AN ARCHITECTURE FOR PROTOTYPING AND APPLICATION DEVELOPMENT OF VISUAL TRACKING SYSTEMS

Mario Becker, Gabriele Bleser, Alain Pagani, Didier Stricker, Harald Wuest

Fraunhofer IGD, Darmstadt, Germany
email: firstname.surname@igd.fraunhofer.de
TU Darmstadt, Interactive Graphics Systems Group, Germany

ABSTRACT

In this paper we introduce a novel architecture for rapid development and assessment of advanced 3D visual tracking systems. Indeed, we notice that it does not exist up to now a universal tracking approach that fulfills the requirements of all possible application scenarios at the same time. On contrary, very specific and performing solutions can be developed for given situations and uses. Therefore, software for visual tracking must be designed as a highly flexible system that can be quickly re-configured in order to enable the development of optimised solutions in terms of accuracy, robustness, frame rate and delay.
to this purpose we designed an architecture that offers many functionalitys, which can be combined together, and thus build a new processing chain. The overall system offers numerous advantages, such as interactive programming, real-time access to the data and parameter at runtime.
Index Terms— architecture, middleware, 3D tracking.

1. INTRODUCTION

Precise and fast tracking of TV-cameras represents a key technology for 3D-TV. Several systems have been developed in the past such as "*free-d*" from the BBC [1]. Nevertheless, these systems require special infrastructure in the environment such as markers or emitters, what restrict strongly their usability and the range of applications. We intend on contrary to develop marker-less solutions that use only natural features and a priori 3D information about the scene [2]. In order to get more stable results, in particular during strong motion, we use a tiny inertial sensor mounted onto the camera [3]. The development of such a system is challenging; we consider that the design and implementation of a powerful software architecture, which offers reliability, fast prototyping, and 3D visualisation of the tracking results to be a key issue.
The system is dedicated to researchers looking for fast prototyping possibilities as well as advanced users who like to adapt given tracking procedures to their specific application scenario.
This paper first presents related work and reviews very briefly existing architecture approaches. It follows an analysis of our requirements and the mapping to software functionalities. Our approach is described in paragraph 4. Paragraph 5 presents major implementation issues and finally two examples of instances of the final software.

2. RELATED WORK

Related existing software architectures can be found in similar domains such as computer vision, robotics, and virtual reality systems.
One well-know system is OpenCV [4], which is actually designed as a classical C/C++ library and offers a large number of utilities and implemented applications. It nevertheless does not support higher-level programming or prototyping and is purely restricted to computer vision.
Architectures for mobile autonomous robots are at the first sight closer to our needs. Those architectures are often built in a hierarchical manner; the data flow is often linear from the sensor over the planning to the actuators modules. Gat [5] proposed a three-layer approach, which consists of a Controller, Sequencer and Deliberator and provides a generic concept for navigation software for robotics.
In the 3D rendering domain, highly configurable systems exist [6]. Based on the X3D and VRML standards, they allow rapid development of VR applications with X3D coding and java scripting. Unfortunately they do not contain extensive image processing or even visual tracking possibilities. Most of these systems are optimised to send data (mostly static) down the graphics hardware and use only small amounts of data to control their behaviour. In a vision system we need to handle large amounts of image data and like to think of stream processing, which is why we choose not to enhance one of these systems but build a new architecture. However, we consider such systems our archetype in terms of runtime configuration and end user application building.

3. MAPPING REQUIREMENTS TO FUNCTIONALITIES

1. Off-line requirements

Fast prototyping: 3D marker-less tracking is highly complex, and today it does not exist one approach that covers all the possible scenarios. Therefore, one must have the possibility to *configure* and *design* a tracking system quickly, at a high-abstraction level and without the need of any re-compilation.

System extension: Any extension of the system at programming language level should not require changes in the overall system.
Integration in VR/AR systems: The final tracking system must be easy to integrate in an existing system and offer different mechanisms and APIs for that integration.

2. Run-time requirements

Performance: the main requirement is of course performance. The complexity and flexibility of the system should not restrict high performance.
Parallelization: In particular multi-core processors should be supported in an efficient way. Parallelization should be independent from the application and thus occurs automatically.
Reliability: The system designed interactively must be very reliable. If hardware components such as sensor or system sub-components are not available it should not stuck the overall system. Re-start of sub-component should be possible.
Testability and tuning: Marker-less tracking is complex, especially if several sensors are used. In order to test and tune the system, each component's parameter and data should be accessible and modifiable during the runtime.
Visualisation: The targeted applications are TV-studios or augmented reality set-ups. For validation and usability the system should offer real-virtual world registration as well as basic 3D visualisation functionalities.

4. DESIGN APPROACH

1. Core concepts

To meet the requirements, we abstract tracking procedures into classes with a fixed interface. These are called "Actions" and contain atomic pieces of an algorithm like an image processing step or feature extraction. Those can be easily reused in other tasks or as a stand alone processing method. The Actions exchange data through a global shared memory, which we call "Dataset". All objects in the Dataset are identified by a unique key which is used by the Action to fetch the data before execution. The Actions are put together to form algorithms by defining data dependencies, which is done by assigning keys to their input and output ports. Another view of this concept is the top-down view: the algorithms are split into small reusable units and encapsulated by the Action interface.
The indirection over a key element for data access allows the use of a graphical interface or a scripting language to setup the data-flow. The Action interface also allows export of internal variables as attributes, these can be viewed and changed in the runtime system to tune process parameters.

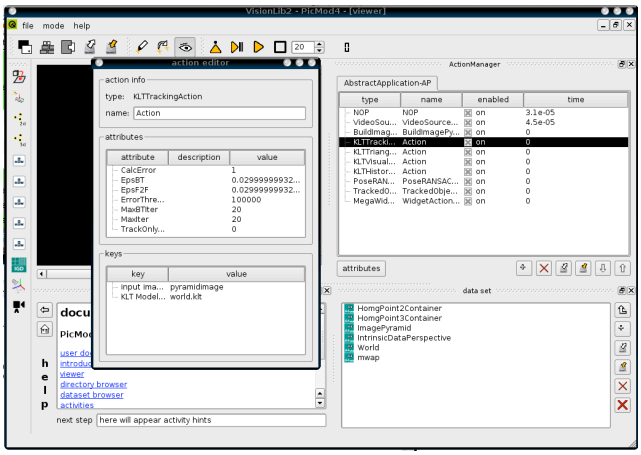
2. Parallelization and synchronization

The system offers three different levels of parallelisation, in order to support state of the art multi-core and multi-processor systems.
The lowest level consists in the classical and straight forward multi-threading implementation achieved by the developer to improve performance of a specific algorithm.
The mid-level is done automatically and is based on dependency analysis of the dataflow. As a result we get a dependency graph which is used to select Actions with independent data inputs, able to run in parallel without any conflicts. This method can be seen from the outside as a sequential processing and does not require precautions to protect the data from access by simultaneous running threads, due to the precondition of having independent data. The method does not require any knowledge about the algorithm and is the easiest to use. Furthermore the user can develop the tracking system in a clear hierarchical manner (top down). Although this method might not produce an optimal result, it is a convenient way to improve performance.
As a third method, our system provides a "Component" which is a configuration of Actions, executed in its own thread. This concept is important for tracking with devices running at different frame-rates, like inertial sensors (e.g. 100Hz) and cameras (e.g. 25Hz). In such a case the sensor interface and a pose estimation algorithm, like a Kalman filter, would run in one component at sensor framerate, while the slower camera and vision part runs in another component.
In order to reduce unnecessary data exchange between these components, the data objects incorporate a locking mechanism. The same mechanism also handles signals to notify other components about changes in the data. Additionally our data objects can be time stamped to identify related data in an asynchronous setup, where multiple sensor inputs must be handled.

3.High-level programming and prototyping

New Action and data classes built into shared libraries can be added to our framework by loading them during runtime. This makes the system extendable without rebuilding it, allowing enhancements by adding more problem specific Actions while reusing the existing Actions for common processing steps.
All Actions can be instantiated from a TypeFactory [7]. This allows creation by name, i.e. Actions can be created "dynamically" from a GUI or a configuration file.
Once the tracking procedure is set up it can be saved to a XML configuration file for later usage within a runtime system or for further editing. The runtime system can be a device backend in a 3D visualisation system.
Image1 shows a snapshot from our GUI front-end. It can be used for algorithm configuration and execution. It also serves as a testbed for newly implemented processing steps and provides generic methods to gain insight into the data as it is processed. Furthermore data specific viewers can be

added together with the processing steps to visualise intermediate data and results. In addition it takes time-samples of the active processing steps which is a first instance in finding performance bottlenecks.



[image1] GUI at work, the left shows the attribute editor, the right the running actions and contents of the dataset.

5. TRACKING FRAMEWORK

5.1 Basic steps of 3D tracking

In order to simplify the implementation of tracking algorithms in the presented architecture, we define a tracking framework consisting of several base classes representing the major steps of tracking procedures. We identified the following steps which are common to all visual 3D tracking systems:

Image processing: This base class contains methods to modify the incoming video images to fit the requirements of the following processing steps.

Feature detection: At this step higher level information such as interest points of an image are generated. This step is specific to a given tracking procedure.

Tracking, **matching or classification**: The process of correspondence determination is very specific to a tracking procedure and can consists in searching features seen in the past images (tracking), comparison with reference data (matching) or features sorting (classification).

Pose estimation: Computation of a camera pose with a given set of correspondences.

Between each of these steps, we defined dedicated data blocks, which are passed from one processing step to another. Additionally, an Action might need static data (e.g. object or scene descriptions). This kind of data can be passed to the Action using the Attributes mentioned above.

5.2 Definition of the 3D environment

The basic idea is that every tracker is fed with a (eventually preprocessed) live video image and a world description, containing one or more objects to be tracked. The tracker(s) then tries to track all tracked objects of the world based on their object models description. The tracking results are written into a pose object in the Dataset.

In our system the real world and our cameras or objects to be tracked are represented by a few elementary data structures. These are the following:

World – A "World" is a container for the object descriptions. It also defines a reference coordinate system for the tracking.

Tracked object – A "Tracked object" is used to describe a real world object that has to be tracked. It consists of one or more object models containing the necessary data for tracking with the different methods.

Object model - The "Object model" represents concrete data for tracking (some instances are listed in section 6). A tracked object can then be described by one or more object models, of different or same type. E.g. an object which will be tracked can be represented by 3 Markers, 1 KLT model and 1 line model, so it can be tracked with any of these methods. The tracked object also contains a 3D-pose representing the translation and rotation of the camera in object coordinates, a tracker uses this pose to store the output of the tracking process.

6. BUILDING BLOCKS FOR VISUAL TRACKING ALGORITHM DEVELOPMENT

For our common processing steps we have a library of implementations, a few of them are listed in the following table:

Image processing:
gradient image, image pyramid, binarisation, edge detection, video sources from file and cameras, …

Feature detection:
Harris, KLT [8], SURF [9]

Correspondence generation:
tracking = KLT point tracking, line tracker
matching = match features with references - MSER [10]
classification = e.g. randomised trees [11]

Pose computation:
generic pose calculation: Kalman Filter, RANSAC (linear or non linear estimation with inliers/outliers tests)
linear pose calculation: HEIV [12]
none linear pose estimation (Levenberg-Marquardt)

Based on this selection from our library we now outline two tracking procedures.

6.1 KLT tracker

This tracker consists of a feature extractor which detects interest points in the image and cuts out image patches acting as the descriptor for a point.

The correspondence generation part tracks these features in 2D under affine invariance with brightness and contrast compensation. The 2D/3D correspondences are determined

by back-projection into a given 3D model or by triangulation of 2D points from different views. This can be combined with non linear pose estimation and RANSAC to calculate the camera pose.

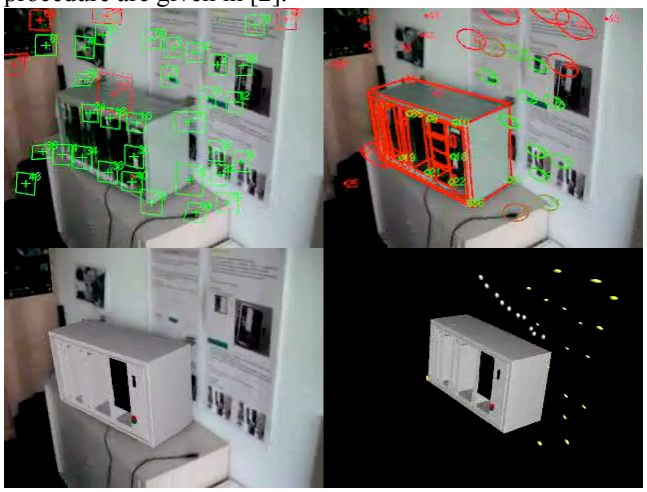
6.2 Line tracker

The line tracker [13] projects a "wireframe" 3D model of an object into an image and uses orthogonal search for gradient extrema at equidistant control points at the model lines to fit the model into the image and produces 2D/3D correspondences, which can be fed into the pose computation. Since this tracker does not use features in the sense of points in the image, it has no feature extraction part.

6.3 Combining tracking methods

It is also possible to combine a frame to frame tracker, like the KLT, with an absolute tracker, such as a line tracker, to compensate drift. Such combinations are very powerful because they can benefit from all tracking methods to overcome problem situations where a single method would fail.

The two procedures described in the last two sections can be combined to form a tracking system, which uses the line method for initialisation after start up or for re-initialisation when the track was lost, and KLT tracking for real-time frame to frame processing. The details about this tracking procedure are given in [2].



[image2]: line tracker and KLT, the red lines in the top right part show the fitted line model, the top right shows KLT features. The bottom left image shows the camera image overlaid with the 3d object, bottom left is a visualization of intermediate tracking results (uncertainties of the KLT point

7. RESULTS

The overall performance of our tracking systems highly depends on the type of tracking and the algorithms in use. The framework provided, can be used to efficiently implement optimisations and multithreaded algorithms using today's multicore processing environments, thus improving performance with less effort. The actual performance achievements can be looked up in the papers describing our tracking implementation [2][3][13].

8. CONCLUSION

In this paper we presented a novel software architecture which allows us to quickly implement new real-time tracking procedures, and provides a comfortable environment for evaluation and tuning. With the architecture we are able to develop complex and challenging tracking system and thus push the current state of the art in this area.

REFERENCES

[1] G A Thomas (BBC), J Jin, T Niblett, C Urquhart (The Turing Institute), "A versatile camera position measurement system for virtual reality TV production", proceedings of IBC 1997

[2] G.Bleser, H.Wuest, D.Stricker, "Online camera pose estimation in partially known and dynamic scenes", proceedings of ISMAR 2006

[3] J. Chandaria at al, "REAL-TIME CAMERA TRACKING IN THE MATRIS PROJECT", proceedings of IBS 2006

[4] OpenCV: "http://www.intel.com/technology/computing/opencv/index.htm"

[5] E. Gat. "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots" Conference on Artificial Intel-ligence, 1992

[6] Behr at. al., "Utilizing X3D for Immersive Environments", Proceedings International Conference on 3D Web Technology (Web3D) 2004. p.71-78

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns", Addison-Wesley, 1994

[8] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision." International Joint Conference on Artificial Intelligence, 1981.

[9] Herbert Bay1, Tinne Tuytelaars2, and Luc Van Gool12 , "SURF: Speeded Up Robust Features", ETH Zurich

[10] J. Matas, O. Chum, M. Urban, T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions", BMVC, 384-393, 2002.0.3

[11] V. Lepetit, P. Lagger, P. Fua, "Randomized Trees for Real-Time Keypoint Recognition", Conference on Computer Vision and Pattern Recognition, San Diego 2005

[12] F. Porikli, O.Tuzel, P. Meer: "Covariance Tracking using Model Update Based on Means on Riemannian Manifolds.", 2006 Computer Vision and Pattern Recognition Conference, New York City, NY, June 2006, vol. I, 728-735.

[13]Wuest, Harald; Vial, Florent; Stricker, Didier: "Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality." ISMAR 2005 : Proceedings of the Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality. p. 62-69