrated. An improved and altered version of the PSO algorithm (HPSO) is proposed in the next section. Finally, our adaptation of the standard genetic algorithm for the automatic camera placement problem is brought.

6.1 Particle Encoding

As it was discussed in previous chapters, in particle swarm optimization, usually each particle represents a potential solution for the problem at hand. These solutions try to enhance themselves by employing their own local knowledge of their best performance and the available neighbourhood knowledge of other topologically connected particles. Since our purpose is to find the best arrangement of the cameras in a closed polygon with regard to different constraints, it is crucial to identify the representation of a solution in form of a particle.

We have used a fixed length particle encoding to represent the cameras’ arrangement. Hence, in order to arrange $n$ cameras in a given layout, each particle of the swarm should contain $n$ genes in which each gene (corresponding to a specific camera) consists of the camera’s orientation and pose as well as its model information. The particle encoding is illustrated in Fig. 6.1.

Each gene is associated with a camera that contains additional information about its specific intrinsic parameters. These parameters will be used further in the calculations of the camera cone, the visibility, and the partial coverage of each camera.
6.2 Initialization

In the initial phase of the PSO algorithm, $N_c$ number of particles are generated and randomly initialized. To prevent unnecessary computational costs, we only generate feasible (valid) solutions. In order to ensure the feasibility of the solutions, we first force the initialization process to place the cameras inside the main outer polygon and not in any of the inner polygons (holes). In this way, we ensure that no camera is placed on an invalid location. This process, in computer graphics, is called **Hit-Testing**.

For hit-testing, a simple and efficient algorithm is used based on the work of Franklin (2009). In this algorithm, a semi-infinite horizontal ray (setting $y$ coordinate fixed and increasing $x$ coordinate) is emitted from a test point. If the ray crosses any odd number of edges, then the point is located in the polygon and the Hit-Test algorithm will return True. Otherwise, the point is located outside of the polygon. (Fig. 6.2)

![Figure 6.2: Emitting a semi-infinite horizontal ray from a test point (Hit-Test)](image)

The procedure used for hit testing is listed in Alg. 4. For the polygons with holes, Alg. 4 run for all polygon holes to ensure that
Chapter 6. PSO for Automatic Camera Placement

Algorithm 4 Hit-Test algorithm

Input: PolyPoints, x, y
odd ← false
j ← PolyPoints.Count − 1
for (i ← 0; i < PolyPoints.Count; i ← i + 1) do
    if (y.IsBetween(y_i, y_j)) and (x < ((x_j − x_i) * (y − y_i)/(y_j − y_i) + x_i)) then
        odd ← !odd
    end if
    j ← i
end for
return odd

A point resides correctly inside the closed polygon and is not placed in one of the holes. In summary, each particle’s gene is initialized according to the constraints mentioned in Eq. 6.1. The camera models are sequentially assigned to the genes with regard to the user requirements. The tilt angle (β) is restricted to ±\(\frac{\phi_v}{2}\) (\(\phi_v\) is the camera’s vertical FoV), because in practice, cameras do not tilt too much around the horizontal line. Hence, again to avoid computational costs, we limit the tilt angle’s degree of change. Obviously, the pan angle (\(\phi\)) can be set to any arbitrary angle, since we want to examine all angles of the scene.

\[ M_i \leftarrow \text{User specified camera model } i \]
\[ \beta \leftarrow \text{Rand}\left(-\frac{\phi_v}{2}, \frac{\phi_v}{2}\right) \]
\[ \phi \leftarrow \text{Rand}\left(0, 2\pi\right) \]
\[ (x, y) \leftarrow \text{A random location inside polygon } P_e \text{ (using Alg. 4)} \]

Velocity Initialization

A particle’s initialization is completed by assigning it a random velocity vector. This value determines the particle’s initial tendency to move in the scene. To be able to initialize the velocity vectors, first we need to set the upper and lower bound limits of each velocity. We use the following limitations based on our empirical observations:
Chapter 6. PSO for Automatic Camera Placement

\[ \text{step} \leftarrow 20, \]

\[ V_{x_{\text{max}}} = \frac{\tau_{\text{width}}}{\text{step}}, \quad V_{y_{\text{max}}} = \frac{\tau_{\text{height}}}{\text{step}}, \]

\[ V_{x_{\text{min}}} = -\frac{\tau_{\text{width}}}{\text{step}}, \quad V_{y_{\text{min}}} = -\frac{\tau_{\text{height}}}{\text{step}}, \]

\[ V_{\beta_{\text{max}}} = 15, \quad V_{\phi_{\text{max}}} = 5, \]

\[ V_{\beta_{\text{min}}} = -15, \quad \text{and} \quad V_{\phi_{\text{min}}} = -5, \]

where \( V_x, V_y, V_\beta, \) and \( V_\phi \) respectively represent velocity vectors of \( x, y, \) \( \text{pan}, \) and \( \text{tilt} \) of the gene. \( \tau \) represents the bounding box (smallest enclosing box) around the outer polygon that can be simply calculated by finding the maximum and minimum \((x, y)\) coordinates of the polygon’s vertices.

The velocity vectors of a particle’s genes are, finally, calculated using the following equations:

\[
\begin{align*}
V_x &= \text{rand} \ast (V_{x_{\text{max}}} - V_{x_{\text{min}}}) + V_{x_{\text{min}}}, \\
V_y &= \text{rand} \ast (V_{y_{\text{max}}} - V_{y_{\text{min}}}) + V_{y_{\text{min}}}, \\
V_\beta &= \text{rand} \ast (V_{\beta_{\text{max}}} - V_{\beta_{\text{min}}}) + V_{\beta_{\text{min}}}, \\
V_\phi &= \text{rand} \ast (V_{\phi_{\text{max}}} - V_{\phi_{\text{min}}}) + V_{\phi_{\text{min}}},
\end{align*}
\]

(6.3)

6.3 Velocity Update

The main idea in updating a particle’s velocity vectors lies behind the way each particle can learn from a better particle. In our method, the learning process happens through improving the particle to be similar to the better performing particle in the particle’s neighbourhood. We do this by making similar genes (the genes with the same type of camera) progressively converging together. This means that, each gene’s velocity should be coerced towards the better performing gene. We do that by directing the velocities (in each dimension of location, pan, and tilt) in a way that they fill the gap between these two genes.

\[
\begin{align*}
V_x &= \omega V_x + c_1 r_1 (p_{\text{Best}_x} - x) + c_2 r_2 (l_{\text{Best}_x} - x), \\
V_y &= \omega V_y + c_1 r_1 (p_{\text{Best}_y} - y) + c_2 r_2 (l_{\text{Best}_y} - y), \\
V_\beta &= \omega V_\beta + c_1 r_1 (p_{\text{Best}_\beta} - \beta) + c_2 r_2 (l_{\text{Best}_\beta} - \beta), \text{ and} \\
V_\phi &= \omega V_\phi + c_1 r_1 (p_{\text{Best}_\phi} - \theta) + c_2 r_2 (l_{\text{Best}_\phi} - \theta),
\end{align*}
\]

(6.4)
where

\( p_{\text{Best}} \) is the best known state (wrt performance) of the particle so far (personal best),

\( l_{\text{Best}} \) is the best known state (wrt performance) among particle’s neighbours so far (local best),

\( \omega \) is the inertia coefficient set to 0.975,

\( c_1 \) is the nostalgia coefficient set to 2.01,

\( c_2 \) is the social coefficient set to 2.01,

\( r_1, r_2 \) are random numbers between \([0, 1]\).

The equivalent C# implementation of Eq. 6.3 could be found in Sec. A.2.2.

### Velocity Clamping

As described in Sec. 3.3.4, after calculating the new velocities and in order to prevent wide oscillation effect, it is needed to clamp velocities with defined boundary values. Using the variables specified in Eq. 6.2, we have limited the velocities as follows:

\[
V_x = \begin{cases} 
V_{x_{\text{max}}} & \text{if } V_x > V_{x_{\text{max}}} \\
V_{x_{\text{min}}} & \text{if } V_x < V_{x_{\text{min}}} 
\end{cases} \\
V_y = \begin{cases} 
V_{y_{\text{max}}} & \text{if } V_y > V_{y_{\text{max}}} \\
V_{y_{\text{min}}} & \text{if } V_y < V_{y_{\text{min}}} 
\end{cases} \\
V_{\beta} = \begin{cases} 
V_{\beta_{\text{max}}} & \text{if } V_{\beta} > V_{\beta_{\text{max}}} \\
V_{\beta_{\text{min}}} & \text{if } V_{\beta} < V_{\beta_{\text{min}}} 
\end{cases} \\
V_{\phi} = \begin{cases} 
V_{\phi_{\text{max}}} & \text{if } V_{\phi} > V_{\phi_{\text{max}}} \\
V_{\phi_{\text{min}}} & \text{if } V_{\phi} < V_{\phi_{\text{min}}} 
\end{cases}
\]

### 6.4 Particle Movement

After the velocities were updated, the particles should actually move (fly) to find a better solution. A particle’s flight is performed by moving each gene with regard to its associated velocity that impacts the next position, the pan and the tilt angles of that gene. One aspect to consider, is that this movement of the particle should not lead to an out of bound movement (i.e. not cause infeasible solutions.) In case it happened, we need to correct the move. For that, we update the velocity vector in two major ways: first, dividing its value by two (it could be in \(x\) or \(y\) direction or both); second, if the mentioned corrective method was not successful, we reflect the velocity vector (again in \(x\) or \(y\) direction or both) to create a valid move. In both cases, this change of velocity will not be permanent and will have a one-time-only impact on the movement so that it could enable the gene to escape from an invalid move. For the tilt and the pan angles, we do not apply such restrictions. Implementation of the mentioned method is listed in Sec. A.2.4.
6.5 Neighbourhood Selection

Each particle in the PSO topology follows a leader (a superior particle). The leader can either be global to all other particles (also called a Global Best PSO (Sec. 3.3.2)), or local to a neighbourhood of particles in the flock (also called a Local Best PSO (Sec. 3.3.2)). One reason to choose a Local Best over a Global Best PSO is that the former is a natural way of overcoming premature convergence (Mendes et al., 2004) since there are more than one leader and hence more than one good spots in the flock. The particles in a neighbourhood interact with each other in order to find a better solution to move to. Which particles communicating with what other particles is decided through a neighbourhood topology.

In case of a highly connected neighbourhood topology (for example a Star Neighbourhood topology as shown in Fig. 6.3a), we risk converging to a local optima since the information about the best particle can be quickly spread throughout the swarm. It is also empirically shown that using star topology can help reaching to solutions faster, yet meeting the optimal solutions less frequently compared to when using the other neighbourhood topologies (Kennedy and Mendes, 2002). Kennedy and Mendes recommended employing more suitable topologies, such as the one that was called Singly-Linked Ring by Aguirre et al. (2007), for more complicated problem settings. Here, we use this structure as our neighbourhood topology (Fig. 6.3b) for our PSO and HPSO algorithms.

*Figure 6.3: Star and Singly-linked Topologies*

Each particle $i$ in our singly-linked ring topology is in a neighbourhood with particle numbers $i-2$, $i-1$, $i+1$, and $i+2$. All particles are obviously considered to be located in a ring meaning that for a swarm with 25 particles, particle number 0 has particle numbers 23, 24, 1, and 2 as its neighbours. The C# implementation of a particle’s neighbourhood selection is shown in Sec. A.2.5.
6.6 Camera Coverage

The coverage for each camera is computed by calculating the geometrical intersection of the camera’s FoV cone and the visibility polygon for a point in a given scene. The resulting polygon will give us the information regarding which points are visible to a given camera, with a certain pose and orientation, in the desired polygonal floor plan. We, therefore, need to calculate the area of each camera’s visible field of view. The computed area of the combined intersected polygons will be used later for the optimization purpose. In order to calculate the area of a polygon, we can simply add up the areas that lie between each line segment and the x-axis. This area is in the shape of a trapezoid as shown in Fig. 6.4.

![Figure 6.4: Area calculation](image)

As illustrated in Fig. 6.4, for a given line segment A-B (from point A to point B), the trapezoid’s area between the line A-B and the x-axis could be defined as:

\[
\text{Area of A-B section} = A_y(B_x - A_x) + \frac{(B_x - A_x)(B_y - A_y)}{2}
\]

After simplification of the above formula, and extending the calculations for B-C we have the following formulas:

\[
\begin{align*}
\text{Area of A-B section} &= \frac{B_xA_y - A_xA_y + B_xB_y - A_xB_y}{2} \\
\text{Area of B-C section} &= \frac{C_xB_y - B_xB_y + C_xC_y - B_xC_y}{2}
\end{align*}
\]
Area of A-C section = \[ \frac{B_xA_y - A_x A_y + B_xB_y - A_x B_y + C_xB_y - B_xB_y + C_xC_y - B_xC_y}{2} \]

Area of A-C section simplified = \[ \frac{B_xA_y - A_x A_y - A_x B_y + C_xB_y + C_xC_y - B_xC_y}{2} \]

It is obvious that \( B_xB_y \) is positive in A-B section and negative in B-C section, so they cancel out each other. This logic continuously extends by adding the trapezoids’ areas of all line segments. Considering that the area is negative if the line segment is in CCW order (like segment C-D in Fig. 6.4), we can see that for each adjacent points, \( P_i - P_{i+1} \), we only need to calculate \( \frac{1}{2}(P_{ix}P_{(i+1)y} - P_{(i+1)x}P_{iy}) \).

Alg. 5 shows the procedure used for area calculation.

Algorithm 5 Calculating the area of a polygon

**Input:** PointList

\( last \leftarrow first \leftarrow PointList[0] \)

area \( \leftarrow 0 \)

i \( \leftarrow 1 \)

while i < PointList.Count do

\( current \leftarrow PointList[i] \)

area \( \leftarrow area + current.X * last.Y - last.X * current.Y \)

last \( \leftarrow current \), i \( \leftarrow i + 1 \)

end while

area \( \leftarrow area + first.X * last.Y - last.X * first.Y \)

**return** area/2

### 6.7 Fitness Function

The fitness function for each particle is calculated based on the area covered in each RoI and the total coverage of the floor plan. Different RoI types contribute differently to the fitness of a particle. Hence, we need to calculate the coverage for the permutation of each camera and each RoI. For instance, if we have three cameras and two RoIs, we need to perform \( 3 \times 2 = 6 \) permutations of coverage calculations to evaluate the overall fitness value of the particle as shown in Alg. 6. Since the problem definition requires maximizing the coverage of certain RoIs and minimizing some others, the total coverage is calculated by summing up all calculated fitness values.

One important aspect that should be considered when calculating the coverage for different RoIs, is that we need to combine all the covered regions together to discard the effect of overlapping FoVs. This is done through combining all the coverage geometries that contribute to each RoI and at the end applying the
RoI’s specific constraints. After calculating the RoI’s specific fitness value, the total fitness is calculated by multiplying the average of these values by the total area of the scene and adding the total coverage to it. In this way, we ensure that maximizing the RoI coverage is our first priority and maximizing the total coverage is our second priority.

The three implemented RoI types, introduced in Sec. 5.1.4, have different effects on the calculated fitness values that will be described further in the following.

### 6.7.1 Essential RoIs

For Essential RoIs, the most important requirement that should be satisfied is the resolution threshold constraint that should be fulfilled by at least one camera. In order to do so, we should recalculate each camera’s FoV cone to find out the visible area with the required resolution threshold, then the ordinary FoV cone calculation that was described earlier will be used to find the geometrical visibility. Finally, the geometry’s area will be calculated and added to the fitness value.

### 6.7.2 Blocked RoIs

Blocked RoIs should not be visible to any cameras. Hence, the calculation of the fitness value for them is the same as the Essential RoIs, except that the final calculated area should be subtracted from the fitness value. This will act as a penalty value to the fitness function and, hence, coercing the particles to avoid camera arrangements that cover these blocked regions.

### 6.7.3 Double Sided Door RoIs

For Double Sided Door RoIs, what is important is the angle with which the camera views the region. Only cameras with a specific angle in relation to the RoI can be included in the calculation of the RoI’s fitness function. The angle between these cameras ($\vec{R}$ in Fig. 6.5) and the normal vector of the RoI ($\vec{N}$ in Fig. 6.5) should be less than the specified angle of view constraint.

The angle between $\vec{R}$ and $\vec{N}$ can be calculated using their dot product as follows.

$$\theta = \arccos \left( \frac{\vec{R} \cdot \vec{N}}{|\vec{R}| |\vec{N}|} \right).$$
where

\( \theta \) is the angle with which the camera views the RoI,
\( \vec{R} \) is the ray emanated from camera to the center of the RoI, and
\( \vec{N} \) is the RoI’s normal vector located in the center of the RoI.

Alg. 6 shows the procedure for a particle’s fitness update. In this algorithm, we have different fitness value calculations for each RoI: one for the gene and one for the particle. The C# implementation of this algorithm could be found in Sec. A.2.6.

\begin{algorithm}
\caption{Update particle’s fitness values based on its genes}
\begin{algorithmic}
\STATE totalCoverage \leftarrow Calculate particle’s visibility with minimum resolution
\STATE total \leftarrow 0
\FOR {each roi in model.RoIs}
\STATE roiFitness \leftarrow 0
\FOR {each gene in particle.genes}
\STATE Update gene’s camera coverage geometry wrt roi’s minimum resolution
\ENDFOR
\STATE genesCoverage \leftarrow Combine all genes’ coverages
\STATE roiFitness \leftarrow roi.CalculateFitnessForParticle (genesCoverage)
\STATE total \leftarrow total+roiFitness
\ENDFOR
\STATE FitnessValue \leftarrow (total / roiCount) \ast model.area + totalCoverage
\end{algorithmic}
\end{algorithm}
6.8 Applying PSO in ACP: A Sample Visualization

Iteration by iteration, using the fitness function described in Sec. 6.7, the particles are attracted to or repelled from certain areas of the scene. In each iteration, these particles tend to converge to their personal best and their cluster’s local best as it was discussed in Alg. 1.

The swarm’s global best particles during different iterations are illustrated in Fig. 6.6. The floor plan consists of a closed polygon that should be covered by two cameras, in addition to a Blocked RoI that should not be captured by any camera. As it can be seen at iteration number 300, a feasible solution is achieved in which the Blocked RoI is avoided by both cameras and the scene is covered maximally.
Figure 6.6: Visualization of iterative optimization
6.9 Hybrid PSO

Some additional knowledge regarding the current problem could be incorporated to the algorithm in order to enhance the search capabilities. Hybridization of an evolutionary algorithm with standard search methods like greedy algorithm or hill-climbing is one way to approach this enhancement (Michalewicz and Fogel, 2000). In fact, there are numerous possible ways to form this hybridization. However, what is important in the combination of these approaches is to find a synergy between them while keeping the overall approach as simple as possible (Michalewicz and Fogel, 2000).

To improve the results of the PSO algorithm, we choose to combine two apparently conflicting objectives of exploration and exploitation search. Exploration aims to find new good regions in the search space while exploitation tries to find a possible improvement to the best available solution so far. Even though PSO enables us to explore the search space (through neighbourhoods) in a reasonably “good” manner, its performance in enhancing its global best is often not very efficient. Since the global best is its own $p_{Best}$ and $l_{Best}$, according to Eq. 6.4, its velocity will be equal to $\omega V$ in each dimension. Hence, the improvement of the global best is limited to its velocity direction and, therefore, it might take several iterations to reach a better fitness value. In order to handle this shortcoming, we decided to employ a local optimizer (hill climbing heuristic) in each iteration solely to the global best in order to systematically search its surroundings (it will only be applied if the global best was changed.) In this way, the process of improving the global best, and with that PSO’s overall performance, might be accelerated.

We also found that when we had many cameras on the scene, the tendency of the PSO algorithm to find good enough solution was rather slow due to the variation of invalid or poor candidates. That was, obviously, because of the increasing number of components. Hence, we employed a greedy strategy on top of the PSO algorithm. With this combination, we aim for sub-optimality of the solutions by applying the PSO algorithm on one camera at a time for a certain number of iterations. While the PSO algorithm is being applied on one camera, the other cameras are considered to have fixed attributes (position, pan angle, and tilt angle). After the algorithm proceeds for a certain number of iterations, the camera that was moving on the scene will become fixed and PSO will be applied on one of the other fixed ones and the algorithm goes on repeatedly.

In order to enable PSO to navigate more through the search space, borrowed from genetic algorithms, we introduced a mutation operator to each gene. This will let PSO genes to be mutated with a certain probability (mutation rate). In such a way, we not only employed PSO’s functionalities (guiding particles toward their local best and personal best), but also we let some of the genes to sometimes fly more freely to explore other regions that might help, to some degree, with
escaping from the local optima.

6.9.1 Hill-Climbing Heuristic

Hill-climbing is a local search optimization technique that starts with a complete configuration and then attempts to incrementally improve using local transformations. While this method is suitable for finding a local optimum, it does not guarantee to find the global optimum. Hill-climbing technique is applied to a single point in the search space and iteratively new points from the neighbourhood will be selected and examined against the evaluation function. In case one of the new points has a better fitness value than the current point, it will replace the current point; otherwise the current point stays the same. Nevertheless, this part of the algorithm will be run on the other points of the neighbourhood (Michalewicz and Fogel, 2000).

Since this approach can just guarantee finding local optima, by combining the hill-climbing method with the PSO algorithm, we can have a set of good starting points to initiate the new hybrid algorithm. To be able to improve the solutions found by PSO using the hill-climbing technique, in each iteration we need to examine all directions that every camera of the particle can move for a minimum length of step. Then again the values of the other parameters of the camera (the pan and tilt angles) will be changed by a minimum step length to be able to evaluate the new configuration. Afterwards, the best neighbour configuration will be selected and will replace the current configuration of the particle.

We limited the directions of the search to four cardinal and four ordinal directions (W, NW, N, NE, E, SE, S, SW). The pan and tilt angles were examined against $\pm \alpha$ and $\pm \theta$ only. This implied that each hill-climbing iteration required 72 ($8 \times 3 \times 3$) times of calculating the evaluation function for each camera. Obviously, this was costly for the entire optimization process and should have been reduced. Therefore, as described earlier, we only applied this heuristic search on the global best particle of the swarm in each iteration.

The experiments that we have done have showed that this combination could significantly improve the quality of the solutions. The results and the description of the experiments will be further described in the next chapter. The algorithm for the particle hill-climbing is listed in Alg. 7. Each particle’s gene carries a $hillClimbStop$ value that is initially set to one. As the algorithm proceeds, if no better configuration has been found in the neighbourhood, $hillClimbStop$ will be increased and will cause a decrease in all speed values. This will help the algorithm to decrease its search radius in case it could not improve the gene.
Algorithm 7  Particle hill-climbing

for each gene in particle.genes do
    \[ v_x \leftarrow \frac{\text{speed}}{\text{gene.hillClimbStop}}; \quad v_y \leftarrow \frac{\text{speed}}{\text{gene.hillClimbStop}} \]
    \[ \alpha \leftarrow \frac{2 \times \text{gene.hillClimbStop}}{\text{gene.hillClimbStop}}; \quad \phi \leftarrow \frac{2 \times \text{gene.hillClimbStop}}{\text{gene.hillClimbStop}} \]
    for each dir in \{ W, NW, N, NE, E, SE, S, SW \} do
        for each pan in \{ +\alpha, 0, -\alpha \} do
            for each tilt in \{ +\phi, 0, -\phi \} do
                tmp \leftarrow \text{gene.CalculateNewConfiguration(dir, pan, tilt)}
                fitness \leftarrow \text{EvaluateFitness(tmp)}
                if fitness > gene.fitness then
                    gene.hillClimbStop \leftarrow 1
                    gene.ReplaceConfiguration(tmp)
                    continue to next gene
                end if
            end for
        end for
    end for
end for

gene.hillClimbStop++

6.9.2 Greedy Strategy

Greedy algorithms are based on heuristic strategies that provide the best possible move (locally optimum choice) at each step. The name greedy comes from the fact that this approach is based on the best profit (Michalewicz and Fogel, 2000). Greedy algorithms do not always provide an optimal solution but since they make choices that looks best at the moment, hoping that this choice will lead to a globally optimal solution, they can be integrated into many algorithms to gain speed and possibly the solutions with a better quality (Leiserson et al., 2001).

After several experiments, we observed that when the number of cameras in a scene were increased, the convergence of the PSO algorithm to find a good solution slowed down. This was mainly because the number of adjustable parameters that should satisfy the fitness function was increased. Therefore, the newly generated positions and orientations for some of the cameras might overly affect the fitness function.

The greedy algorithm seeks for optimal substructures\(^1\). In the camera placement problem we cannot generally guarantee that having optimal substructures leads to an optimal solution. However, attempting to find the optimal substructure...

---

\(^1\) “A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems.” (Leiserson et al., 2001)
tures for certain intervals could help the PSO algorithm to explore and refine the solutions more efficiently. To achieve this, we introduced the concept of a WorkerGene to each particle. A particle’s WorkerGene, keeps track of the part of the particle that is actively participating in the PSO algorithm. Assuming that we have \( n \) cameras in the scene, the WorkerGene can be assigned to a value between \([0, n)\). After each \( GP_{\text{max}} \) iterations, the WorkerGene will be increased and once it reaches \( n \), it will again be set back to 0. Using this strategy, we give each camera a time window (in terms of the number of iterations) to find its optimal placement with respect to other cameras in the scene (that are considered to have fixed parameters), while the overall solution benefits from the PSO dynamics.

This heuristic is indeed greedy since in \( GP_{\text{max}} \) steps it tries to gain the maximum profit by improving one camera at a time and ignoring the other cameras. While holding the new configuration for that camera, the greedy method will try to do the same thing for the next camera. Alg. 8 describes how this procedure works in a sample iteration. The main difference between our greedy PSO approach and the original PSO method is that in each iteration of our approach, the PSO algorithm is only applied on the camera that the WorkerGene points at.

**Algorithm 8 Greedy PSO iteration**

```plaintext
for each p in particles do
    gene.Velocity.UpdatePSOVelocities ()
    gene.PerformVelocityBasedFlight ()
    p.EvaluateFitness ()
    if iterationNo % \( GP_{\text{max}} \) = 0 then
    end if
end for
```

### 6.9.3 Mutation Operator

In genetic algorithms, the concept of mutation is defined by Engelbrecht (2007) as “the process of randomly changing the values of genes in a chromosome. The main objective of mutation is to introduce new genetic material into the population, thereby increasing genetic diversity.” Although the PSO algorithm does not introduce mutation within its dynamics, we decided to embed a degree of mutation to let a portion of population escape from the dynamics of PSO and move more freely in order to explore other regions. Albeit, this should be applied with considerations (Engelbrecht, 2007) since we do not want to distort the good particles.
Moreover, we want mutated particles to preserve their relations to their original selves to some degree.

Instead of mutating certain bits in the parameters, we decided to add constrained randomness to them. Additionally, we apply mutation to only one gene of the particle at a time to avoid too much randomness as well as production of too many distorted particles. Hence, a mutated gene of a particle is relocated randomly within a mutation radius and its pan and tilt parameters are rotated by a random value between \((-v_{\text{max}}, +v_{\text{max}})\). In order to promote exploration at the beginning, the mutation rate is initially set to a larger value (0.35) and over time it will be decreased to allow more exploitation. Moreover, we only replace the parent in case the fitness value of the newly generated offspring is better than that of its related parent.

**6.9.4 Forgetful Particles**

We observed that, in some cases, the arrangement of a particle’s personal best and local best is in such a way that makes the particle fluctuating between these two without any improvements in its personal best. To resolve this issue, we introduced the concept of forgetful particles. A forgetful particle checks whether or not its personal best was improved in a certain number of iterations. In case it was not, the particle will forget the personal best and assumes that its current state is the new personal best. Obviously, any particle in the swarm can be a forgetful particle.

**6.9.5 HPSO Algorithm**

The overall description of the proposed hybrid PSO (referred to as HPSO) method is presented in Alg. 9. The algorithm starts by creating random particles and initializing their corresponding velocity vectors. Each particle consists of a set of cameras that are located randomly inside the polygonal region (floor plan) with random pan and tilt values. The cameras will be tested to be inside the polygon and not on the possible holes, using the hit-test procedure (Alg. 4). During each iteration, for each particle a randomly generated number will be compared to the mutation rate. If the generated number was less than the mutation rate, one of the cameras in the particle will be mutated and then the whole particle’s fitness will be evaluated; if it was improved we will keep the mutated particle, otherwise we will revert the particle to its previous state. The algorithm continues with finding each particle’s personal best and the swarm’s global best. Then, as described in Sec. 6.9.2, using the WorkerGene variable, we select one of the cameras in the particle and adjust it based on the particle’s individual experience \((P_{\text{Best}})\) and the best solution found so far \((G_{\text{Best}})\) according to Eqs. 3.1 and 3.2. This will force the algorithm to improve the solution according to one of the cameras only and will
actually make it searching for the suboptimalities. To enhance the convergence speed and the optimality, and before the algorithm proceeds to the next iteration, we improve the quality of the local best particles using the hill-climbing method as described in Sec. 6.9.1. Finally, when the \( \text{maxIteration} \) is reached, the current \( g\text{Best} \) is the output of the algorithm and is considered to be the good enough solution for the camera placement problem.

**Algorithm 9** HPSO algorithm

<table>
<thead>
<tr>
<th>Initialize a population of particles with random velocities and positions on ( d ) dimensions in the problem space.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>while</strong> ( \text{maxIteration} ) reached <strong>do</strong></td>
</tr>
<tr>
<td>For each particle generate a random number between 0 and 1, if the number is less than the mutation rate, mutate a random dimension of the particle.</td>
</tr>
<tr>
<td>Keep the particle if it was improved, otherwise ignore the change.</td>
</tr>
<tr>
<td>Evaluate the fitness function for each particle.</td>
</tr>
<tr>
<td>Compare particle’s fitness with particle’s ( P\text{Best} ). If current value is better than ( P\text{Best} ), set ( P\text{Best} ) to the current particle in ( d )-dimensional space.</td>
</tr>
<tr>
<td>Compare fitness evaluation with the population’s overall previous best. If current value is better than ( G\text{Best} ), set ( G\text{Best} ) to the current particle.</td>
</tr>
<tr>
<td>Change the velocity and position of the particle according to Eqs. 3.1 and 3.2 but only apply the change to the dimensions that ( \text{WorkerGene} ) is pointing at (as described in Sec. 6.9.2).</td>
</tr>
<tr>
<td>Improve the quality of the global best particle using embedded hill-climbing heuristic according to Eq. 7</td>
</tr>
<tr>
<td>Update mutation rate in proportion with current iteration number.</td>
</tr>
<tr>
<td><strong>end while</strong></td>
</tr>
</tbody>
</table>

The \( G\text{Best} \) particle is the sufficiently good fitness achieved through the optimization process.

### 6.10 Genetic Algorithm

We modelled the ACP problem using an adapted version of the genetic algorithm in order to be able to compare our proposed HPSO algorithm with another evolutionary algorithm. The implemented genetic algorithm is briefly described in this section, for further details please refer to Engelbrecht (2007).

Genetic algorithm is a stochastic search method based on natural selection and evolution (Goldberg, 1989). The standard genetic algorithms apply three basic operators (selection, crossover, and mutation) on the population in each iteration. The selection operator models the survival of the fittest behaviour of the population
where the most highly rated individuals are transferred or regenerated in the new generation. The crossover operator models the reproduction of offsprings from two parents through recombination. Finally, the mutation operator randomly alters some genes in an individual based on the mutation probability (Engelbrecht, 2007).

To employ the genetic algorithm for the ACP problem, we defined each chromosome (individual) of the population in the same way as we modelled them in the PSO algorithm (Sec. 6.1). Hence, each chromosome is partitioned into certain number of cameras where each camera has a variety of parameters (Camera Model, X-Coordinate, Y-Coordinate, Pan Angle, and Tilt Angle). Moreover, the same fitness evaluation used for PSO and HPSO is employed in GA.

6.10.1 Selection Operator

A simple elite selection\(^2\) is used as our selection operator. The selection operator sorts the whole population pool based on each chromosome’s fitness value. Half of the population with higher fitness values will then be kept and the other half will be replaced by new randomly generated chromosomes. Finally, the selection operator shuffles the whole population to avoid having a half sorted population. This operator is described in Alg. 10.

**Algorithm 10** Elite selection

**Input:** population
  \(n\leftarrow\text{population.size}\)
  population.SortByFitness()
  population.RemoveRange(n/2,n)
  for i from 0 to n/2 do
    population.add(GenerateNewSolution())
  end for
  for i from 0 to n/2 do
    c1 \leftarrow \text{rand.Next}(n)
    c2 \leftarrow \text{rand.Next}(n)
    population.swap(c1,c2)
  end for

6.10.2 Crossover Operator

We defined a one-point crossover as our crossover operator. The one-point crossover randomly selects a crossover point and the bitstrings after that point are swapped

\(^2\) In this selection scheme, the offspring needs to compete with its parents in order to obtain the possibility of participating in the next generation (Thierens, 1997).
between the two parents (Engelbrecht, 2007). To implement this, all of the camera’s parameters are converted to integer bitstrings and then the one-point crossover is applied on both parents. The two newly generated offsprings will then be added to the population. This procedure is listed in Alg. 11.

Algorithm 11 Crossover operator

Input: population

\[
\begin{align*}
& n \leftarrow \text{population.size} \\
& \text{for } i \text{ from 1 to } n \text{ step 2 do} \\
& \quad \text{if rand.Next } \leq \text{crossOverRate then} \\
& \quad \quad c_1 \leftarrow \text{population[i-1].clone()} \\
& \quad \quad c_2 \leftarrow \text{population[i].clone()} \\
& \quad \quad c_1.\text{OnePointCrossOver}(c_2) \\
& \quad \quad c_1.\text{UpdateLayout} \\
& \quad \quad c_2.\text{UpdateLayout} \\
& \quad \quad \text{population.add}(c_1) \\
& \quad \quad \text{population.add}(c_2) \\
& \quad \text{end if} \\
& \text{end for}
\end{align*}
\]

6.10.3 Mutation Operator

The mutation operator is defined in the same way as for HPSO (Sec. 6.9.3). However, here we do not decrease the mutation rate as the iterations proceed and no constraints will be checked against the mutated chromosome.

6.10.4 The implemented Genetic Algorithm

The implemented genetic algorithm is described in Alg. 12. In the beginning, the population pool is initialized by random chromosomes and then each chromosome’s fitness value will be evaluated. Afterwards, the iterative process of solution finding will start. In each iteration, the pairs of parents are recombined via the crossover operator and some of the individuals in the population pool will be mutated based on the mutation rate. The newly generated chromosomes will then be evaluated and the selection operator will be applied to the whole population. The chromosome with the best fitness value in the population pool will be considered the best heuristic answer of the algorithm in the current iteration and the algorithm will then proceed to the next iteration.
Algorithm 12 Genetic algorithm

Initialize a population of chromosomes with random positions and orientations in the problem space.
Evaluate the fitness function for each chromosome.
while maxIteration reached do
    CrossOver pair of parents
    Mutate the resulting children
    Evaluate the fitness function for children
    EliteSelection (population)
    $G_{Best} \leftarrow$FindBestChromosome (population)
end while

The $G_{Best}$ chromosome is the individual with the best fitness value achieved throughout the whole optimization process.
Chapter 7

Experimental Results and Discussions

This chapter presents the experimental results obtained from the execution of all the algorithms we implemented to tackle the ACP problem, together with their analysis and validation, in order to completely answer our RQ3. First, an overall explanation of the experiments is brought. The impact of different numbers of particles for each iteration of PSO and HPSO is then examined. Consequently, the general results of each algorithm in each scenario are presented. Discussions on the obtained fitness values and optimality gaps in different scenarios is then provided. The next section contains the exploration and exploitation behaviours of PSO and HPSO. How the particles converge in PSO and HPSO is discussed later and, finally, the required computation time for the HPSO algorithm is brought.

7.1 Experiments

To evaluate the performance of HPSO in the ACP problem, we also implemented and applied the PSO and Genetic algorithms on the ACP problem. Since PSO, HPSO, and GA are all stochastic algorithms, the heuristic solutions for our ACP problem vary at every run of each algorithm. Each run constitutes of several iterations. To be able to draw reliable conclusions about the accuracy and robustness of the algorithms, several runs of various iterations were performed in each experiment. Additionally, in order to compare these different algorithms more accurately, we used the same set of random seeds for all algorithms in each run.

7.1.1 Experimental Settings

Our experiment was performed for three sample indoor scenarios with different levels of complexity. In the first two scenarios, all three algorithms were executed 1000 number of runs (i.e., replications) with 400 iterations at each run where the fitness functions were evaluated at each iteration. After analysing the results of these two scenarios, it was decided to run the third scenario for 500 replications
Chapter 7. Experimental Results and Discussions

of 200 iterations\(^1\).

As described in Sec. 7.2.6, the fitness function evaluations (including the calculations for the point visibility, FoV projection, and polygon intersection) has the most significant influence over the total processing time. Hence, having a fixed number of iterations for each scenario enabled us to objectively compare our three algorithms.

**Scenarios:** Our experiment was executed for three sample indoor scenarios. The level of complexity was gradually increased in the design of these scenarios, hence, the first scenario was the least, and the last scenario was the most, complicated.

The first scenario (Fig. 7.1) was designed as a floor layout of an apartment in the shape of a rectangular (area: 18.18 \(\times\) 11.24 $m^2$) with occlusions but without any RoI. Two identical camera types (Axis P3346)\(^2\) were considered as inputs of all three algorithms of PSO, HPSO, and GA. Since no RoI was placed in the scene, the optimization objective was only to find the maximum coverage of the floor layout.

![Figure 7.1: Scenario 1](image)

The second scenario (Fig. 7.2) was designed as a floor layout of another apartment in the shape of a rectangular (area: 14.75 \(\times\) 13.06 $m^2$) with three blocked RoIs (black boxes) and two essential RoIs (red boxes). The minimum camera resolution of both essential RoIs were assigned to 50 px/m. The optimization objective, in this case, was to maximize the coverage of the essential RoIs and avoid the blocked

---

\(^1\) The reason to decrease the numbers of runs and iterations for the third scenario was first because running 1000 replications of each algorithm seemed to be very time consuming without increasing the reliability of our results. That is due to the size of this scenario as well as the number of existing cameras and RoIs in the scene. Second, we realized that in the first two scenarios all three algorithms reached to a good enough answer at around 130\(^{th}\) iteration and, hence, 200 iterations (instead of 400) seemed to be reasonable.

\(^2\) See Table A.1
Chapter 7. Experimental Results and Discussions

RoIs while ensuring a maximum coverage of the rest of the scene. In this scenario, two identical camera types (Axis M3203)\(^3\) were passed to the algorithms.

![Scenario 2](image)

**Figure 7.2:** Scenario 2

The third scenario (Fig. 7.3) was designed as the largest and most complex floor layout of a bank in the shape of a triangle (area: \(\frac{50 \times 30 \times 30}{2} \text{ m}^2\)) with both blocked and essential RoIs. The required resolution of the essential RoIs were assigned to two different values: 25 px/m for the regions that necessitate recognition\(^4\), and 50 px/m for the areas that needed identification\(^5\). The optimization objective was the same as the second scenario with the exception of having four cameras of different types (Axis M3203, Axis M3204, Axis P3346, and Axis P3367-V)\(^6\) in the scene.

![Scenario 3](image)

**Figure 7.3:** Scenario 3

Table 7.1 summarizes our scenarios.

\(^3\) See Table A.1
\(^4\) Being able to recognize a person who was seen before, with a high degree of certainty.
\(^5\) Being able to identify a person beyond reasonable doubt.
\(^6\) See Table A.1
Table 7.1: Experimental settings for all scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Replications</th>
<th>Iterations</th>
<th>Scene Area</th>
<th>Blocked RoIs</th>
<th>Essential RoIs</th>
<th>Cameras</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>1000</td>
<td>400</td>
<td>18.18 × 11.24 m²</td>
<td>0</td>
<td>0</td>
<td>P3346, P3346</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>1000</td>
<td>400</td>
<td>14.75 × 13.06 m²</td>
<td>3</td>
<td>2</td>
<td>M3203, M3203</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>500</td>
<td>200</td>
<td>( \frac{50.30 \times 50.30}{2} ) m²</td>
<td>4</td>
<td>3</td>
<td>M3203, M3204, P3346, P3367-V</td>
</tr>
</tbody>
</table>

7.1.2 Swarm Size vs. Iterations

Depending on the optimization problem, PSO’s performance can be influenced when changing the number of particles in the swarm (Bratton and Kennedy, 2007). In order to find a good combination of the swarm size and the number of iterations for our ACP problem, we performed a small experiment for the second scenario. We intend to choose the combination that has the minimum average deviation from the best found solution. The experiment was done by running 20 replications of the PSO algorithm, each run with different number of particles and iterations while keeping the total fitness value evaluation numbers equal to 10,000. Hence, the algorithm was run with 10 particles and 1,000 iterations at first (10 × 1,000 = 10,000). The next 20 replications took place with 20 particles and 500 iterations (20 × 500 = 10,000) and so forth.

In Fig. 7.4 a sorted list of the percentage of the deviations from the best found solution (DBFS) in different runs is illustrated. In this figure the solutions obtained by PSO result in lower fitness values when having low or high particle numbers and performs better when having a medium swarm size (in our case, 25 particles). The empirical results in Bratton and Kennedy (2007) confirms our findings. Consequently, in our experiment, we chose to execute PSO with 25 particles and 400 iterations. Since we wanted to compare the performance of HPSO with PSO, we kept the same configuration for HPSO to see how much better it will perform in the best configuration of PSO. The number of chromosomes and iterations of the implemented GA were set to the same configuration as PSO and HPSO in order to verify which algorithm performs better in the same amount of time.

---

7 The best found solution is the one with the best fitness value throughout the whole experiment.

8 The run time is dependent on the number of evaluations, i.e., Number of Iterations × (Number of Particles or Chromosomes)
### Table 7.2: Particles Vs. Iterations - Scenario 2

<table>
<thead>
<tr>
<th>Particles</th>
<th>Iterations</th>
<th>DBFS Percentage</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1000</td>
<td>8.9%</td>
<td>P10I1000</td>
</tr>
<tr>
<td>20</td>
<td>500</td>
<td>6.0%</td>
<td>P20I500</td>
</tr>
<tr>
<td>25</td>
<td>400</td>
<td>2.4%</td>
<td>P25I400</td>
</tr>
<tr>
<td>40</td>
<td>250</td>
<td>4.9%</td>
<td>P40I250</td>
</tr>
<tr>
<td>50</td>
<td>200</td>
<td>6.3%</td>
<td>P50I200</td>
</tr>
</tbody>
</table>

In the Name column, ’P’ corresponds to the number of particles and ’I’ corresponds to the number of iterations.

### Figure 7.4: Deviation from best found solution - Scenario 2

#### 7.1.3 Configurations

In order to maintain the consistency of the experiment, we considered a number of basic as well as specific configurations. Table 7.3 presents the general configurations we used for all three scenarios. The Max Height parameter was set based on the average human height. The Min Height parameter was set to the ground level so that even a crawling object could be detected by the cameras. In general, the height on which the cameras should be installed is determined by the surveillance system designers based on the project’s characteristics. However, in indoor installations, like in our case, the cameras are usually installed close to the ceiling in order to have a maximal area coverage as well as to make it more difficult for unauthorized people to reach them. The Camera Installation Height parameter was, hence, set to 230 cm since the standard ceiling height of a room in most countries is 240 cm.
Chapter 7. Experimental Results and Discussions

Table 7.3: Base configuration - All scenarios

<table>
<thead>
<tr>
<th>Camera Installation Height</th>
<th>Min Height</th>
<th>Max Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>230 cm</td>
<td>0 cm</td>
<td>180 cm</td>
</tr>
</tbody>
</table>

In our scenarios, all of the three algorithms were configured to use 25 particles/chromosomes (as discussed in Sec. 7.1.2). The mutation rate for both the HPSO and GA was set to 0.35. Additionally, the HPSO’s greedy intervals was set to 15 iterations and GA’s crossover rate was set to 0.75. The mutation and crossover rates as well as the greedy intervals were selected based on various observations and trials.

7.1.4 General Results of the Experiments

The general results obtained from running all three algorithms on each of the scenarios are presented in Table 7.4. The Best Coverage column refers to the acquired scene coverage associated with the best fitness value found in each algorithm.

Table 7.4: Experiments’ results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Algorithm</th>
<th>Best Fitness Value</th>
<th>Best Coverage</th>
<th>Worst Fitness Value</th>
<th>Average Fitness Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>PSO</td>
<td>1,025,425</td>
<td>59.03%</td>
<td>623,075</td>
<td>857,843</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>HPSO</td>
<td>1,040,617</td>
<td>59.42%</td>
<td>776,283</td>
<td>916,144</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>GA</td>
<td>1,031,277</td>
<td>58.81%</td>
<td>605,153</td>
<td>898,019</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>PSO</td>
<td>49,176,194,048</td>
<td>46.78%</td>
<td>41,426,561,067</td>
<td>46,432,608,780</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>HPSO</td>
<td>49,176,199,171</td>
<td>47.57%</td>
<td>46,720,637,869</td>
<td>48,609,019,452</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>GA</td>
<td>49,176,197,713</td>
<td>47.19%</td>
<td>41,341,143,632</td>
<td>48,026,498,604</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>PSO</td>
<td>292,346,078,235</td>
<td>14.53%</td>
<td>133,977,857,316</td>
<td>240,010,054,953</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>HPSO</td>
<td>292,348,159,211</td>
<td>31.59%</td>
<td>148,714,323,884</td>
<td>287,369,122,840</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>GA</td>
<td>292,347,569,572</td>
<td>26.90%</td>
<td>134,963,557,273</td>
<td>275,818,241,256</td>
</tr>
</tbody>
</table>

Validation of the Experiments’ General Results

For the best found solution in each scenario, the cameras’ position and orientation can be viewed in both 2D (Figs. 7.5-A, 7.7-A, and 7.9-A) and 3D (Figs. 7.5-B, 7.7-B, and 7.9-B). As described in Sec. 4.3, the cameras are automatically placed exactly at the positions and orientations computed by the algorithms using the Importing Solution functionality in the CTSS (Figs. 7.5-B, 7.7-B, and 7.9-B). Figs. 7.6, 7.8, 7.10, and 7.11 illustrate the scenes seen by each installed camera.
Chapter 7. Experimental Results and Discussions

As it could be seen in Fig. 7.8-A, Camera 1 is placed near a wall and can see both essential RoIs with the pole avoiding it from capturing the room in which one of the blocked RoIs is located. In Fig. 7.8-B, Camera 2 is placed nearly in the middle of the scene and can capture the bigger essential RoI. As illustrated in this figure, Camera 2 can see the walls of the two rooms with blocked RoIs and not inside the rooms. Using a similar analysis, the validation of the results obtained by all three scenarios were done by the field experts with the help of the CTSS’ Camera View functionality (Sec. 4.3).

Figure 7.5: Scenario 1 - Final result: A- Optimizer B- SketchUp

Figure 7.6: Scenario 1 - Final result seen by: A- Camera 1 B- Camera 2
Figure 7.7: Scenario 2 - Final result: A- Optimizer B- SketchUp

Figure 7.8: Scenario 2 - Final result seen by: A- Camera 1 B- Camera 2

Figure 7.9: Scenario 3 - Final result: A- Optimizer B- SketchUp
Figure 7.10: Scenario 3 - Final result seen by: A- Camera 1 B- Camera 2

Figure 7.11: Scenario 3 - Final result seen by: A- Camera 3 B- Camera 4
7.2 Comparing HPSO with PSO and GA

In this section, we discuss and analyse the results acquired from comparing different aspects of our three implemented algorithms. With the help of these discussions we aim to answer our RQ3 (Sec. 2.1).

7.2.1 Average Fitness Values

Figs. 7.12 - 7.14 show how the average $G_{Best}$ fitness values for all 1000 replications of each algorithm grew in each iteration. In the first and second scenarios, all the three algorithms started with approximately same average fitness values in the first iteration. In the third scenario (Fig. 7.14), however, HPSO started off with the worst and PSO with the best average fitness values in the initial iterations. In all three scenarios, as the iterations proceeded, the solutions were improved with PSO having the worst and HPSO having the best average fitness values at their last iteration. An answer to the RQ3 could be obtained here indicating that HPSO performs best in terms of having better average fitness values after a number of iterations in comparison with PSO and GA.

![Figure 7.12: Average fitness values - Scenario 1](image1)

![Figure 7.13: Average fitness values - Scenario 2](image2)
7.2.2 Deviation from the Best Found Solution (DBFS)

As stated earlier, we are dealing with stochastic algorithms and the solutions we get differ based on how we seed the algorithm. In this part, we present and compare the DBFS associated with each run of PSO, HPSO, and GA. The difference between the global best fitness value of each run of each algorithm with the best found heuristic solution of all algorithms is considered as the DBFS. Figs. 7.15-A, 7.16-A, and 7.17-A show the comparisons between DBFS in each run of the HPSO and Genetic algorithms in relation to each other for all three scenarios. Consequently, Figs. 7.15-B, 7.16-B, and 7.17-B depict the comparisons between the DBFS in each run of HPSO and PSO algorithms in relation to each other. A look at these charts suggests that HPSO occupied less areas compared to the PSO and Genetic algorithms in all three scenarios. This means that in average, the results obtained by HPSO are closer to the best found solution for all three algorithms. Table 7.5 lists the averaged results of optimality gaps for our three algorithm in each scenario. As expected, HPSO outperforms the other two algorithms in all three scenarios. Therefore, again to answer the RQ3, HPSO performs best in terms of having less deviation from the best found solution in comparison with PSO and GA.
Figure 7.16: Scenario 2 - DBFS comparison: HPSO vs. GA and PSO

Figure 7.17: Scenario 3 - DBFS comparison: HPSO vs. GA and PSO

Table 7.5: Average DBFS comparison of PSO, HPSO, and GA for all scenarios

<table>
<thead>
<tr>
<th></th>
<th>Average Optimality Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSO</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>17.56%</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>5.58%</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>17.91%</td>
</tr>
</tbody>
</table>

7.2.3 Fitness Values

In Figs. 7.18 - 7.20 the charts showing sorted lists of the obtained $G_{Best}$’s fitness value at the end of each run for each algorithm are presented. HPSO, compared to the other two algorithms, shows to have $G_{Best}$ with the best fitness value at its worst run\(^9\) (i.e. best worst case performance). On the other hand, all algorithms seem to be capable of achieving a $G_{Best}$ with similar highest fitness value. However, it is obvious from the charts that the possibility of HPSO to reach this value is higher than the other two algorithms. In order to support this claim, for each algorithm, we calculated the fraction of the runs where the fitness values were at least equal to a Threshold value (Table 7.7). To find this value, we extracted the

\(^9\) The run in which each algorithm performed its worst, with the worst fitness value.
highest fitness value achieved (e.g. 1,040,617 belonging to the HPSO algorithm in the first scenario). We then decided to accept a 5% distance from the best solution of all algorithms, 95% \* Max(BestPSO, BestHPSO, BestGA), as our acceptable solution with 95% threshold from the best found solution by all algorithms. Table 7.6 shows the best found solutions in each algorithm and the 95% thresholds. HPSO, again, performs best in terms of most frequently reaching 95% threshold fitness values in comparison with PSO and GA, another result that can answer the RQ3.

**Table 7.6:** Best found solutions and 95% threshold for all scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Best PSO</th>
<th>Best HPSO</th>
<th>Best GA</th>
<th>95% threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>1,025,425</td>
<td>1,040,617</td>
<td>1,031,277</td>
<td>988,586</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>49,176,194,048</td>
<td>49,176,199,171</td>
<td>49,176,197,713</td>
<td>46,717,389,212</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>292,346,078,235</td>
<td>292,348,159,211</td>
<td>292,347,569,572</td>
<td>277,730,751,250</td>
</tr>
</tbody>
</table>

Table 7.7 contains the fraction of each algorithm’s solutions in 1000 replications that are equal or higher than the 95% Threshold in each scenario.

**Table 7.7:** Fraction of 95% threshold fitness values in each algorithm for all scenarios

<table>
<thead>
<tr>
<th>95% Threshold</th>
<th>PSO</th>
<th>HPSO</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>0.3%</td>
<td>19.3%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>55.6%</td>
<td>100%</td>
<td>90.1%</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>3.6%</td>
<td>86.8%</td>
<td>50.8%</td>
</tr>
</tbody>
</table>

![Figure 7.18: Sorted G_Best fitness values for each run - Scenario 1](image)
Figure 7.19: Sorted $G_{Best}$ fitness values for each run - Scenario 2

Figure 7.20: Sorted $G_{Best}$ fitness values for each run - Scenario 3
7.2.4 Exploration and Exploitation Comparison between HPSO and PSO

As discussed in Sec. 6.9, we expect to have enhanced exploration and exploitation behaviours in HPSO compared to those of PSO as results of our proposed modifications in HPSO.

Fig. 7.21a illustrates the movement path of a selected particle (a camera located at the position \((X,Y)\)) in a sample floor layout for 60 iterations using the PSO algorithm. In order to register a particle’s movement during its quest, we recorded its positions in each iteration and connected them using directed arrows. As expected, the result shows that after certain number of iterations the particle continues to oscillate around a certain point that will prevent it from further explorations. On the other hand, Fig. 7.21b depicts the movement path of the same particle with the same random seed for 60 iterations of the HPSO algorithm. The result shows a great enhancement in HPSO’s exploration ability while adhering to PSO’s systematic search dynamics. HPSO’s exploration ability could be a result of its implemented mutation operator.

\[\text{(a) PSO} \quad \text{(b) HPSO}\]

**Figure 7.21:** Sample particle exploration movement path

Fig. 7.22a illustrates the movement path of the swarm’s global best \((G_{\text{Best}})\) in a sample layout for 30 iterations using the PSO algorithm. As it can be seen, in the 4th move (iteration) the \(G_{\text{Best}}\) particle gets stuck in a region (the red circle) without much improvement for 7 iterations and then leaves the region. On the contrary, in Fig. 7.22b, it could be observed that the global best particle smoothly escapes from the region due to the hill-climbing operator introduced in the HPSO (Sec. 6.9.1). The hill-climbing algorithm aids the global best particle to be improved without adhering to the PSO dynamics (which causes the oscillation effect in the red circle area of Fig. 7.22a) and, hence, resulting in better exploitation of the best found solution so far.
Chapter 7. Experimental Results and Discussions

7.2.5 Particles’ Convergence

In order to realize how the particles converge as iterations go by, we calculate the corrected sample standard deviation (STDEV) of a sample run for Scenario 1. The corrected sample STDEVs for the swarm introduced in Fig. 7.23 are calculated based on the following formula:

\[
s = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2},
\]

(7.1)

where \((x_1, ..., x_n)\) are the particles’ fitness values and \(\bar{x}\) is the mean value of those fitnesses, while the denominator \(N\) stands for the swarm size (number of the particles). Lower values of the standard deviation indicate less scattered, and, hence, more converged average fitness values. As seen in Fig. 7.23, the PSO particles converge faster (around iteration number 100). However, the HPSO particles converge better than the PSO particles by reaching much lower standard deviation values. A fast convergence indicates premature convergence to local optima, which is the case with PSO. This problem is solved in the HPSO by introducing the mutation operator the addition of a randomness in the population might result in finding a better solution and escape being trapped in a local optima.

The fluctuations in HPSO’s graph is an indication of the introduced mutation operator for the particles. This operator increases the randomness in the swarm, making the particles converge slower compared to the PSO, but at the same time allowing more thorough search of the solution space.
Another interesting observation is PSO’s high standard deviation value after convergence. This means that the particles after their convergence are scattered in the search space in such a way that their standard deviation values are nearly the same. These high standard deviation values cannot be seen in HPSO’s graph when the iteration number increases. The reason lies behind the fact that we introduced the concept of forgetful particles (Sec. 6.9.4). With the forgetful particles, in cases where a particle oscillates between its personal best and local best without any improvement in a number of iterations, the particle will replace its personal best. With this strategy, HPSO can reach

![Figure 7.23: Particles’ convergence - PSO vs. HPSO](image)

### 7.2.6 Computation Time

As mentioned earlier, one aim of this thesis is to develop an interactive ACP-DSS. By interactivity we mean that ACP-DSS should suggest a heuristically good solution in a reasonable amount of time (1-3 minutes in our case) when using a normal desktop PC. Our whole system was implemented in the .Net framework using C# and WPF. To enable faster evaluations of different algorithm configurations, the C# implementation was optimized in the following ways:

- The polygon intersection library of WPF was used to compute the intersected geometries.
- Since we needed to completely avoid capturing the blocked RoIs, the fitness
evaluations for all these RoIs were done at the same time considering a minimum resolution of 1 for each camera (i.e. we cached the cone projection for the blocked RoIs).

- Several caches were introduced to the visibility detection part to avoid unnecessary conversions between Polar and Cartesian systems.

Taking into account the above optimizations, the HPSO spends most of its total computation time (around 74%) evaluating the fitness function. This includes calculating the FoV cone projection and polygon intersection. The visibility detection, on the other hand, requires about 13% of the total time. Table 7.8 presents the time spent on different modules that have the biggest impacts on the total time consumed by the HPSO algorithm. The results are averaged for 1000 runs of the C# implementation of HPSO with 130 iterations for Scenario 2.

<table>
<thead>
<tr>
<th>Module</th>
<th>Time (s)</th>
<th>Percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility Detection</td>
<td>4</td>
<td>12.90</td>
</tr>
<tr>
<td>Cone Projection</td>
<td>8</td>
<td>25.81</td>
</tr>
<tr>
<td>Polygon Intersection</td>
<td>15</td>
<td>48.39</td>
</tr>
<tr>
<td>Rest of the algorithm</td>
<td>4</td>
<td>12.90</td>
</tr>
<tr>
<td>Total</td>
<td>31</td>
<td>100</td>
</tr>
</tbody>
</table>

### 7.3 Validity Threats

A study’s validity corresponds to the extent to which its results are trustworthy and not influenced by the researchers’ subjective opinion (Runeson and Höst, 2009). In the literature various classification schemes are used to categorize validity and threats to validity aspects. In this thesis we chose the four following aspects of validity introduced in Runeson and Höst (2009).

**Construct Validity:** Construct validity reflects the extent to which the studied operational measures really represent what the researcher have in mind (Runeson and Höst, 2009).

The construct validity in our thesis could be discussed in two different areas. The first area is in evaluating CTSS. As mentioned earlier, we used different means (both formal and informal) to evaluate CTSS. Our online questionnaire’s feedback considered to be a formal feedback. The informal feedback included the ones we received after the Global A&E Conference as well as other frequent feedback from the field practitioners at Axis while implementing different versions of CTSS. Most of the feedback providers had previous experience in working with the existing surveillance design systems (i.e. Revit and Visio) and all of them were
completely familiar with the domain knowledge. Additionally, the entire system was presented to the respondents before getting any kind of feedback. Even though one respondent did not understand one of the questions, we believe that both in the questionnaire and in our informal enquiries, the questions were clear to the respondents to a great extent. Hence, not so many questions (if any) could have been interpreted differently compared to what we had in mind. However, since the system is going to be officially released to the public users (who may or may not be familiar with the domain knowledge) in the near future, we intend to continue improving the questionnaire by having its questions reviewed by a number of field and user experience experts to reduce the construct validity threats.

The second area concerns using 2D projections of FoV cones for the blocked RoIs considering dead zones in ACP-DSS. After performing the set of experiments and analysing the results, we found out that in some situations the 2D trapezoid projection of the FoV cones might not completely satisfy the blocked RoIs’ constraints and could, hence, be a construct validity threat. Although the difference is small and probably neglectable, it can still be avoided by considering normal fan-shaped FoV cone projections (without dead zone) solely when calculating the blocked RoIs’ specific fitness values.

Internal Validity: Internal validity is concerned with the examination of casual relations between different factors and the extent to which the relations between two investigated factors could be affected by a third factor (Runeson and Höst, 2009). While we introduced some modifications into the standard PSO algorithm in order to enhance the HPSO algorithm’s performance, we kept all the shared factors of both algorithms (including \(c_1, c_2, \omega_1, \omega_2\), number of particles, and the neighbourhood topology) fixed. By keeping the shared factors fixed and ensuring that the introduced operators in HPSO do not affect them, we believe that the threat to the internal validity of the research in designing HPSO-DSS is reduced.

External Validity: External validity deals with the extent to which the findings of the research are generalizable and the extent to which these findings are interesting for other researchers (Runeson and Höst, 2009). All proposed algorithms were tested in three different settings (scenarios). However, although we believe that our findings can be generalized in the context of ACP, with the problem constraints we considered and described in this thesis, it is not possible to claim that the thesis adequately validated all problem domains.

Reliability: Reliability is concerned with the extent to which the data and its analysis are dependent to a specific researcher (Runeson and Höst, 2009). We tried to minimize the threats to reliability and increase the repeatability by clearly
describing all important steps, activities, and algorithms used in this thesis so that other researchers could conduct the same study.
Chapter 8

Discussion and Concluding Remarks

In this chapter, discussions on the conclusions we arrived at after completing this thesis as well as our suggestions on the future work is presented.

The goal of this thesis was to improve the design process of surveillance systems. We identified two major areas of improvement: first, concerning the modelling environment and second, with regard to assisting the designers in good arrangement of cameras in closed layouts. Addressing these issues led us to three research questions and, thus, the design of two separate, and at the same time integrated, subsystems: HPSO-DSS integrated into CTSS. Answering all three research questions formed a basis to satisfy the main purpose of our thesis and, consequently, our main research question.

To enhance surveillance system design process, CTSS was designed through an iterative process. It was built by performing analysis on how to improve user experience in surveillance system design modelling, gathering stakeholders requirements, and then building and evaluating the system through workshops, questionnaires, and interviews. According to the received feedback, it proved to be a desirable modelling tool for the surveillance system designers, sales engineers, trainers, and other stakeholders, in terms of user experience and usability. The users expressed significant satisfaction level when performing their design using the final updated version (Fig. 4.3) compared to the first one (Fig. A.1). The first version itself, however, received a very high degree of satisfaction (more than 80% - Question 9 - Table A.17) with more than 90% of respondents believing that a quicker system design could be achieved by using CTSS rather than the camera toolsets in Revit and Visio (Question 13 - Table A.25). The results that came out of all received feedback could be considered as the answer to our RQ1.

In an integration with CTSS, a decision support system to suggest automatic camera placement was designed. The design of ACP-DSS was done by performing theoretical analysis in order to construct the visibility detection algorithm (Sec. 5.2) as well as identifying and defining the PSO optimization model (Chap. 6). In
this way we addressed RQ2 of our thesis. Additionally, part of RQ3 was answered by presenting a novel optimization algorithm (HPSO - Sec. 6.9.5) that used PSO as its basis while introducing a combination of different operators and concepts to improve its performance. The modifications applied on PSO were based on several step by step observations of its performance. We believe that the greedy strategy (Sec. 6.9.2) could improve PSO’s performance in terms of reaching better fitness values in less number of iterations. Using hill-climbing algorithm (Sec. 6.9.1) could, additionally, speed up the enhancements of the global best. The mutation operator (Sec. 6.9.3) helped keeping the diversity in the swarm and the forgetful particles (Sec. 6.9.4) assisted the particles to escape from fluctuating situations. According to the results obtained from the sets of experiments we performed at the end of this thesis, HPSO outperformed the other implemented algorithms. A complete answer to RQ3 was obtained here.

In conclusion, the integrated HPSO-DSS in CTSS allows the surveillance system designers to model their desired buildings using accurate measurements. They could then pass the floor layout of the monitoring area to the optimizer and visualize the result in their building model. The final result could afterwards be imported to SketchUp and validated by the designers using the features provided in CTSS. Further, if they were satisfied with the result, they can implement it in the real world. Having such an easy-to-use system can assist the designers with virtually simulating their camera planning process and avoid wasting resources such as time and money on creating inaccurate or inefficient systems in real world before testing them.

### 8.1 Future Work

We plan to expand CTSS by gathering and analysing public and non-domain expert users’ requirements and suggestions regarding user experience and usability as well as improving and updating the system based on the feedback.

Future work in PSO-based automatic camera placement may be in utilizing more recent alternatives of the PSO algorithm such as cooperative sub-swarm method (Van den Bergh and Engelbrecht, 2004). Another favourable area is to use a multi-objective PSO algorithm for optimizing the cost and number of cameras. This optimization should be done with great considerations due to the existence of many constraints as well as the need for human knowledge for choosing the right models of cameras in a given scene.

Furthermore, investigating the computational time of our algorithm prompted us about the variety of areas that might be improved by applying smarter algorithms to calculate the FoV projection and polygon intersection. We believe that the current implementation is time efficient enough for not loosing the system’s
interactiveness. However, it can be improved for more complex scenarios. One way of doing so is to cache the projected cones. Another way might be to more properly formulate the projected cones for different resolutions instead of completely reconstructing FoV visibility polygons. Additionally, an area of improvement could be the parallel implementation of different parts of the algorithm to benefit from multi-core processors that could be found in any personal computer these days.

Another area that could be more analysed is adding constraints on boxed camera installation places. Although we discussed how to solve the general ACP problem, in practice there are some preferences on how to place boxed cameras (Sec. 3.1) in a building. For dome cameras, the provided solution works perfectly but for the boxed cameras, more investigations on the problem definition should be done and probably some new constraints should be added. A sample additional constraint to be defined could be that the boxed cameras are usually mounted on the walls rather than on any arbitrary place throughout the scene.
Bibliography


BIBLIOGRAPHY


Appendix A

A.1 Figures

Figure A.1: CTSS First Version

A.2 Source Codes

A.2.1 Face Extractor (Ruby)

```ruby
def self.ExtractFace(filename)
    mod = Sketchup.active_model # Open model
    sel = mod.selection # Current selection
    f = File.open(EdgeExtractor.Path+filename, 'w')
    sel.each { |s|
        # Code...
    }
```

102
Figure A.2: Relation between Project object and camera components

```ruby
s.loops.each { |l|
  lastv = l.edges[l.edges.length - 1]
  lstart = lastv.start
  lend = lastv.end
  pt = nil
  l.edges.each { |v|
    pt = v.start
    if (pt != lstart && pt != lend)
      f.puts "%.2f,%.2f,%.2f" % [pt.position.x * 2.54, pt.position.y * 2.54, pt.position.z * 2.54]
    end
    pt = v.end
    if (pt != lstart && pt != lend)
      f.puts "%.2f,%.2f,%.2f" % [pt.position.x * 2.54, pt.position.y * 2.54, pt.position.z * 2.54]
    end
    lstart = v.start
    lend = v.end
  }
  f.puts "@@@@@@@@@"
A.2.2 Velocity Update (C#)

```csharp
public void UpdatePSOVelocities(GenData pBest, PSOGen lBest)
{
    const double w1 = .975;
    const double c1 = 2.05, c2 = 2.05;

    ClampVelocities();
}
```

A.2.3 Velocity Clamping (C#)

```csharp
public void ClampVelocities()
{
    if (VX > VXMax) VX = VXMax;
    if (VX < VXMin) VX = VXMin;
    if (VY > VYMax) VY = VYMax;
    if (VY < VYMin) VY = VYMin;
    if (VPan < VPanMin) VPan = VPanMin;
    if (VPan > VPanMax) VPan = VPanMax;
    if (V_Tilt < V_TiltMin) V_Tilt = V_TiltMin;
    if (V_Tilt > V_TiltMax) V_Tilt = V_TiltMax;
}
Procedure A.3 — Velocity Clamping

A.2.4 Particle Movement (C#)

```csharp
internal void PerformVelocityBasedFlight()
{
    double vx = Velocity.VX, vy = Velocity.VY;
    double X = 0, Y = 0, tmp = 0;
    bool handled = false;
    int count = 0;
    while (!handled && count < 8)
    {
        X = Data.Position.X + vx;
        Y = Data.Position.Y + vy;
        handled = Problem.Model.HitTest(X, Y);
        if (!handled)
        {
            switch (count)
            {
                case 0: vx /= 2; break;
                case 1: vx *= 2; vy /= 2; break;
                case 2: vx /= 2; break;
                case 3: tmp = vx; vx = 0; break;
                case 4: vx = tmp; tmp = vy; vy = 0; break;
                case 5: vy = tmp; vx = -vx; break;
                case 6: vx = -vx; vy = -vy; break;
                case 7: vx = -vx; break;
            }
            count++;
        }
    }
    Data.Position = new CartesianPoint(X, Y);
    Data.PanAngle += Velocity.VPan;
    Data.TiltAngle += Velocity.VTilt;
}
```

Procedure A.4 — Particle Movement
A.2.5 Neighbourhood Selection (C#)

```csharp
private void SinglyLinkedRecluster()
{
    foreach (PSOParticle particle in this.Particles)
    { particle.UnCluster();
        int n = Particles.Count;
        for (int i = 0; i < n; i++)
        {
            Particles[i].Neighbours.Add(Particles[(n + i - 2) % n]);
            Particles[i].Neighbours.Add(Particles[(n + i - 1) % n]);
            Particles[i].Neighbours.Add(Particles[i]);
            Particles[i].Neighbours.Add(Particles[(i + 1) % n]);
            Particles[i].Neighbours.Add(Particles[(i + 2) % n]);
        }
    }
}
```

**Procedure A.5 — Neighbourhood Selection**

A.2.6 Particle::UpdateFitness(C#)

```csharp
public virtual void UpdateFitnessValue()
{
    MakeTotalCoverageGeometry();
    double total = 0;
    foreach (RegionOfInterest roi in Problem.Model.RegionOfInterests)
    {
        roi.ResetForIteration(Problem, this);
        foreach (PSOGen gene in this.Genes)
                CalculateResTargetDistance(roi.MinRes));
            PathGeometry combined = GetTotalGenCoverage();
            roi.CalculateFitnessForParticle(Problem, combined, this);
            total += roi.Fitness;
        }
    PData.FitnessValue = total / Problem.Model.RegionOfInterests.Count * ←
                        Problem.Model.Area + PData.TotalCoverage;
    ...
}
```
A.3 Cameras used in the experiments

*Table A.1:* Camera’s intrinsic parameters used in our experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>Resolution</th>
<th>$\text{FoV}<em>{h</em>{\text{min}}}$</th>
<th>$\text{FoV}<em>{h</em>{\text{max}}}$</th>
<th>$\text{FocalLength}_{\text{min}}$</th>
<th>$\text{FocalLength}_{\text{max}}$</th>
<th>Default $\text{FocalLength}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis M3203</td>
<td>800 × 600</td>
<td>18</td>
<td>66</td>
<td>2.8</td>
<td>10</td>
<td>2.8</td>
</tr>
<tr>
<td>Axis M3204</td>
<td>1280 × 800</td>
<td>22</td>
<td>80</td>
<td>2.8</td>
<td>10</td>
<td>2.8</td>
</tr>
<tr>
<td>Axis P3346</td>
<td>2048 × 1536</td>
<td>30</td>
<td>84</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Axis P3367-V</td>
<td>2592 × 1944</td>
<td>30</td>
<td>84</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

A.4 Questionnaire

A.4.1 Questions

The list of questions in the questionnaire is as follows:

1. Please specify your name and your Email address:
2. How familiar are you with system design?
3. How difficult was it to place cameras on the scene?
4. How difficult was it to aim cameras at the targeted area?
5. How did you find using two panels to work with camera parameters and functionalities?
6. Are you satisfied with the amount of time it takes to complete the tasks?
7. How do you find it to interact with the tool?
8. What is the level of flexibility when using Axis toolbox in SketchUp?
9. How satisfied are you with Axis toolbox in SketchUp?
10. How difficult is it to correct mistakes while using Axis toolbox in SketchUp?
11. How difficult is it to become skilful at using Axis toolbox in SketchUp?
12. To what extent do you agree/disagree with this statement: Axis toolbox in SketchUp is designed for all levels of users.
13. Do you think that with using this tool, you can perform system design more quickly?
14. What work well with Visio and what don’t?
15. What work well with Revit and what don’t?
16. List the most negative and positive aspects of Axis toolbox in SketchUp.
17. Did you find any fault/error when using the toolbox?
18. What features do you think should be added to the toolbox?
19. How probable is it that you would use SketchUp for your next system design?
20. What is your suggestion for overall improvement of the toolbox in order to suit your needs?

A.4.2 Result Statistics

Axis Global Conference 2013: SketchUp Workshop
(Response rate: 53%)

Table A.2: Names of the respondents (the email addresses were removed from the table for privacy reasons.)

<table>
<thead>
<tr>
<th>Respondents’ Names</th>
<th>Respondents’ Names</th>
<th>Respondents’ Names</th>
<th>Respondents’ Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Claudio Musazzi</td>
<td>Jindrich Světnica</td>
<td>Mani Gurung</td>
<td>Paulo Silva</td>
</tr>
<tr>
<td>Ettiene</td>
<td>Gavin Daly</td>
<td>Mariana Rojas</td>
<td>Philippe</td>
</tr>
<tr>
<td>Sanchez xavier</td>
<td>Viktor Svärm</td>
<td>Staffan Olsson</td>
<td>Justin Casanave</td>
</tr>
<tr>
<td>Peter Gustafsson</td>
<td>Noriyoshi Okumoto</td>
<td>Yohei Nagata</td>
<td>Michael Chen</td>
</tr>
<tr>
<td>Denis Lyapin</td>
<td>russell</td>
<td>Anders Grimsberg</td>
<td>Leonardo Cossio</td>
</tr>
<tr>
<td>peter grau</td>
<td>Rodrigo</td>
<td>Alan Paterson</td>
<td>Edwin Brokke</td>
</tr>
<tr>
<td>Pavel Rozhkov</td>
<td>VIKTOR</td>
<td>Jörg Rech</td>
<td>Ian James</td>
</tr>
<tr>
<td>Chris Tangsilsat</td>
<td>michel barge</td>
<td>Aleksey Novak</td>
<td>zhigang, yang</td>
</tr>
<tr>
<td>Miloš Kohout</td>
<td>Ajith Surendran</td>
<td>Assad</td>
<td>Nabi Hacigaffaroglu</td>
</tr>
<tr>
<td>Sonny Hardarsson</td>
<td>FranciscoRodriguez</td>
<td>Mattias Wiberg</td>
<td>Steve Burdet</td>
</tr>
<tr>
<td>Doug Adams</td>
<td>Lina He</td>
<td>Patrik</td>
<td>Paolo Mura</td>
</tr>
<tr>
<td>chris liu</td>
<td>Nelson Woo</td>
<td>David Luna</td>
<td>Wilson Tang</td>
</tr>
<tr>
<td>Joseph Kan</td>
<td>Raul Alvaro Fraser</td>
<td>Yuko Tomono</td>
<td>Kuppuswamy</td>
</tr>
<tr>
<td>patrick p</td>
<td>Vincent Lin</td>
<td>Gokulan Pathmanaban</td>
<td>Donato Testa</td>
</tr>
<tr>
<td>Stephen Wong</td>
<td>Markus Lai</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A.3: Answers to Question No. 2: How familiar are you with system design?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not at all familiar</td>
<td>19%</td>
<td>11</td>
</tr>
<tr>
<td>Have a little experience with Axis toolbox in Visio</td>
<td>53%</td>
<td>31</td>
</tr>
<tr>
<td>Have a little experience with Axis toolbox in Revit</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td>Have expert experience with Axis toolbox in Visio</td>
<td>21%</td>
<td>12</td>
</tr>
<tr>
<td>Have expert experience with Axis toolbox in Revit</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Have experience with system design but not with Axis toolbox in Revit/Visio</td>
<td>10%</td>
<td>6</td>
</tr>
<tr>
<td>NA</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Comment please...</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.4: Comments to Question No. 2

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Not physical design like this. More from a software architectural side I have some experience.</td>
</tr>
<tr>
<td>2.</td>
<td>Work with it on Trainings - Design Class</td>
</tr>
<tr>
<td>3.</td>
<td>camera placement is only one part of the system design. also bandwidth, storage, intelligence, etc. are part of it.</td>
</tr>
<tr>
<td>4.</td>
<td>We need to have more training on the system design tools, and also need to organize webinar for consultants and partners</td>
</tr>
</tbody>
</table>

Table A.5: Answers to Question No. 3: How difficult was it to place cameras on the scene?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely difficult</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Difficult</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Fairly difficult</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>Fairly easy</td>
<td>40%</td>
<td>23</td>
</tr>
<tr>
<td>Easy</td>
<td>38%</td>
<td>22</td>
</tr>
<tr>
<td>Extremely easy</td>
<td>10%</td>
<td>6</td>
</tr>
<tr>
<td>NA</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Please comment...</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>
Table A.6: Comments to Question No. 3

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>as a first time user it was a little tricky at first but then became rather easy to place the camera.</td>
</tr>
<tr>
<td>2.</td>
<td>I managed to place a camera “floating” and couldn’t get it on a surface until I reopened the floor plan</td>
</tr>
<tr>
<td>3.</td>
<td>it is a different way of working that I was not used to.</td>
</tr>
<tr>
<td>4.</td>
<td>it was somewhat difficult because I was new to the system. However, it appears easy to place them.</td>
</tr>
</tbody>
</table>

Table A.7: Answers to Question No. 4: How difficult was it to aim cameras at the targeted area?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely difficult</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Difficult</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td>Fairly difficult</td>
<td>16%</td>
<td>9</td>
</tr>
<tr>
<td>Fairly easy</td>
<td>45%</td>
<td>26</td>
</tr>
<tr>
<td>Easy</td>
<td>19%</td>
<td>11</td>
</tr>
<tr>
<td>Extremely easy</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>NA</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Please comment...</td>
<td>9%</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td></td>
<td>58</td>
</tr>
</tbody>
</table>

Table A.8: Comments to Question No. 4

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>It takes a long time of clicking and dragging on top of menu clicking to get to the “Camera Eye”</td>
</tr>
<tr>
<td>2.</td>
<td>It would have made it easier if one could select some item in the scene that the should be the center of the camera aim. The chair in the example for instance. Even a possibility to graphically adjust the cone would be better.</td>
</tr>
<tr>
<td>3.</td>
<td>“grab and direct” is needed</td>
</tr>
<tr>
<td>4.</td>
<td>Better to be able to rotate instead of putting a number in a box</td>
</tr>
<tr>
<td>5.</td>
<td>not very intuitive. Should be some sort of slider or angle button.</td>
</tr>
</tbody>
</table>
Table A.9: Answers to Question No. 5: How did you find using two panels to work with camera parameters and functionalities?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely difficult</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Difficult</td>
<td>10%</td>
<td>6</td>
</tr>
<tr>
<td>Fairly difficult</td>
<td>21%</td>
<td>12</td>
</tr>
<tr>
<td>Fairly easy</td>
<td>31%</td>
<td>18</td>
</tr>
<tr>
<td>Easy</td>
<td>26%</td>
<td>15</td>
</tr>
<tr>
<td>Extremely easy</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>NA</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Please comment...</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.10: Comments to Question No. 5

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>One panel consolidating all the camera setting would be fantastic</td>
</tr>
<tr>
<td>2.</td>
<td>Change rotation =&gt; pan for the dome camera</td>
</tr>
<tr>
<td>3.</td>
<td>setting up camera angle and resolution like the VISIO tools are more easy then put an angle in the drawing.</td>
</tr>
</tbody>
</table>

Table A.11: Answers to Question No. 6: Are you satisfied with the amount of time it takes to complete the tasks?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely unsatisfied</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Unsatisfied</td>
<td>9%</td>
<td>5</td>
</tr>
<tr>
<td>Fairly unsatisfied</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>Fairly satisfied</td>
<td>24%</td>
<td>14</td>
</tr>
<tr>
<td>Satisfied</td>
<td>41%</td>
<td>24</td>
</tr>
<tr>
<td>Extremely satisfied</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td>NA</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>Please comment...</td>
<td>10%</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>
Table A.12: Comments to Question No. 6

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>it will take some practice, just like learning any new program</td>
</tr>
<tr>
<td>2</td>
<td>Goes really fast to sketch up a project</td>
</tr>
<tr>
<td>3</td>
<td>It was too short to finish</td>
</tr>
<tr>
<td>4</td>
<td>It was too short to finish</td>
</tr>
<tr>
<td>5</td>
<td>To early to judge. Need more experience first.</td>
</tr>
<tr>
<td>6</td>
<td>Timeslot is too short</td>
</tr>
<tr>
<td>6</td>
<td>We only had 30 min to perform the tasks and we had no one there to help.</td>
</tr>
</tbody>
</table>

Table A.13: Answers to Question No. 7: How do you find it to interact with the tool?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely ambiguous</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Somewhat ambiguous</td>
<td>12%</td>
<td>7</td>
</tr>
<tr>
<td>Somewhat clear</td>
<td>28%</td>
<td>16</td>
</tr>
<tr>
<td>Clear and understandable</td>
<td>49%</td>
<td>28</td>
</tr>
<tr>
<td>Extremely clear and understandable</td>
<td>4%</td>
<td>2</td>
</tr>
<tr>
<td>NA</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Please comment...</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>Total Responses</td>
<td></td>
<td>57</td>
</tr>
</tbody>
</table>

Table A.14: Comments to Question No. 7

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Some better written instructions I think would help</td>
</tr>
<tr>
<td>2</td>
<td>Required a bit more practice to learn interface</td>
</tr>
<tr>
<td>3</td>
<td>The time was a little bit short for all tasks</td>
</tr>
</tbody>
</table>
Table A.15: Answers to Question No. 8: What is the level of flexibility when using Axis toolbox in SketchUp?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely rigid</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Rigid</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Slightly rigid</td>
<td>9%</td>
<td>5</td>
</tr>
<tr>
<td>Slightly flexible</td>
<td>20%</td>
<td>11</td>
</tr>
<tr>
<td>Flexible</td>
<td>57%</td>
<td>32</td>
</tr>
<tr>
<td>Extremely flexible</td>
<td>4%</td>
<td>2</td>
</tr>
<tr>
<td>NA</td>
<td>4%</td>
<td>2</td>
</tr>
<tr>
<td>Comment please...</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>56</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.16: Comments to Question No. 8

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>a click and drag feature for panning/tilting/zoom in/out would be very useful</td>
</tr>
<tr>
<td>2.</td>
<td>I don’t understand the question.</td>
</tr>
<tr>
<td>3.</td>
<td>little buggy sometime, the screen didn’t refresh regularly</td>
</tr>
<tr>
<td>4.</td>
<td>don’t know yet</td>
</tr>
</tbody>
</table>

Table A.17: Answers to Question No. 9: How satisfied are you with Axis toolbox in SketchUp?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely frustrated</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Frustrated</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Slightly frustrated</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>Slightly satisfied</td>
<td>14%</td>
<td>8</td>
</tr>
<tr>
<td>Satisfied</td>
<td>60%</td>
<td>35</td>
</tr>
<tr>
<td>Extremely satisfied</td>
<td>12%</td>
<td>7</td>
</tr>
<tr>
<td>NA</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>Please comment...</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>51</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.18: Comments to Question No. 9

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>i think it is a fantastic tool for a sales engineer</td>
</tr>
<tr>
<td>2.</td>
<td>good start</td>
</tr>
<tr>
<td>3.</td>
<td>needs more work, but looks good.</td>
</tr>
</tbody>
</table>
Table A.19: Answers to Question No. 10: How difficult is it to correct mistakes while using Axis toolbox in SketchUp?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely difficult</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Difficult</td>
<td>4%</td>
<td>2</td>
</tr>
<tr>
<td>Fairly difficult</td>
<td>28%</td>
<td>15</td>
</tr>
<tr>
<td>Fairly easy</td>
<td>35%</td>
<td>19</td>
</tr>
<tr>
<td>Easy</td>
<td>19%</td>
<td>10</td>
</tr>
<tr>
<td>Extremely easy</td>
<td>4%</td>
<td>2</td>
</tr>
<tr>
<td>NA</td>
<td>6%</td>
<td>3</td>
</tr>
<tr>
<td>Please comment...</td>
<td>6%</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>54</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.20: Comments to Question No. 10

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>difficult, but this is not so much a limitation of the tool i think as it is in sketchup. but, am sure this will improve over time as my familiarity increases</td>
</tr>
<tr>
<td>2.</td>
<td>don’t know</td>
</tr>
<tr>
<td>3.</td>
<td>it’s easier to delete camera and add than correct mistake for me</td>
</tr>
</tbody>
</table>

Table A.21: Answers to Question No. 11: How difficult is it to become skillful at using Axis toolbox in SketchUp?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely difficult</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Difficult</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>Fairly difficult</td>
<td>12%</td>
<td>7</td>
</tr>
<tr>
<td>Fairly easy</td>
<td>41%</td>
<td>24</td>
</tr>
<tr>
<td>Easy</td>
<td>34%</td>
<td>20</td>
</tr>
<tr>
<td>Extremely easy</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td>NA</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Please comment...</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.22: Comments to Question No. 11

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>difficult to say at this point as had very limited exposure to it. i could provide more accurate feedback in short term future</td>
</tr>
<tr>
<td>2.</td>
<td>need more time to learn and judge the system</td>
</tr>
</tbody>
</table>
Table A.23: Answers to Question No. 12: To what extent do you agree/disagree with this statement: Axis toolbox in SketchUp is designed for all levels of users.

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completely disagree</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Disagree</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Somewhat disagree</td>
<td>17%</td>
<td>10</td>
</tr>
<tr>
<td>Somewhat agree</td>
<td>36%</td>
<td>21</td>
</tr>
<tr>
<td>Agree</td>
<td>36%</td>
<td>21</td>
</tr>
<tr>
<td>Completely agree</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>NA</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Please comment...</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.24: Comments to Question No. 12

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>i don't believe this is suitable to all users. Non-technical people will have trouble (not so much with axis toolbox itself) but with Sketchup program.</td>
</tr>
<tr>
<td>2.</td>
<td>agree but need academy class</td>
</tr>
</tbody>
</table>

Table A.25: Answers to Question No. 13: Do you think that with using this tool, you can perform system design more quickly?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completely disagree</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Disagree</td>
<td>0%</td>
<td>0</td>
</tr>
<tr>
<td>Somewhat disagree</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td>Somewhat agree</td>
<td>24%</td>
<td>14</td>
</tr>
<tr>
<td>Agree</td>
<td>52%</td>
<td>30</td>
</tr>
<tr>
<td>Completely agree</td>
<td>12%</td>
<td>7</td>
</tr>
<tr>
<td>NA</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Please comment...</td>
<td>3%</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.26: Comments to Question No. 13

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>for smaller systems yes. For larger systems no. Example, 600 cameras in a prison. We cannot re-draw the prison for camera placement. It would take a huge amount of time.</td>
</tr>
<tr>
<td>2.</td>
<td>it definitely won’t be more quickly. It will put more effort on the person using it as whoever sees it will expect so much more from the person designing it.</td>
</tr>
</tbody>
</table>
## Table A.27: Answers to Question No. 14: What work well with Visio and what don’t?

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Visio works fine. The problem is that you can not see the blind spots and cannot export to CAD.</td>
</tr>
<tr>
<td>2.</td>
<td>Importing plans do not work well, it is good for designing network schemes I guess</td>
</tr>
<tr>
<td>3.</td>
<td>A bit more difficult to startup and when defining page layout and/or scale settings.</td>
</tr>
<tr>
<td>4.</td>
<td>Visio scales are always a problem</td>
</tr>
<tr>
<td>5.</td>
<td>Works Well, - quick deployment of camera FOV calulations in 2D - layer features Not works well: axis tool missing accessories visio user interface is awkward</td>
</tr>
<tr>
<td>6.</td>
<td>Camera placement is reallyu fast, resizing the map is not</td>
</tr>
<tr>
<td>7.</td>
<td>Angle of view, number of pixels / 2D</td>
</tr>
<tr>
<td>8.</td>
<td>Adaptation of the scale is very difficult</td>
</tr>
<tr>
<td>9.</td>
<td>Pixel req and pixel density at the end</td>
</tr>
<tr>
<td>10.</td>
<td>N/A</td>
</tr>
<tr>
<td>11.</td>
<td>NA</td>
</tr>
<tr>
<td>12.</td>
<td>I haven’t worked with it.</td>
</tr>
<tr>
<td>13.</td>
<td>Pretty good commonly, but its flat. No ability to see exactly what camera sees.</td>
</tr>
<tr>
<td>14.</td>
<td>3D and camera angle of view work well than visio.</td>
</tr>
<tr>
<td>15.</td>
<td>-</td>
</tr>
<tr>
<td>16.</td>
<td>The DWG importing not always work</td>
</tr>
<tr>
<td>17.</td>
<td>2D is fine in visio but scale and adapting scale was very tricky</td>
</tr>
<tr>
<td>18.</td>
<td>Importing CAD Drawings is not easy. Most of the time they not appear in the Visio.</td>
</tr>
<tr>
<td>19.</td>
<td>System design is OK if cameras are placed far art. However in a very dense design - a lot of cameras in a small space - the drawing gets very unclear. Also, changing lens angle of view is difficult. What does work is the case of use and click and drag interface.</td>
</tr>
</tbody>
</table>
| 20. | make a design based on given goals per camera (detection/recognition/identification). the tool in unfortunately only for standard AXIS deliverd lenses and not with optional lenses. |}

Continued on next page
### Table A.27: Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>I liked the 3D aspect and the ability to place objects in the scene. I felt it was difficult to place the cameras because I was new at the system. Also, I did not know much about making corrections.</td>
</tr>
<tr>
<td>30</td>
<td>easy to design a system getting the drawing scale correct - especially when using a cad drawing</td>
</tr>
<tr>
<td>31</td>
<td>Haven't used</td>
</tr>
<tr>
<td>32</td>
<td>Visio is a useful product: major difficulties are about to import some DWG with more layers coming from CAD and to configure the correct scale. For the rest is fairly easy create projects with Visio.</td>
</tr>
<tr>
<td>33</td>
<td>Placing camera and find out the coverage</td>
</tr>
<tr>
<td>34</td>
<td>Almost everything work well, but it’s complicated to configure it if you don’t know Visio perfectly.</td>
</tr>
<tr>
<td>35</td>
<td>-</td>
</tr>
<tr>
<td>36</td>
<td>Did not try visio</td>
</tr>
<tr>
<td>37</td>
<td>Does not crash, stable software, over all seems like a very purpose built application which fits camera design needs Not overly open, does not requires user to get familiar with software , easy to drag and drop icons.</td>
</tr>
<tr>
<td>38</td>
<td>It’s easy to use and create the dimension then Visio.</td>
</tr>
<tr>
<td>39</td>
<td>Visio stencil, put the camera on drawing and go</td>
</tr>
<tr>
<td>40</td>
<td>no 3D in visio</td>
</tr>
</tbody>
</table>

### Table A.28: Answers to Question No. 15: What work well with Revit and what don’t?

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Have no experience with Revit yet.</td>
</tr>
<tr>
<td>2</td>
<td>too complex for beginners, but great if you know how to use it.</td>
</tr>
<tr>
<td>3</td>
<td>I didn’t tried yet as I’m not having yet a Revit license.</td>
</tr>
<tr>
<td>4</td>
<td>N/A: i have not used it</td>
</tr>
<tr>
<td>5</td>
<td>Have not worked with it</td>
</tr>
<tr>
<td>6</td>
<td>Path Throught the Walls, pole</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>NA</td>
</tr>
<tr>
<td>10</td>
<td>I haven't worked with it.</td>
</tr>
<tr>
<td>11</td>
<td>Did not use it</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>never used</td>
</tr>
<tr>
<td>14</td>
<td>Never used.</td>
</tr>
</tbody>
</table>

*Continued on next page*
Table A.28 – Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.</td>
<td>NA</td>
</tr>
<tr>
<td>16.</td>
<td>-</td>
</tr>
<tr>
<td>17.</td>
<td>not familiar</td>
</tr>
<tr>
<td>18.</td>
<td>I have never used it</td>
</tr>
<tr>
<td>19.</td>
<td>The software takes obstructions such as walls into consideration</td>
</tr>
<tr>
<td>20.</td>
<td>I don’t no</td>
</tr>
<tr>
<td>21.</td>
<td>I do not know</td>
</tr>
<tr>
<td>22.</td>
<td>No experience with Revit</td>
</tr>
<tr>
<td>23.</td>
<td>Revit is very expensive and require a bigger level of knowledge to use it well.</td>
</tr>
<tr>
<td>24.</td>
<td>N/A</td>
</tr>
<tr>
<td>25.</td>
<td>n/a</td>
</tr>
<tr>
<td>26.</td>
<td>Haven’t used</td>
</tr>
<tr>
<td>27.</td>
<td>I don’t have experience with Revit</td>
</tr>
<tr>
<td>28.</td>
<td>Haven’t try yet. I don’t have AutoCAD software</td>
</tr>
<tr>
<td>29.</td>
<td>-</td>
</tr>
<tr>
<td>30.</td>
<td>Not sure what Revit is</td>
</tr>
<tr>
<td>31.</td>
<td>Don’t use</td>
</tr>
<tr>
<td>32.</td>
<td>don’t need to learn skill from Autodesk</td>
</tr>
<tr>
<td>33.</td>
<td>Revit is too complicate</td>
</tr>
</tbody>
</table>

Table A.29: Answers to Question No. 16: List the most negative and positive aspects of Axis toolbox in SketchUp.

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Most positive is that SketchUp is free. Event with the Pro version the price is not that dramatic. Easy to operate when you know how :) I can imagine that working in 3D could be hard for some people who don’t have previous experiences.</td>
</tr>
<tr>
<td>2.</td>
<td>none that I have noticed but it will come up</td>
</tr>
<tr>
<td>3.</td>
<td>So, far only positive things such as, intuitive GUI and nice tools allowing 3D design.</td>
</tr>
<tr>
<td>4.</td>
<td>A little complex if you’re not trained on the program</td>
</tr>
<tr>
<td>5.</td>
<td>positive: its very visually impressive negative: sketchup has a slightly steeper learning curve that will take some more time to learn. simple is always better.</td>
</tr>
<tr>
<td>6.</td>
<td>Easy to create floors walls, rooftops and pieces of furniture. Hard to place cameras (long time)</td>
</tr>
<tr>
<td>7.</td>
<td>Simple and efficient</td>
</tr>
<tr>
<td>8.</td>
<td>Fast and easy. 3D. Sketchup is easy to work with. Negative - none so far</td>
</tr>
</tbody>
</table>

Continued on next page
# Table A.29 – Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Pros: 1. Allows the user to elaborate on camera placement and provides instant feedback. 2. Integrated in a great tool that makes it really easy to build up scenes. Cons: 1. The orientation of the cameras could be more intuitive.</td>
</tr>
<tr>
<td>10</td>
<td>It’s better to have a general text book for SketchUp. Maybe Google provides it??</td>
</tr>
<tr>
<td>11</td>
<td>Totally good, just get used to Skrutchup</td>
</tr>
<tr>
<td>12</td>
<td>Negative - no PTZ cameras? Positive - impressive to show and easy to use</td>
</tr>
<tr>
<td>13</td>
<td>Some bugs found, not all the cameras available there - but for sure that will be resolved. Very very nice application for the market. As an idea - probably to add wiring going to cameras, ability to place switches, servers, power supplies etc - with no branding of course - just for being able to demonstrate there whole the system look and give name or comment to a generic component listed above. So it becomes not only CCTV but a complex system.</td>
</tr>
<tr>
<td>14</td>
<td>more axis models included in AXIS toolbox.</td>
</tr>
<tr>
<td>15</td>
<td>+ Easy to use - Explain that it is easy, do not be afraid</td>
</tr>
<tr>
<td>16</td>
<td>good to have all camera Parameters there but it is some clicks needed to reach the menu again to Change the Parameters. maybe a Icon on the toolbar on top would be nice to reach Parameter Settings of a camera</td>
</tr>
<tr>
<td>17</td>
<td>Powerful application with excellent views and verifications. Seems like easy to use, but I don’t feel like any user or installer can handle. Probably is much better than other tools in the market, but is not intuitive enough.</td>
</tr>
<tr>
<td>18</td>
<td>Positive - end result gives a very clear indication of the project. Negative - difficult to learn and use.</td>
</tr>
<tr>
<td>19</td>
<td>angle of view difficult to setup no idea what the image quality is at a given distance, like in VISIO</td>
</tr>
<tr>
<td>20</td>
<td>+ very easy to learn and to use, 3D - quite difficult to direct camera</td>
</tr>
<tr>
<td>21</td>
<td>I need more experience with sketch up to answer this question</td>
</tr>
<tr>
<td>22</td>
<td>Manual figures for fov etc.</td>
</tr>
<tr>
<td>23</td>
<td>Negative: Most icons in the toolbox look very similar so sometimes difficult to intuitively select the right one without sending a lot of time learning where each icon is located on the toolbar first. A lot of times I found myself clicking on the correct icon mainly because I memorized where it was meant to be, as opposed to recognizing it’s graphics. Positive: With a few days practice, it can become a fairly easy yet powerful tool</td>
</tr>
<tr>
<td>24</td>
<td>change the angle of view - negative</td>
</tr>
<tr>
<td>25</td>
<td>- not many layouts for projects would be easy to rebuild to SketchUp</td>
</tr>
<tr>
<td>26</td>
<td>No negatives so far, since its a new tool and need more time to explore the possibilities. 1. 3D visualization is very useful 2. Very good as a presentation tool</td>
</tr>
<tr>
<td>27</td>
<td>I think it could be a very popular tool and can help integrators to create the 3D enviroments they need.</td>
</tr>
</tbody>
</table>

*Continued on next page*
Table A.29 – Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.</td>
<td>3D is big. Looking through the cameras is also helpful a lot. Downside, I worry about the time it will take to create those details. It was not super simple to use, but then again what system is at first?</td>
</tr>
<tr>
<td>29.</td>
<td>hard to position the field of view by using the information window - dynamic placement/moving of cone would be better</td>
</tr>
<tr>
<td>30.</td>
<td>Have to change values to rotate the camera etc. It would be nice to make all the changes directly on the graphics. Not all the values are self-explanatory. You can get direct feedback in the graphics most of the time.</td>
</tr>
<tr>
<td>31.</td>
<td>Most negative: * Not clear where to find the options for a selected camera. * When selecting camera you can very well select a completely different camera just because you where “standing” in the other camera’s view cone. Most positive: Once you get used to the way it works and you understand how to see things and think. It is easy to use</td>
</tr>
<tr>
<td>32.</td>
<td>Very useful the possibility to download objects to introduce in the drawing I appreciate the possibility to widely modify camera specifications</td>
</tr>
<tr>
<td>33.</td>
<td>Insert object is easy. Simple GUI. Good drawing 3D function. For the negative, need more time to use the tools.</td>
</tr>
<tr>
<td>34.</td>
<td></td>
</tr>
<tr>
<td>35.</td>
<td>Negative: Lack of experience in SketchUp Positive: Suits our needs</td>
</tr>
<tr>
<td>36.</td>
<td>I think it is a little bit difficult to change camera angle. I want to adjust more visually.</td>
</tr>
<tr>
<td>37.</td>
<td>Crashes often, (SketchUp has for years). Very open, requires user to get familiar with software before he can design a system. Conceptualize 3d can be hard for some.</td>
</tr>
<tr>
<td>38.</td>
<td>positive: 3d ambient very easy to manage, the eye view are great thing negative: the toolbox icon are not very userfriendly</td>
</tr>
<tr>
<td>39.</td>
<td>easy to use</td>
</tr>
</tbody>
</table>

Table A.30: Answers to Question No. 17: Did you find any fault/error when using the toolbox?

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Target resolution was not working well. According to Axis lens calculator the numbers were wrong.</td>
</tr>
<tr>
<td>2.</td>
<td>no</td>
</tr>
<tr>
<td>3.</td>
<td>Not yet.</td>
</tr>
<tr>
<td>4.</td>
<td>No</td>
</tr>
<tr>
<td>5.</td>
<td>No</td>
</tr>
<tr>
<td>6.</td>
<td>I managed to place a camera on the air, the camera couldn’t be placed on the wall (surfaces) for some reason. Had to close the map and reopen.</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.30 – Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>not during my short session</td>
</tr>
<tr>
<td>8.</td>
<td>Some issues with the UI not properly drawn. Was told though that it was due to a new version of Sketchup was released.</td>
</tr>
<tr>
<td>9.</td>
<td>No</td>
</tr>
<tr>
<td>10.</td>
<td>yes sometimes the screen won't refresh, and strange things happen when you click on the camera a few times.</td>
</tr>
<tr>
<td>11.</td>
<td>Fixed cameras button named PTZ camera Required to give name to camera even after I chose required model, either it changes later to default name - not sure but P1344 most probably for fixed one.</td>
</tr>
<tr>
<td>12.</td>
<td>sometime the application error. need restart the application.</td>
</tr>
<tr>
<td>13.</td>
<td>-</td>
</tr>
<tr>
<td>14.</td>
<td>No</td>
</tr>
<tr>
<td>15.</td>
<td>not yet ;-)</td>
</tr>
<tr>
<td>16.</td>
<td>No</td>
</tr>
<tr>
<td>17.</td>
<td>No</td>
</tr>
<tr>
<td>18.</td>
<td>-</td>
</tr>
<tr>
<td>19.</td>
<td>oh yeah :) 1. poor Axis portfolio 2. bugs with camera naming 3. it crashes sometimes</td>
</tr>
<tr>
<td>20.</td>
<td>I had problems with my installation I get every time an error message after I modified something</td>
</tr>
<tr>
<td>21.</td>
<td>no</td>
</tr>
<tr>
<td>22.</td>
<td>no</td>
</tr>
<tr>
<td>23.</td>
<td>Haven’t explored the tool yet.</td>
</tr>
<tr>
<td>24.</td>
<td>The tool works very fine. The two difficult parts was getting access to the menus and to direct the camera view. But still it was only an hour practice.</td>
</tr>
<tr>
<td>25.</td>
<td>did it have all the axis cameras? Did it have generic cameras for putting in non-axis cameras into the system?</td>
</tr>
<tr>
<td>26.</td>
<td>some of the images of cameras looked incorrect?</td>
</tr>
<tr>
<td>27.</td>
<td>No</td>
</tr>
<tr>
<td>28.</td>
<td>During the session the toolbox crashed while I was choosing the camera to import in the schema</td>
</tr>
<tr>
<td>29.</td>
<td>So far not.</td>
</tr>
<tr>
<td>30.</td>
<td>It crashed two times and I had to restart the Project.</td>
</tr>
<tr>
<td>31.</td>
<td>-</td>
</tr>
<tr>
<td>32.</td>
<td>No</td>
</tr>
<tr>
<td>33.</td>
<td>program crashes</td>
</tr>
<tr>
<td>34.</td>
<td>no. text &amp; icon overlay on camera info page.</td>
</tr>
<tr>
<td>35.</td>
<td>No</td>
</tr>
<tr>
<td>36.</td>
<td>No, or not yet :)</td>
</tr>
</tbody>
</table>
Table A.31: Answers to Question No. 18: What features do you think should be added to the toolbox?

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Setting up angle and field of views by turning the camera as an object. Doing that nowadays means putting numbers into the settings panel and that’s not very fast.</td>
</tr>
<tr>
<td>2.</td>
<td>more cameras with all the possible lens/IR capabilities</td>
</tr>
<tr>
<td>3.</td>
<td>Distance and pixel calculator features.</td>
</tr>
<tr>
<td>4.</td>
<td>No comment</td>
</tr>
<tr>
<td>5.</td>
<td>- corridor format. (rotate all cams FOV to 90) - Aspect ratios (4:3, 16:9, 9:16) - Camera accessories, for example, mounting brackets, Illuminators (with IR, White light Field of View)</td>
</tr>
<tr>
<td>6.</td>
<td>Easier rotation/manipulation of the map with key-strokes (takes longer with the mouse) Possibility to change the lens of the cameras to non standard lenses from the Axis catalogue (add data by the optical team when they test with new lenses)</td>
</tr>
<tr>
<td>7.</td>
<td>minimum resolution at the far end of the “cone”</td>
</tr>
<tr>
<td>8.</td>
<td>Integration with a light simulation system, so that daytime dependent light analysis could be done. (Maybe not possible, but could perhaps be nice?) Maybe also export the model to some rendering software, so that scenes could be created for purpose of testing, analysis and commercials?</td>
</tr>
<tr>
<td>9.</td>
<td>Non</td>
</tr>
<tr>
<td>10.</td>
<td>Standard accessories, such as ceiling mount, pole mount, etc- geometries that you can place on walls or ceilings for camera placement?</td>
</tr>
<tr>
<td>11.</td>
<td>Listed above in 16. If its interesting I listed there - contact me - I have more ideas around that.</td>
</tr>
<tr>
<td>12.</td>
<td>not sure right now.</td>
</tr>
<tr>
<td>13.</td>
<td>-</td>
</tr>
<tr>
<td>14.</td>
<td>see 16.</td>
</tr>
<tr>
<td>15.</td>
<td>A start wizard</td>
</tr>
<tr>
<td>16.</td>
<td>Improve usability and ease of use.</td>
</tr>
<tr>
<td>17.</td>
<td>-</td>
</tr>
<tr>
<td>18.</td>
<td>all I noted during seminar, but it was said they are already under construction</td>
</tr>
<tr>
<td>19.</td>
<td>“grab and direct” functionality would be the most appreciated thing.</td>
</tr>
<tr>
<td>20.</td>
<td>Na</td>
</tr>
<tr>
<td>21.</td>
<td>Slide bars b</td>
</tr>
<tr>
<td>22.</td>
<td>Ability to drag camera view. Ability to take obstructions such as walls into consideration when looking at the field of view.</td>
</tr>
<tr>
<td>23.</td>
<td>video samples</td>
</tr>
<tr>
<td>24.</td>
<td>The final positioning camera’s view from the camera eye mode.</td>
</tr>
<tr>
<td>25.</td>
<td>Needs more time to explore the tool and give feedback</td>
</tr>
<tr>
<td>26.</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.31 – Continued from previous page

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>dynamic drag/position of the field of view cone for the cameras - then update the information window</td>
</tr>
<tr>
<td>28</td>
<td>Be able to make all the changes directly on the graphics.</td>
</tr>
<tr>
<td>29</td>
<td>I need to work with the program for a while in order to get familiarity</td>
</tr>
<tr>
<td>30</td>
<td>Editable Lens parameter, sometime need to use 3 party lens with different mm.</td>
</tr>
<tr>
<td>31</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>Unsure</td>
</tr>
<tr>
<td>33</td>
<td>mouse rotation to adjust FOV of mounted camera, instead of entry values.</td>
</tr>
<tr>
<td>34</td>
<td>can we change the camera’s color by selection</td>
</tr>
<tr>
<td>35</td>
<td>shortcuts to be more rapid</td>
</tr>
</tbody>
</table>

Table A.32: Answers to Question No. 19: How probable is it that you would use SketchUp for your next system design?

<table>
<thead>
<tr>
<th>Response</th>
<th>Percentage</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not likely at all</td>
<td>2%</td>
<td>1</td>
</tr>
<tr>
<td>Slightly probable</td>
<td>10%</td>
<td>6</td>
</tr>
<tr>
<td>Moderately probable</td>
<td>14%</td>
<td>8</td>
</tr>
<tr>
<td>Very likely</td>
<td>41%</td>
<td>24</td>
</tr>
<tr>
<td>Extremely likely</td>
<td>21%</td>
<td>12</td>
</tr>
<tr>
<td>NA</td>
<td>5%</td>
<td>3</td>
</tr>
<tr>
<td>Comment please...</td>
<td>7%</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total Responses</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table A.33: Comments to Question No. 19

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>will need to have the full product line and accessories to be practical for the projects I work on.</td>
</tr>
<tr>
<td>2</td>
<td>need more experience and toolbox need to be as requested in answer 20 below</td>
</tr>
<tr>
<td>3</td>
<td>I do not perform system design in my current position</td>
</tr>
<tr>
<td>4</td>
<td>I do not perform system designs as part of my job</td>
</tr>
</tbody>
</table>
### Table A.34: Answers to Question No. 20: What is your suggestion for overall improvement of the toolbox in order to suit your needs?

<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>don’t know, what to do a sketchup of a real environment and validate IRL. Great looking tool! I want to have it for use as soon as possible and also use it in our education towards our customers.</td>
</tr>
<tr>
<td>2.</td>
<td>I don’t understand the question.</td>
</tr>
<tr>
<td>3.</td>
<td>I thought it was very easy to use, especially as someone that only as limited experience with using system design tools. I don’t have any suggestions at this time.</td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Localization for the toolbox if possible.</td>
</tr>
<tr>
<td>6.</td>
<td>Need Thermal range based on Johnson’s criteria</td>
</tr>
<tr>
<td>7.</td>
<td>if you can have a button that toggles all the view cones on or off on the page.</td>
</tr>
<tr>
<td>8.</td>
<td>It is very easy to make it truly like in reality - info from google, images and maps becoming plan of the object in minutes.</td>
</tr>
<tr>
<td>9.</td>
<td>local language is necessary.</td>
</tr>
<tr>
<td>10.</td>
<td>-</td>
</tr>
<tr>
<td>11.</td>
<td>More models</td>
</tr>
<tr>
<td>12.</td>
<td>as I want to use it for Training purpose probably you can provide as well some sample designs which I can already use and redesign it as exercise.</td>
</tr>
<tr>
<td>13.</td>
<td>Workaround in the intuitive way of the tool</td>
</tr>
<tr>
<td>14.</td>
<td>None at this time.</td>
</tr>
<tr>
<td>15.</td>
<td>I think it will be a nice tool if angle of view is more easy to setup (with the mouse) and show the quality in pixel on a certain distance. of course the complete AXIS offering should be available including the option to change the lens. This might also be a custom lens with focalpoint= xxx mm.</td>
</tr>
<tr>
<td>16.</td>
<td>keep doing great job!</td>
</tr>
<tr>
<td>17.</td>
<td>multilingual GUI</td>
</tr>
<tr>
<td>18.</td>
<td>Na</td>
</tr>
<tr>
<td>20.</td>
<td>n/a</td>
</tr>
<tr>
<td>21.</td>
<td>n/a</td>
</tr>
<tr>
<td>22.</td>
<td>easier to change the angle of view</td>
</tr>
<tr>
<td>23.</td>
<td>If there are some guide on the feature, will be prefect.</td>
</tr>
<tr>
<td>24.</td>
<td>The final positioning camera’s view from the camera eye mode.</td>
</tr>
<tr>
<td>25.</td>
<td>1. Needs offline versions of the tools. 2. Needs userguides/how to videos on using, lens calculator, product selector, revit, sketch up etc.</td>
</tr>
<tr>
<td>26.</td>
<td>No suggestion from me.</td>
</tr>
<tr>
<td>27.</td>
<td>I haven’t used the toolbox enough to make any suggestions.</td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>#</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>The tool work very good, so just update the models and I think is good to go. Good work!!</td>
</tr>
<tr>
<td>29</td>
<td>Improve camera placement and rotation configuration</td>
</tr>
<tr>
<td>30</td>
<td>.</td>
</tr>
<tr>
<td>31</td>
<td>Very good tool - would like to see better interaction with the field of views (to adjust the vari-focal lens) etc</td>
</tr>
<tr>
<td>32</td>
<td>More intuitive UI design.</td>
</tr>
<tr>
<td>33</td>
<td>Make the camera specific options easier to find.</td>
</tr>
<tr>
<td>34</td>
<td>I have seen SketchUp only during training session: I need to work with the program for a while in order to know it more in detail</td>
</tr>
<tr>
<td>35</td>
<td>no</td>
</tr>
<tr>
<td>36</td>
<td>I need more practise and experience on the tool before answering this question. So far, I only try that for 30mins in the workshop. Lot of function need to be discovered.</td>
</tr>
<tr>
<td>37</td>
<td>I didn’t have enough time to tested it properly, but it works fine in general.</td>
</tr>
<tr>
<td>38</td>
<td>-</td>
</tr>
<tr>
<td>39</td>
<td>Having Back button when you did something wrong, there is a Back button to revert back previous</td>
</tr>
<tr>
<td>40</td>
<td>No info about the toolbox. It would have been good to have more time for the tasks, whilst supervised.</td>
</tr>
<tr>
<td>41</td>
<td>Please translate the tool into Japanese.</td>
</tr>
<tr>
<td>42</td>
<td>We should have pixel and angle of view details when you select the camera and focus area</td>
</tr>
<tr>
<td>43</td>
<td>None</td>
</tr>
<tr>
<td>44</td>
<td>new camera support update to follow release time .</td>
</tr>
<tr>
<td>45</td>
<td>Would fine if the time duration would be extended more. Since the tool is new for most of us, so to understand the functioning tool and get familiar.</td>
</tr>
<tr>
<td>46</td>
<td>in general it’s a powerfull tool, to improve the feature we need to use it day by day for give a real feedback</td>
</tr>
<tr>
<td>47</td>
<td>It can replace visio in Fundamental training, however need a clear step by step document for exercise.</td>
</tr>
<tr>
<td>48</td>
<td>More cameras, easier to point/aim/adjust</td>
</tr>
</tbody>
</table>