Documentation

Michael Bellem, Jan-Philipp Follenius, Oliver Röwer, Miao Wang

Contents

1 Introduction **3**

2 Installation **4**
2.1 Windows XP . . . 4
2.2 Windows Mobile 5.0 . . . 4
2.3 Additional Setups Steps Required for all Platforms . . . 4

3 Architecture **5**
3.1 Overview . . . 5
3.2 Image Grabber . . . 6
3.3 Pose Estimation . . . 6
3.4 Layout Manager . . . 7
3.5 Workflow and Device Engine . . . 8
3.6 Rendering Engine . . . 10

4 Stabilizing Pose Estimation Data **11**
4.1 Description of the Problem . . . 11
4.2 Averaging . . . 12
4.3 Eliminating Outliers Using Euclidean Distance . . . 13
4.4 Eliminating Outliers Using Mahalanobis Distance . . . 13
4.5 Evaluation and Conclusion . . . 14

5 Format Descriptions **15**
5.1 Workflow Definition File . . . 15
5.2 Device Definition File . . . 18
5.3 Marker Definition File . . . 19

6 Obtaining CAD Data for Components **20**

7 External Libraries **20**
7.1 ARToolKit . . . 20
7.2 ARToolKitPlus . . . 20
7.3 TinyXML . . . 20

8 Summary and Conclusion **21**

A Document Type Definitions **22**

B Work Environment **24**
B.1 Hardware . . . 24
B.2 Software . . . 24

1 Introduction

The goal of this document is to outline some of the details of our implementation. This should enable other persons to reuse parts of our implementation by getting a deeper understanding of how the different parts work together and of what the purpose of several classes is. This document will not cover all implementation details. For detailed interface descriptions, inheritance diagrams and method signatures, please use the doxygen documentation provided together with this documentation.

In the first section the installation of `SMART` is described.

The second section describes the architecture of `SMART` and outlines the important parts that are then described in more depth. A closer look is taken at the algorithms and the concepts used througout the implementation.

A third section has a closer look at the instability of pose estimation data, which was one big problem we had to adress during the development of `SMART`, and presents some of our approaches to solve this problem.

`SMART` uses several XML file formats which are described in the fourth section. If you want to write workflows, machine configuration files or marker files please adress this section as it will give you all the information you need to do so.

A sixth section describes how CAD data for components can be obtained and how they can be integrated into `SMART`.

Another section outlines all external libraries that we used and gives some links for further information on those libraries.

2 Installation

2.1 Windows XP

`SMART` comes together with a setup file called *setup.exe*. Simply execute this file and follow the installer's instructions. Use the start menu entry or the *SMART.exe* file to start the program.

Apart from that, a webcam has to be installed and ready to use.

2.2 Windows Mobile 5.0

Copy the *SMART.cab* file to the pocket PC and execute it. `SMART` will then be installed and can be found in the start menu for future use.

Additionaly, the drivers for the SDIO cam have to be installed. Use the proprietary drivers to do that.

2.3 Additional Setups Steps Required for all Platforms

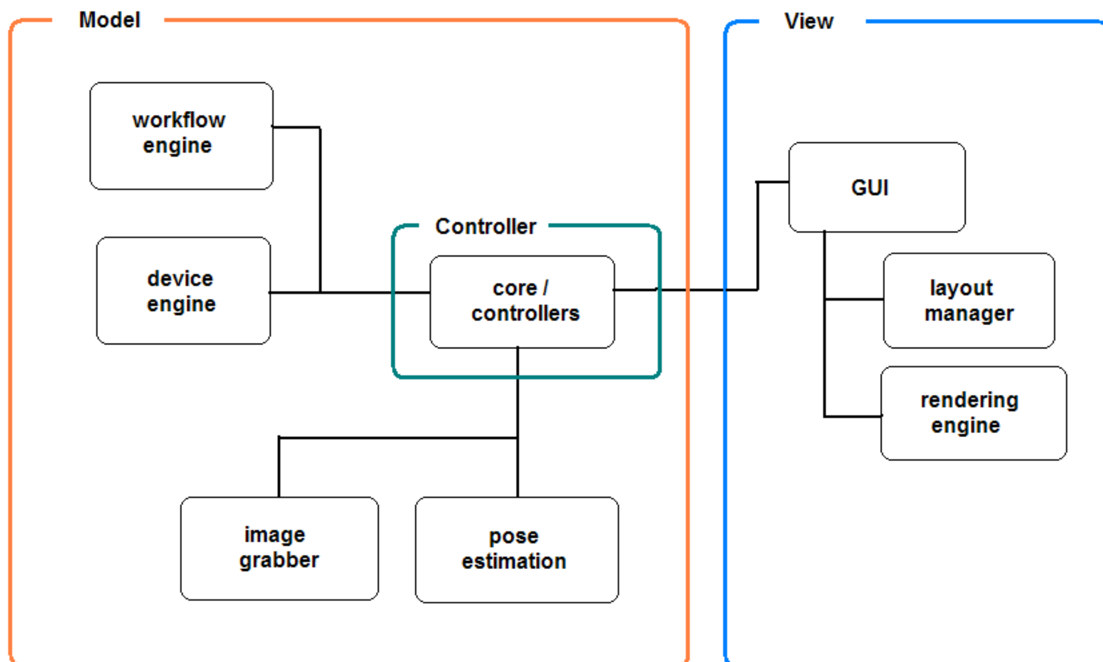
After installing `SMART` there are a view things to be done. Edit the file *settings.xml* which is located in the data directory of `SMART` installation folder. You need to provide `SMART` with a valid workflow defintion file, a device definition file and a marker definition file. Refer to the corresponding sections within this document to see how you can create these files.

3 Architecture

This section describes the architecture of SMART.

3.1 Overview

The following diagram shows the important parts SMART consists of and how they work together.



As the graphic shows, the architecture follows the model-view-controller pattern. We tried to make sure that the GUI and the model are separated as much as possible and that therefore the GUI can be exchanged easily. This is very important because SMART was designed to run on a variety of platforms including laptops and pocket PCs, which have completely different display constraints. Besides exchanging the whole GUI we have invented a layout manager that arranges all GUI elements intellegently on the screen. By inheriting subclasses of this layout manager the display behaviour can easily be adopted to new platforms and new requirements. For further information on layout management take a look at the related subsection later in this section.

The rendering engine is the part of the GUI that takes care of the actual rendering. The engine uses OpenGL for efficient rendering of GUI elements as well as 3d models.

The core and controllers part controls the program flow of SMART and reacts to user input captured within the GUI.

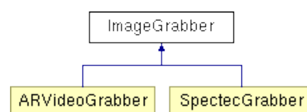
While the image grabber is responsible for grabbing live images from a camera, the pose estimation is the critical part of `SMART` that actually detects the markers in the images obtained from the camera and estimates the position of the mainframe in space. This information is then used by the rendering engine to display 3d models of the relevant components at the right position and with the correct perspective. Again, we refer to the related subsection for further information on this topic.

The workflow engine handles workflows that describe how repair tasks should be illustrated with `SMART`. These workflows can be loaded from XML files (see chapter *Format Descriptions*).

The device engine holds the current machine configuration, i.e. which devices are inside the machine and the relative positions of these devices, and can be loaded from an XML file as well.

3.2 Image Grabber

The image grabber is responsible for grabbing live images from a camera. Currently, there are two subclasses inheriting from the abstract class *ImageGrabber*.

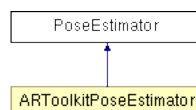


The *ARVideoGrabber* was written for dealing with webcams on laptop platforms and uses - as the name implies - the ARToolKit (see chapter *External Libraries*) to grab images.

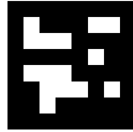
The *SpectecGrabber* is the image grabber for the pocket PC platform. It uses an SDK delivered by Spectec to adress SDIO cameras.

3.3 Pose Estimation

Although we provide an abstract class *PoseEstimator* and therefore allow several implementations, we have only implemented one pose estimator - the *ARToolkitPoseEstimator*. It uses the ARToolKitPlus library as outlined in the chapter *External Libraries*. This library uses black-and-white markers and calculates their orientation in space.



The pose estimator takes an image and looks for markers in the image. If there are visible markers, their ID can be extracted using the two-dimensional barcode on the surface of the markers. The pose estimator is provided with the perspective mapping properties of the camera and can therefore extract the orientation of the marker in space described by a 4×4 transformation matrix.



Since the pose estimator knows the relative position of each marker to the root device (a mainframe in our case) it can then combine the orientational information of several markers to calculate the position of the root device itself.

The result of the pose estimator is a final 4×4 transformation matrix that can just be passed to OpenGL by the rendering engine to change to the root device's local coordinate system.

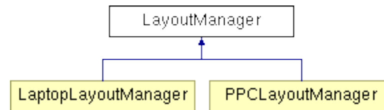
3.4 Layout Manager

The idea behind the layout manager is to simply specify the elements that should illustrate a workstep and letting the layout manager arrange these elements autonomously.

For example, there could be an arrow pointing toward device x, a text box associated with device y and an animation applied to device z. The layout manager will then take all these elements and arrange them on the screen so that they are easy to see for the user. Moreover, the workflow writer can consider some devices as *forced visible* and the layout manager will make sure that these devices will not be covered by other

elements. Ideally, a layout manager should use free space on the screen efficiently and should avoid to rearrange elements too often.

We have created two layout managers, one for the laptop platform and one for the pocket PC platform. The *LaptopLayoutManager* is capable of arranging arrows, circles and text boxes dynamically while the *PPCLayoutManager* always places the text boxes at the lower part of the display and does not show arrows and circles due to the limited amount of space on pocket PC platforms.

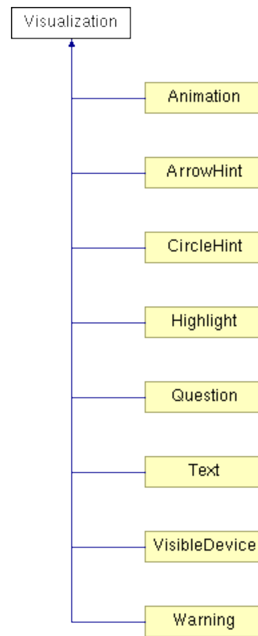


To be able to avoid that some devices are covered by visualization elements, the layout manager has to calculate the screen areas covered by a three-dimensional device. Therefore, it has to reproduce the effect of the OpenGL transformation pipeline. The exploitation of this information is a powerful instrument and a new approach to user interface layout management especially suited for augmented reality applications, where we have to deal with the combination of two-dimensional and three-dimensional objects.

3.5 Workflow and Device Engine

In this section, we will have a closer look at how workflows and devices are represented within `SMART`. A workflow is simply a sequence of worksteps. There are two types of worksteps: normal worksteps and conditional worksteps. Normal worksteps simply proceed to the next workstep in the sequence after they have been completed (currently indicated by a click on the next button). In contrast, conditional worksteps ask the user a question, which he has to answer with yes or no. Depending on the user's answer, a workstep is selected as the next workstep. Therefore, all worksteps have a unique ID associated with them.

As we have already seen in the previous section about layout management, a workstep is a list of visualizations - i.e. elements that illustrate the workstep. Visualizations are easily extendable and we have already implemented a handfull of them.



Later on, the layout manager will take the list of visualizations connected to the current workstep and will create and position corresponding rendering objects that are drawn on the screen during rendering.

Some of these visualizations are straightforward while others require some more explanations. The *VisibleDevice* visualization is not really something observable for the user but a hint to the layout manager which devices should always be visible. The *Highlight* visualization points out certain devices by simply displaying their associated 3d model. An *Animation* has the same effect as a *Highlight* but additionally applies an animation to the device.

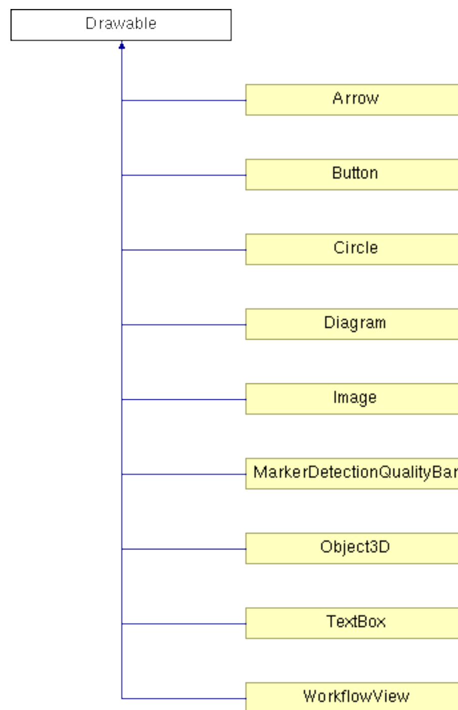
The *WorkflowLoader* is capable of loading a workflow from an XML file. For further information on XML file formats, please refer to section *Format Descriptions*.

The devices in `SMART` are held in a recursive data structure. There is a root device which is the mainframe in our case. Every device can have an arbitrary number of child devices and each device has a unique name using a syntax comparable to the XML XPath syntax. To realize the *Highlight* and *Animation* visualizations, every device has a 3d model associated with it.

Similar to workflows, a device structure (in our case simply called *machine configuration*) can be read from an XML file using the *DeviceLoader*.

3.6 Rendering Engine

The heart of SMART's rendering engine is the *Drawable* data structure. A drawable is any kind of object that is able to render itself. We allow for recursively built drawables using a composite design pattern. Examples for drawables in SMART are *Object3D* - a drawable that is able to render 3d models, *Arrow* and *TextBox* but also user interface elements like *Button* and composite drawables like the *WorkflowView*.



The *GUI* class contains a list of drawables that are updated every frame. In our case the GUI contains only the *WorkflowView* which serves as a container for all other drawables. Each drawable is then responsible for drawing itself and its child drawables.

Besides rendering themselves, drawables receive events when clicked and can react to those clicks by triggering actions through the responsible controller. The reason why we put this mouse click handling inside the drawables is that only the drawables know exactly how they draw themselves and can therefore make more appropriate decisions on wether they want to handle a click or not.

4 Stabilizing Pose Estimation Data

4.1 Description of the Problem

One big problem we had to face during the development of `SMART` was the stability of the pose estimation results. As stated earlier in this document, we use an external library called *ARToolKitPlus* for pose estimation and are then supplied with a transformation matrix describing the orientation of our root device in space. Although there is a lot of stuff implemented inside the toolkit itself (using the recommended *robust planar pose estimation* was too slow for our purpose), the pose estimation is very unstable and rendered devices bounce too much to be helpful for a service engineer.

In our opinion, the unstable pose estimation is a result of the partially difficult circumstances we had to face for our application. Markers are not detected constantly but vanish from time to time and are not always detected properly which seems to be due to the following reasons:

- small marker sizes due to very limited free space for the markers on a mainframe
- changing lighting characteristics (changing daylight, shadows, AC lights)
- auto-adjusting features of the camera that cannot be turned off
- sometimes bad camera quality (especially on the pocket PC platform)
- blurred camera images during movement
- numerical instabilities (perhaps due to fixed point implementations of parts of the toolkit)
- dynamic parts of the algorithms used within the toolkit (auto thresholding for example)

Since time and resource constraints did not allow us to change the hardware premises nor the image processing parts of the toolkit, we had to make the most of the results we got from the toolkit. We tried some different approaches to stabilize pose estimation results, which will be presented in the next sections.

Let (t_1, t_2, t_3) be a transformation in $\mathbb{R}^3$ and let (α, β, γ) describe a rotation around the x-, y- and z-axis throughout the rest of thist chapter.

The resulting transformation matrix then looks like this:

$$T = \begin{pmatrix} \cos\beta * \cos\gamma & \cos\beta * \sin\gamma & -\sin\beta & t_1 \\ -\cos\alpha * \sin\gamma + \sin\alpha * \sin\beta * \cos\gamma & \cos\alpha * \cos\gamma + \sin\alpha * \sin\beta * \sin\gamma & \sin\alpha * \cos\gamma & t_2 \\ \sin\alpha * \sin\gamma + \cos\alpha * \sin\beta * \cos\gamma & -\sin\alpha * \cos\gamma + \cos\alpha * \sin\beta * \sin\gamma & \cos\alpha * \cos\beta & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Since we do not want to reconstruct the angles from the transformation matrix, we simply take the 9 values
$T(0,0), T(0,1), T(0,2), T(1,0), T(1,1), T(1,2), T(2,0), T(2,1), T(2,2)$ as the rotational parameters. Thus, the vector

$$t = \begin{pmatrix} T(0,0) \\ T(1,0) \\ T(2,0) \\ T(0,1) \\ T(1,1) \\ T(2,1) \\ T(0,2) \\ T(1,2) \\ T(2,2) \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} \in \mathbb{R}^{12}$$

describes a transformation. Unfortunately, the entries of this vector are not stochastically indepent, so there is no isomorphism from the space of all transformation matrices to the $\mathbb{R}^{12}$, which makes the application of stochastical methods a bit questionable.

Nevertheless, we tried to apply some basic interpolation and filtering methods to these twelve-dimensional vectors.

4.2 Averaging

The first idea we came up with is to simply average transformation matrices over time. This smooths the bouncing of the orientation result and compensates for smaller errors. Let n be the number of matrices that should be used for averaging and let $(t_0, t_1, ..., t_{n-1})$ be the transformation vectors at times $0, 1, .., n-1$. The averaged vector $\bar{t}$ is then simply computed as

$$\bar{t} = \begin{pmatrix} \sum_{i=0}^{n} t_i(0) \\ \sum_{i=0}^{n} t_i(1) \\ ... \\ \sum_{i=0}^{n} t_i(n-2) \\ \sum_{i=0}^{n} t_i(n-1) \end{pmatrix}.$$

The application of this technique greatly improves the stability of the results. Apparently, the choice of the maximum number of matrices used for averaging n is vital because otherwise the rendered devices will float around and will react much too

slowly to changes of the camera angle. Smaller errors that occur while the camera remains stable can be culled using this technique.

Unfortunately, sometimes completely wrong orientation results occur that ruin the averaging process. Because of this, we came up with the idea of eliminating outliers before averaging. This approach is described in the next two sections.

4.3 Eliminating Outliers Using Euclidean Distance

As we want to eliminate outliers, we need some kind of metrics for our twelve-dimensional space. In this section, we will use the euclidean metrics. Thus the distance between two transformation vectors t_1 and t_2 is defined as

$$d(t_1, t_2) = \sqrt{\begin{pmatrix} t_1(0) \\ t_1(1) \\ ... \\ t_1(n-2) \\ t_1(n-1) \end{pmatrix} \cdot \begin{pmatrix} t_2(0) \\ t_2(1) \\ ... \\ t_2(n-2) \\ t_2(n-1) \end{pmatrix}}$$

Using a sliding-window technique and simple update forumulas we keep track of the expectation value $E(t)$ and the variance $V(t)$ of the sampled transformation vectors. A transformation vector t' is then considered an outlier if

$$d(t', E(t)) > V(t) \cdot s$$

where s is a constant factor that determines the threshold for the outlier detection.

4.4 Eliminating Outliers Using Mahalanobis Distance

Since the outlier detection using the euclidean distance did not work that well for us, we came up with the idea of involving statistical information into our metrics. Using the euclidean metrics together with a threshold implies that the transformation vectors are centered around the expectation value, which is not necessarily the case. In the case of a non-spherical distribution of the transformation vectors around the expectation value, the Mahalanobis distance provides us with a more accurate distance measurement.

Let P be the estimated covariance matrix of the transformation vectors $(t_0, t_1, ..., t_{n-1})$

$$P = \frac{1}{n} \sum_{i=0}^{n-1} (t_i - E(t))(t_i - E(t))^T.$$

The distance between two transformation vectors t_1 and t_2 is then computed as

$$d_m(t_1, t_2) = \sqrt{(t_1 - t_2)^T P^{-1} (t_1 - t_2)}.$$

Thus, we would consider a transformation vector t' as an outlier if

$$d_m(t', E(t)) > s_m$$

where s_m is a threshold for the outlier detection.

4.5 Evaluation and Conclusion

Unfortunately, we have not been able to further improve the detection stability beyond the improvements due to simple averaging. For some reasons, the outlier elimination worked not as well as we would have expected it. Most of the wrong results were not considered as outliers because they've been returned for a longer period of time. There are grounds for the assumption that stability can not be further improved without actually diving into the image processing code or without changing the outer circumstances.

One problem was, that we don't know anything about the stochastical characteristics of the transformation matrices returned. More analysis on that could possibly bring up new techniques to stabilize pose estimation data.

5 Format Descriptions

This chapter provides descriptions of the three XML file formats used with `SMART`.

5.1 Workflow Definition File

A workflow definition file (simply called WDF throughout the rest of this chapter) describes the worksteps a workflow consists of and specifies how these worksteps should be illustrated for the user. The rest of this section will describe the elements of the WDF together with their attributes and possible child elements. The DTD will close this section.

workflow the root element of every WDF

Attributes:

- **id** a unique ID associated with this workflow
- **name** a human readable name for this workflow that can e.g. be used for menu entries

Child Elements:

- **desc** a description of this workflow that can e.g. be used for menu entries, tooltips
  Child Elements:
  - **de** german description
  - **en** english description
- **workstep** a workstep of this workflow
  Attributes:
  - **id** a unique ID associated with this workstep. This is used to refer to other worksteps, especially for conditional worksteps
  - **type** the type of this workstep. This can either be **step** or **conditional**. See the chapter about the workflow engine for any details.
  - **target** the root device for this workstep. Note, that root devices can change e.g. when a book has been removed and further actions for the book are necessary.

  Child Elements:
  - **desc** a description for this workstep. This should be short and incisive.
    Child Elements:
    - **de** german description
    - **en** english description
  - **text** displays a text box with additional information for the user
    Attributes:
    - **target** the ID of the device this text box refers to. This is optional and can be left out.

    Child Elements:

de german text. The text can contain line end characters \n and can mark important sections by surrounding them with \a tags. They will then be highlighted in the text box.

en english text. The text can contain line end characters \n and can mark important sections by surrounding them with \a tags. They will then be highlighted in the text box.

warning displays a warning box

Attributes:

target the ID of the device this warning box refers to. This is optional and can be left out.

Child Elements:

de german warning text. The text can contain line end characters \n and can mark important sections by embracing them with \a tags. They will then be highlighted in the warning box.

en english warning text. The text can contain line end characters \n and can mark important sections by embracing them with \a tags. They will then be highlighted in the warning box.

question displays a question box

Attributes:

target the ID of the device this question box refers to. This is optional and can be left out.

Child Elements:

de german question text. The text should be formulated as a question that can be answered with yes or no. The text can contain line end characters \n and can mark important sections by embracing them with \a tags. They will then be highlighted in the question box.

en english question text. The text should be formulated as a question that can be answered with yes or no. The text can contain line end characters \n and can mark important sections by embracing them with \a tags. They will then be highlighted in the question box.

nextWorkstep the ID of the workstep that will be executed after this workstep. This element only makes sense for normal worksteps. For conditional worksteps use the **yes** and **no** elements.

yes the ID of the workstep that will be executed after this workstep if the user answers with yes. This element only makes sense for conditional worksteps. For normal worksteps use the **nextWorkstep** element.

no the ID of the workstep that will be executed after this workstep if the user answers with no. This element only makes sense for conditional worksteps. For normal worksteps use the **nextWorkstep** element.

highlight the ID of the device that should be highlighted by displaying its 3d model

forceVisible the ID of a device that should always be visible. This is a hint to the layout manager on where to place the graphical user interface elements.

animation apply an animation to a device. This automatically triggers that the device will be highlighted. Don't use **highlight** and **animation** together.

Attributes:

- **target** the ID of the device that should be animated
- **repeat** this can be *true* or *false*. If true, the animation will repeat endlessly. If not, the animation will play only once.

Child Elements:

- **step** an animation step *Attributes:*
  - **pointX** x-coordinate of the center of rotation relative to the device's center.
  - **pointY** y-coordinate of the center of rotation relative to the device's center.
  - **pointZ** z-coordinate of the center of rotation relative to the device's center.
  - **moveX** x-coordinate of the translation vector
  - **moveY** y-coordinate of the translation vector
  - **moveZ** z-coordinate of the translation vector
  - **rotX** rotation angle around the x-axis
  - **rotY** rotation angle around the y-axis
  - **rotZ** rotation angle around the z-axis
  - **t** the time in which to perform the movement specified through translation and rotation. The time should be given in milliseconds.

circle the ID of the device that should be highlighted by drawing a circle around it

Attributes:

- **color** the color of the circle in hexadecimal RGB representation. For example, red would be 0xFF0000 and white would be 0xFFFFFF.
- **lineWidth** the width of the circle's border in pixels

arrow the ID of the device that should be highlighted by drawing an arrow pointing toward the device

Attributes:

- **color** the color of the arrow in hexadecimal RGB representation. For example, red would be 0xFF0000 and white would be 0xFFFFFF.

5.2 Device Definition File

A device definition file (from now on simply called DDF) describes a machine configuration. As outlined in a previous section, devices are held in a recursive data structure. The structure of the XML file reflects this recursive structure. Each device element can have an arbitrary number of child devices as child elements. Every device has a relative position to its parent device associated with it. This structure is very general and can be used for a wide range of scenarios, where augmented reality can be useful. In our case, the root device is the mainframe and its subdevices are the components within the mainframe. Every component can have further child devices of course, e.g. a book can have a chip module and a lot of memory modules as subdevices.

The rest of this section will describe the elements of the DDF together with their attributes and possible child elements. Again, the DTD will close this section.

device the root device. This is the element of every DDF. Note, that there can be only one root device.

Attributes:

- **id** a unique ID associated with this device
- **name** a human readable name for this device that can e.g. be used for menu entries
- **group** devices can be gathered in groups. Simply use equal group names to put two devices into the same group. For example, all memory modules of a book can be combined to one group.

Child Elements:

- **device** a subdevice of this device. This can be continued recursively.
- **relPose** the relative position of this device to its parent device
  Attributes:
  - **relX** x-coordinate of the translation vector
  - **relY** y-coordinate of the translation vector
  - **relZ** z-coordinate of the translation vector
  - **rotX** rotation around the x-axis
  - **rotY** rotation around the y-axis
  - **rotZ** rotation around the z-axis
- **model** the file name of the 3d model for this device. Currently, this can be either a md2 file or a stl file
- **texture** the file name of the texture applied to the 3d model. Currently, this should be a 24 bit bmp file. This is optional and if this is left out, the model is simply rendered with a solid color.
- **color** the color to be used if no texture file name is given. The color has to be specified in hexadecimal RGB representation. For example, red would be 0xFF0000 and white would be 0xFFFFFF.

5.3 Marker Definition File

The marker definition file (from now on simply called MDF) specifies the position of all markers relative to the root device. This information is necessary for the pose estimation part of `SMART` to be able to reconstruct the orientation of the root device in space. Along with the position, further information about the markers is provided, such as the size and the ID of the markers.

The rest of this section will describe the elements of the MDF together with their attributes and possible child elements. Again, the DTD will close this section.

markers the root element of every MDF
- *Child Elements:*
  - **header** general information about all markers in this file
    - *Child Elements:*
      - **borderRatio** the percentage of the border part of the marker given as a number between 0 and 1.
  - **marker** specification of one marker
    - *Attributes:*
      - **id** the ID of the marker. This is necessary to associate the barcode with the correct positional information.
      - **width** the width of the marker in mm
    - *Child Elements:*
      - **relPos** the position of this marker relative to the root device
        - *Attributes:*
          - **relX** x-coordinate of the translation vector
          - **relY** y-coordinate of the translation vector
          - **relZ** z-coordinate of the translation vector
      - **relRotAngle** the rotation of this marker relative to the root device
        - *Attributes:*
          - **rotX** rotation around the x-axis
          - **rotY** rotation around the y-axis
          - **rotZ** rotation around the z-axis

6 Obtaining CAD Data for Components

3D models for important components are a relevant part of `SMART`. CAD data for all components are already available and are stored in a software sytem called *CATIA*. Data can be exported to an STL file format, which describes the geometry of the component. `SMART` includes an STL parser that is able to load 3d models from STL files. Therefore it is possible to use from CATIA extracted CAD data in `SMART` without any necessary conversions. During the export process the sample distance can be specified, which allows control of the wanted level of detail.

7 External Libraries

`SMART` uses several external libraries that will be presented in this chapter.

7.1 ARToolKit

ARToolKit is a software library for building augmented reality applications. It has been developed by Dr. Hirokazu Kato, the University of Washington, the University of Canterbury and ARToolworks, Inc. `SMART` uses only a small part of this toolkit, namely the image grabbing part which provided us with an easy way to grab live images from a webcam. We used version 2.72.1 of the toolkit.

For further information on ARToolKit, please see the website
`http://www.hitl.washington.edu/artoolkit/`

7.2 ARToolKitPlus

The ARToolKitPlus is an extended version of ARToolKit that adds a lot of features and offers a class-based API. It has been developed by the Graz University of Technology. `SMART` uses the toolkit for the whole pose estimation part, so this library is a substantial part of `SMART`. We used version 2.1.1 of the toolkit. For further information about this great toolkit, please see the website
`http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php`

7.3 TinyXML

We use TinyXML for doing all the XML parsing in `SMART`. TinyXML offers an easy-to-use and very small C++ XML parser that saved us a lot of work. We used version 2.5.3 of TinyXML. Please see the website for further information:
`http://sourceforge.net/projects/tinyxml`

8 Summary and Conclusion

We have developed a prototype improving maintenance procedures for mainframes through the use of augmented reality. We rely on black-and-white markers and an external library for the substantial task of pose estimation. Live camera images are combined with additional information in the form of three-dimensional models and two-dimensional graphical user interface elements. Our solution allows for different platforms especially laptop and pocket PC.

Our prototype and the concept of augmented reality can easily be adopted to other maintenance scenarios. It would be imaginable to support customers when upgrading their personal computers or when repairing their own car. There are a lot more areas where augmented reality comes into play, ranging from the gaming sector over navigation tools and tourism to military applications. With the evolution of better and better image processing algorithms and rendering capabilities augmented reality will surely become one of the topics of the next years.

The pose estimation is an area that surely needs further improvements. The stability that we achieved is not acceptable for use in a real product. In addition to that, it is obvious that using markers that have to be attached to a mainframe is not the perfect solution. The use of advanced image processing algorithms can possibly substitute the application of marker-based detection methods.

Advanced rendering techniques like cel shading could improve the display of relevant components and make animations even more helpful.

During the export of STL files from CAD data, the geometry is created by sampling the data in equidistant steps. This process can be further improved by using alogrithms that automatically reduce the number of faces by merging faces on planar parts of the component. The faces saved in this way can be used to increase details in more interesting areas. This can improve performance and make components look more realistic.

Since most maintenance tasks require both hands it is essential to offer a solution that lets the user have his hands free. The use of head-mounted displays is one possibility to achieve this. To really make this solution applicable, a suitable user interface is necessary. It would be thinkable, that the user can control the program via gesture recognition or speech recognition. Additional information in terms of speech can further improve the use of the program.

A Document Type Definitions

DTD for workflow definition files:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT workflow    (desc, (workstep)+) >
<!ATTLIST workflow
  id    NMTOKEN    #REQUIRED
  name  CDATA      #REQUIRED
  xmlns CDATA      #IMPLIED
>

<!ENTITY % languages "de, en" >

<!ELEMENT desc (%languages;) >
<!ELEMENT de     (#PCDATA)  >
<!ELEMENT en     (#PCDATA)  >

<!ELEMENT  workstep (desc, (highlight | forceVisible | arrow | circle | animation)* , (text | warning | question),  (nextWorkstep | (yes, no))) >
<!ATTLIST workstep
  id      NMTOKEN                #REQUIRED
  type    (step | conditional)   #REQUIRED
  target  CDATA                  #REQUIRED
>

<!ELEMENT text     (%languages;) >
<!ATTLIST text
  target  CDATA               #IMPLIED
>
<!ELEMENT warning     (%languages;) >
<!ATTLIST warning
  target  CDATA               #IMPLIED
>
<!ELEMENT question     (%languages;) >
<!ATTLIST question
  target  CDATA               #IMPLIED
>

<!ELEMENT nextWorkstep (#PCDATA) >
<!ELEMENT yes      (#PCDATA) >
<!ELEMENT no       (#PCDATA) >

<!ELEMENT highlight      (#PCDATA) >
<!ELEMENT forceVisible   (#PCDATA) >
<!ELEMENT arrow          (#PCDATA) >
<!ATTLIST arrow
  color CDATA           #REQUIRED
>
<!ELEMENT circle     (#PCDATA) >
<!ATTLIST circle
  color        CDATA         #REQUIRED
  lineWidth    CDATA         #REQUIRED
>

<!ELEMENT animation     (step)* >
<!ATTLIST animation
  target  CDATA               #REQUIRED
  repeat  (true|false)        #REQUIRED
>

<!ELEMENT step     EMPTY >
<!ATTLIST step
  pointX  CDATA               #REQUIRED
  pointY  CDATA               #REQUIRED
  pointZ  CDATA               #REQUIRED
  moveX   CDATA               #REQUIRED
  moveY   CDATA               #REQUIRED
  moveZ   CDATA               #REQUIRED
  rotX    CDATA               #REQUIRED
  rotY    CDATA               #REQUIRED
  rotZ    CDATA               #REQUIRED
  t       CDATA               #REQUIRED
>
```

DTD for device definition files:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT device  (relPose?, size?, model?, (texture | color)?, (device)*) >
<!ATTLIST device
  id  NMTOKEN             #REQUIRED
  name  CDATA     #REQUIRED
  group CDATA     #IMPLIED
  xmlns    CDATA     #IMPLIED
>

<!ELEMENT relPose        EMPTY >
<!ATTLIST relPose
  relX  CDATA     #REQUIRED
  relY  CDATA     #REQUIRED
  relZ  CDATA     #REQUIRED
  rotX  CDATA     #REQUIRED
  rotY  CDATA     #REQUIRED
  rotZ  CDATA     #REQUIRED
>

<!ELEMENT size        EMPTY >
<!ATTLIST size
  width CDATA     #REQUIRED
  height  CDATA     #REQUIRED
  depth CDATA     #REQUIRED
>

<!ELEMENT model        (#PCDATA) >
<!ELEMENT texture        (#PCDATA) >
<!ELEMENT color        (#PCDATA) >
```

DTD for marker definition files:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT markers    (header, (marker)*) >
<!ATTLIST markers
            xmlns    CDATA    #IMPLIED
>

<!ELEMENT header    (borderRatio) >
<!ELEMENT borderRatio    (#PCDATA) >

<!ELEMENT marker    (relPos, relRotAngle) >
<!ATTLIST marker
            id   NMTOKEN    #REQUIRED
            width    CDATA    #REQUIRED
>

<!ELEMENT relPos    EMPTY >
<!ATTLIST relPos
            relX    CDATA    #REQUIRED
            relY    CDATA    #REQUIRED
            relZ    CDATA    #REQUIRED
>

<!ELEMENT relRotAngle    EMPTY >
<!ATTLIST relRotAngle
            rotX    CDATA    #REQUIRED
            rotY    CDATA    #REQUIRED
            rotZ    CDATA    #REQUIRED
>
```

B Work Environment

This appendix summarizes the hardware and software environment we used during the development of `SMART`.

B.1 Hardware

We used T60p and T41 Thinkpads for development on the laptop platform. We used a Logitech QuickCam and a Logitech UltraVision as web cameras.

For development on the pocket PC platform we chose the Dell Axim X51v together with an SDC-001A camera from Spectec.

B.2 Software

We developed for Windows XP and Windows Mobile 5.0 and tested our protoype on Windows 2000. Microsoft Visual Studio 2005 has been used as our development and debug environment.