Max-Planck-Institut
für biologische Kybernetik
Arbeitsgruppe Bülthoff

Spemannstraße 38 • 72076 Tübingen • Germany

Technical Report No. 086

The Motion-Lab — A Virtual Reality Laboratory for Spatial Updating Experiments

Markus von der Heyde[1]

December 2000

[1] Cognitive and Computational Psychophysics Department, Max-Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen, Germany, E–mail: Markus.von.der.Heyde@Tuebingen.mpg.de

This report is available via anonymous ftp at `ftp://ftp.kyb.tuebingen.mpg.de/pub/mpi-memos/pdf/TR-086.pdf` in PDF–format or at `ftp://ftp.kyb.tuebingen.mpg.de/pub/mpi-memos/TR-086.ps.Z` in compressed PostScript–format. The complete series of Technical Reports is documented at: `http://www.kyb.tuebingen.mpg.de/bu/techr/`

The Motion-Lab — A Virtual Reality Laboratory for Spatial Updating Experiments

Markus von der Heyde

Abstract. The main question addressed in the Motion-Lab is: **"How do we know where we are?"** Normally, humans know where they are with respect to the immediate surround. The overall perception of this environment results from the integration of multiple sensory modalities. Here we use Virtual Reality to study the interaction of visual, vestibular, and proprioceptive senses and explore the way these senses might be integrated into a coherent perception of spatial orientation and location. This Technical Report describes a Virtual Reality laboratory, its technical implementation as a distributed network of computers and discusses its usability for experiments designed to investigate questions of spatial orientation.

The primary question we have explored in the Motion-Lab concerns the perceived location of oneself in real space as well as in a virtual environment. This location (here including position and orientation) is important for the interpretation of the other senses. For example, perceiving our location enables us to disambiguate between possible interpretations of a visual scene.

Normally, we know our location and it is naturally updated when we move through space. How is this updating related to our perception? Which of our senses contribute to this automatic spatial updating? If some senses, for example vision, do not contribute, but would profit from the update, a strong coupling of several modalities in our perception would be the result. In traditional psychophysics, one specialized cue in one modality is studied (for example, color perception in vision). However, more recently psychophysics was extended to look for cue integration and adaptation effects across modalities. Nonetheless, no general model so far explains the network of influences between our senses. Current experiments in the Motion-Lab will be presented in a later Technical Report to explore inter-modality effects.

1 Spatial Updating

Under normal conditions, our spatial location (position and orientation) in the environment is known to us and is self-evident. In typical virtual reality applications, the relationship between (mostly) visually perceived location and the real body location is broken. Consequently, our position and movements in space are no longer coherently perceived by all our senses. The visual sense is not only one that provides a spatial frame of reference. We also perceive spatial auditory cues and form reference frames of our own body position and motion by proprioception, and external accelerations by the vestibular system. If those senses agree on one reference frame, we feel in a stable environment immersed. There is hardly any way to misperceive one's own location, which might be the reason why humans have difficulty ignoring the stable world. Moreover, some people believe this basic assumption of a mostly stable world is enhancing our perception because it can reduce processing effort. The assumption of the stable world around us reduces most of the analysis to the changing parts. However, people argue, there must be some sort of boot strapping process which allows us to reset or correct our spatial frame of reference in the environment. Of course, it is possible that both principles exist in

parallel. The faster update process would analyze the changing parts and a more sophisticated and slower process would analyze all the information provided from our senses and correcting the spatial reference frame known from the first process.

Different senses provide either absolute or relative information which could be integrated into the frames of reference. **Vision** is known to provide us with excellent measurements about our location and movements in space. Distance to objects, their movement direction and relative speed can be easily extracted in hardly more than the blink of an eye. Optic flow can give us the sense of self-motion, if the visual field of view is large enough. It can be used to navigate through an environment (Beall & Loomis, 1997; Riecke, 1998). Landmarks, in contrast to optic flow, are more reliable and do not lead to accumulation errors (Bülthoff, Riecke, & Veen, 2000). Spatialized **sound** is worse than vision; front-back confusions and a poor spatial resolution produce an unreliable spatial location (Begault, 1994). However, there is an auditory reference frame, which could help to focus our attention towards targets and localize them roughly in space. Disturbances of this rather inaccurate reference frame, achieved by unnaturally changing sound sources, may nonetheless confuse our belief in one coherent or unique reference frame for all the senses. **Proprioception** provides information about our body position. Active and passive changes to body limbs are ”known” to the system and contribute to motion perception (Mergner, Siebold, Schweigart, & Becker, 1991; Hlavacka, Mergner, & Bolha, 1996). Also, the force on different joints introduces knowledge about our position relative to external forces (e.g. gravity). Finally, the **vestibular system** is known to measure changes in velocity by means of linear and angular accelerations. We maintain a strong sense of gravitational direction, which is known with respect to our body axes (Mergner & Rosemeier, 1998). If the sensation of gravitational force is unpredictable or unstable, we have trouble in maintaining straight gait or posture.

So far it is unknown if and how these different frames of reference are integrated to provide one unique and stable reference frame. One could ask how important these sources of information are to our perceived location in space. One approach is to determine each sensory modality’s ability to provide a reliable perception of our spatial location as some of the above mentioned studies did. Naturally, one could not stop the other senses from providing information, but one should make that information as useless (uncorrelated to the task) as possible. The other extreme would be to try to control the input provided to all senses at once and ask, by changing small parts of the system, how much those changes influenced the perceived location in space. Of course, there are lots of ways that could be attempted in the current setup of the Motion-Lab. However, the approach to present a coherent simulated world which provides multiple frames of reference is technically most demanding.

2 Virtual Reality

In the past, researchers have defined Virtual Reality, for example, as:

> *“... a high-end user interface that involves real-time simulation and interactions through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, taste, etc.”*
> (Burdea, 1993)

In addition to the quote given above, Burdea and Coiffet (1994) summarized attempts at defining VR and stated clearly what Virtual Reality is **not**. The authors rejected definitions where the sense of “presence”, the immersion at a remote location, is dominating, as well as definitions where parts of the real environment are replaced or enhanced by simulated features (“enhanced reality”). In addition, all statements associating the definition of Virtual Reality with a specific set of interaction devices like head mounted displays, position sensing gloves or joysticks are not adequate, since those tools can easily be exchanged or used in applications not at all connected to Virtual Reality.

Today, following the definition given above, the functionality of **Virtual Reality (VR)** is often described as complex computer simulation which

exchanges the natural reality of a potential user with a simulated version. The exchange is limited, in most cases, to some, but not all of the senses the user could experience in the simulation. Visual simulations are typically the main part of today's Virtual Reality applications. The visual simulation tries to mimic the relevant aspects of the visual world, creating a simplified version. Natural looking sceneries (Virtual Environments) are the goal of many research projects, a goal that has not so far been achieved due to the overwhelming complexity of even a simple outdoor scene. Nonetheless, the existing visual simulations cover some important features of the visual world like reflections, shading, shadows, and natural scene dynamics. However, Virtual Reality is not confined to visual simulations, but also must include the user, allowing active behavior inside the simulation. The **user** interacts with the simulation via specialized interfaces. Actions of the user cause changes in the simulation and feedback is provided to let the user "immerse into another world". The sense of **presence** can be strengthened by real-time feedback involving the user in Virtual Reality. Very closely related to Virtual Reality is the term **Virtual Environment (VE)** which refers to the simulated environment itself. Virtual Environments can be presented with Virtual Reality technology in multiple sensory modalities.

In addition to the given definition of VR, Burdea and Coiffet (1994) summarized the history of VR. Already in the 1960's, the Sensorama (Heilig, 1960) provided color and 3D video, stereo sound, aromas, wind and vibrations in a motorcycle ride through New York. This device delivered full sensation in multiple modalities, but the user could not interact. Nonetheless, this historical device can be seen as be beginning of Virtual Reality. It provided a sensational experience in advance of many of today's systems. Modern systems actually got away from the presentation of odor and have concentrated on replacing the video input from the Sensorama with a simulated version which is capable of reacting to user input. The user in a Virtual Environment, or in Virtual Reality in general, can interact with the simulated world. The interaction is made possible by different devices for each sense.

VR interfaces have also changed since the 1960's, becoming smaller, more powerful, and lighter, thanks to the development of microtechnology. Visual simulations are presented mostly by projecting the simulated picture onto a screen in front of the user. Another approach is to make displays very small and integrate them into a helmet, letting the user see through a specialized optic system. These systems are called head mounted displays (HMD) an idea that dates back to Heilig (1960). Other equipment is used to enable users to interact with the simulated world. These interactive interfaces enable users to sense multiple sensory modalities simulated coherently by the computer. For example, virtual touch can be simulated by a device called PHANToM, which can be used for psychophysical experiments (e.g., von der Heyde & Häger-Ross, 1998). With this device virtual objects with a wide variety of properties can be touched by the user with the fingertip or a stylus. Other interfaces like virtual bikes or position sensing gloves are used to let the user navigate through virtual environments. The particular interfaces used in any VR setup mostly depend on the application and its goals.

Today's video games and multimedia applications very often use sound, videos, and 3D animations. Some interaction device like a computer mouse or joystick is typically used for control. Even though such applications use multiple modalities which can induce a sense of presence, we can still distinguish between them and true VR. One way of doing so is by comparing the degree of interaction. For instance, we can switch a video recording on and can stop it any time, but there is no real influence on the picture we see. However, the picture will look realistic, since it is normally taken from the real world and not rendered synthetically. Moving further towards multimedia applications, the degree of interaction increases. In multimedia applications one can choose what to do next and where to look for new information. Nonetheless, the information presented as text, video or sound does not react to our input. In today's video games the degree of interaction is quite high. In a simulated

3D environment, the player can change his/her own position, collect objects, fight, and run. Some of the games are close to our definition of VR. Though in these games one can not feel objects the simulated ego picks up, but one can turn the objects around and use them as tools just as in the real world. The boundaries between VR, complex 3D animated games, and other similar applications become more and more vague and defined by the purpose of the application, and some might say by the costs of the system. Imagine a car race simulating a complex 3D environment, providing realistic sounds and using a force feedback steering wheel as input device. This game already simulates three modalities with immediate feedback to the user's reactions. Do other so called VR applications involve that much realism?

3 Motion-Lab

This report documents the general ideas and implementational details of the Motion-Lab. This lab combines VR equipment for multiple modalities and is capable of delivering high-performance, interactive simulations. Crucial design decisions are explained and discussed as the software and hardware is described. The goal is to enable the reader to understand and compare this implementation of a distributed VR system with solutions demonstrated by other labs.

The overview starts with a general discussion of VR systems as simulations for multiple modalities. This is followed by a short discussion of the advantages of distributed solutions in contrast to a mainframe realization. Focusing on the distributed system, basic communication problems are mentioned and one approach for the solution of those problems is introduced as the main communication structure in the Motion-Lab.

The hardware section (see section 4, p. 9) demonstrates the variety of hardware used in the lab and introduces all the devices to the reader. Each piece of equipment is described in terms of its functionality as well as the technical details. Alternative solutions are discussed and the main differences are rated.

Finally, the software concept is described in detail, but without going too deeply into the source code (see section 5, p. 17). The latter is available online in combination with the DOC++ documentation of most of the parts. General ideas about software development which guided this project are compiled into a short introduction to distributed software development in context of multiple OS.

3.1 Overview and purpose

This introduction to the realization of a distributed VR system explains some of the general design criteria and concepts. Different principles are discussed and guide the reader towards an understanding of the overall system. This part is meant as an introduction to the Motion-Lab, as well as a guide for those who start working in the lab and would like to learn basic rules and principles.

3.2 VR systems integrate simulations for multiple modalities

As the reader might know from his experience, the realizations of many so called VR system are confined to the simulation of a visual world. Most of the setups involve at least one interaction device for controlling a virtual camera, allowing the observer to change the view. Nonetheless, our VR definition given in the introduction (see page 2) requires the involvement of more than one modality in the simulation. Some authors like to call the input device itself a device for haptic interaction just because one touches it. Very rarely is force feedback provided in real time for the controlling devices and therefore the information is often going only in one direction: from the user into the system.

In driving simulators, acoustic cues are relatively simple to add and control. Starting a virtual car and changing pitch of the motor noise or simulating other sound properties with respect to the driving parameters like speed is adding to the sensation of a realistic system. Background auditory stimulation has been shown to considerably improve the sense of presence (Gilkey & Weisenberger, 1995). Providing sound which is simulated in three dimensions is more complicated and involves considerably more effort. However, the sense of presence in the VE is significantly enhanced by spatialized sound in comparison to non-spatialized sound as Hendrix and Barfield

(1996) pointed out.

Flight and driving simulators can be divided in two groups by considering vestibular cues. Some of the simulators are mounted on so called *motion platforms* to be moved as whole. The others can not simulate whole body movements (by means of short accelerations) and are called *fix-base simulators*. In both cases, simulations try to move an observer in a large virtual environment. Nonetheless, the simulators themselves stay in a confined space even when they can move a short distance. The mismatch between large changes in simulated location (movements in three dimensions) and the actual position in the laboratory might be one factor of simulator sickness (Virre, 1996). The real accelerations can not possibly be matched to the simulated accelerations without performing the actual perfect movement. The movement type which should closest approximate the important information for the vestibular system of humans is defined by terms of *motion cueing* or *motion simulation*.

In the Motion-Lab at the MPI, visual, haptic, vestibular and acoustic simulations are integrated into the system. Subjects[1] can therefore perceive a simulated world in many different modalities. The simulation of non-spatial sound is clearly the most simple one, due to limited implementation time. On the other hand, it provides ways of instructing the observer even without vision by a synthetic speech system presented via headphones or loudspeakers. The other modalities involve additional hardware equipment which is often accompanied by software and libraries from the manufacturer. Integrating different modalities requires the design of control programs for the specific modality based on different time limits. For example, a visual display runs optimally at a rate of 60 Hz, whereas haptic systems should reach beyond 1 kHz and sound simulations should be even faster (44 kHz to reach CD quality). Showing this wide range of speed requirements, it makes sense to work the devices in a parallel manner, not disturbing each other.

[1]Persons participating in the experiment are in the following referred to as masculine or feminine. It is understood, that the respective other gender is meant to be referred to as well.

3.3 Distributed system or stand alone computer?

There are mainly two distinct ways of achieving powerful simulations:

- The “big solution” runs on one very fast mainframe computer providing all the connections to different parts of the equipment.
- The distributed solution connects smaller and specialized computers which are connected to one device at a time, but provide sufficient speed to run this device at the required speed.

These two solutions are discussed by focusing on their respective qualities and drawbacks in the two following sections. The advantages of one solution are very often the disadvantage of the other.

3.3.1 The “big solution”:

One of the obvious but important advantages of having one big computer which does everything in the VR-setup is that no synchronization between multiple databases for different modalities is necessary[2]. In this solution there is just one OS and one set of libraries involved. The maintenance is therefore low in sense of work for a technician, but the costs for the special hardware which might be involved are considerably high. Furthermore, an advantage which clearly separates this solution from the other is the possible load-balancing across modalities. The programmer can design his program in a way that the most urgent work is done first. In addition, the “big solution” can reliably synchronize tasks below a time resolution of 1 ms.

The biggest disadvantage might be the missing support for some special hardware with a given OS. It seems to be difficult or at least much more expensive to get some parts and interfaces changed later. The system mostly stays as it is because it is hard to extended only a part of it. The overall costs are considerably high, since the computer needs to be in the high performance sector of the market. Special cooling and noise problems

[2]Nonetheless, most VR programs still use different object trees for different modalities. However, there are recently several approaches which try to integrate, for example, sound, haptic and vision into one representation.

might occur and additional problems are posed by short cabling or other interface communication.

3.3.2 The distributed solution:

There are some advantages of the distributed solution which could at the same time be seen as disadvantages of the "big solution". It is more flexible and easy to extend the system gradually or to substitute parts of the whole system. The use of special OS platforms and special libraries for those platforms becomes possible, since not all the different computers have to run the same OS. For most of the parts, it becomes possible to use standard components, which are more common and have the advantage of lower investment costs. Different parts of the simulation – or let's say the system – are running independent of each other which makes critical parts safer from general crashes of the system. The stability of the overall system increases since the nodes of a computer network can replace each other in functionality.

The biggest disadvantage is the communication overhead of the system for synchronizing the data. No real synchrony is possible, but if the speed of the system if sufficiently high, the reached synchrony is acceptable for some projects. Ryan and Sharkey (1998) propose a smooth connection between asynchronous update for objects close to the observer, allowing real-time interactivity, and synchronous, but delayed update for distant objects. The authors argue that network latency (differences in time) thus will not cause discontinuity in space for the user.

3.4 Distributed components and asynchronous communication structure

We decided to implement the Motion-Lab as a distributed VR system. The following section will explain how the communication latency is actually made acceptable for our system by implementing a "soft synchrony" strategy. The communication is the crucial point of a distributed system, especially when different processes have to provide fast feedback at different speeds. The main point of the communication in the Motion-Lab is the asynchrony of all processes and the explicit statement that there is no guarantee for a special message to be accepted by the recipient at a given point in time. Moreover, the information has to be coded in a way that provides the current state and additional information which allows the recipient to extrapolate the status into future.

Let us illustrate the main problem with an example. Imagine the situation where an input device (e.g., a joystick) is controlled by interrupts on a system level and therefore has a rate between 1 and 60 Hz. On the other side, a motion platform is updated very strictly with 30 Hz. The simulation in between has to connect to both devices and the programmer chooses to run it at 10 Hz. In some of the simulation steps, there is no new input from the input device, but the system just takes the last known value. In other simulation steps several records of the input devices had been available, but the last record is the most important, since it codes the most recent state. For the simulation it might be sufficient to always take the last known record to update, for example, an internal model to move a virtual observer forward. Based on the internal state, the simulation can therefore send information to the motion platform, which would arrive there at a rate of 10 Hz. The platform needs to interpolate now for at least two steps between two new data records in order to come up with a smooth movement at 30 Hz rate.

There are two opposing principles working here: One to slow down update rate (from 60 to 10 Hz) and the other to interpolate in time to increase update rate (from 10 to 30 Hz). If the process which is providing information is running at a higher speed (faster update rate) than the consuming process, it is always safe to take the last record which was available. Having the situation the other way around would then result in a jumpy movement and would cause noticeable disturbance for visual and vestibular simulations. Therefore, if at some point the consuming process is running faster than the data records from the providing processes arrive, the program should extrapolate from the last know record into the future to guess the momentary status of the system. If at each point in time the status of the system (take position of an observer as an example) is know, together with a prediction of the rate of change (velocity), extrapolation becomes easy.

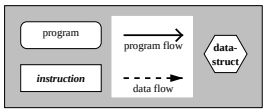


Figure 3: Legend for all flow figures (2, 4, 8, 11, and 19).

This extrapolation method is also useful when short breaks in the system make the information flow unsteady and change the update frequency. Since a lot of different devices work at their inherent speed or changing rates, it makes sense to soft synchronize them using the above principles (see Fig. 1).

3.5 Synchrony, and closed- or open-loop functionality

Synchrony is an issue by itself, when different modalities are involved. If someone drops a cup and one hears it breaking on the floor before seeing it happen, the situation would seem unnatural to us. Having the sound reach the ears later than the visual event reaches the eyes would, on the other hand, feel normal when seen and heard from a distance, since sounds travels more slowly than light. Extending the distance further, one would always expect to perceive the lightning before the thunder. Events in the real world often provide feedback in multiple modalities. If we substitute some of the modalities in the simulation with a virtual version, we should provide the same synchrony. Exceptions are experimental paradigms explicitly dealing with time differences as done by Cunningham, von der Heyde, and Bülthoff (2000a)[3]. In closed loop experiment it will therefore be necessary to provide synchronized feedback for events which where caused by the observer without a noticeable loss of time. The feedback should be provided in a closed loop so that every action is directly coupled to its effect (see Fig. 2).

For an open loop condition the observer has no influence on the occurrence of events in time[4].

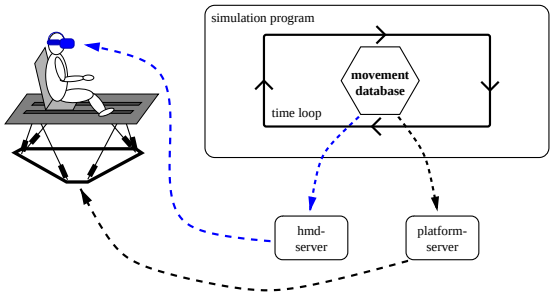


Figure 4: The open loop condition is in comparison easier because most of the events in the simulation can be calculated beforehand, stored in a movement data base, and do not have to be updated based on the users action.

The system gets simpler when it does not provide feedback to the actions of the observer (see Fig. 4): The simulation can be reduced to a playback for different modalities in a predefined time schedule. If the accuracy of time resolution and synchronization on a system level is guaranteed, the playback will appear synchronous to the observer[5]. One could, in this situation, exactly define events to occur at a certain point in time and compensate even for slow transfer rates, if the simulation is completely known beforehand[6].

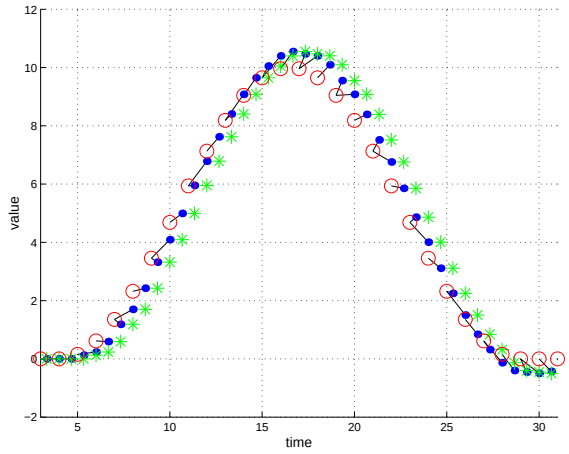
In contrast, the closed loop condition demands that the system react with respect to actions of a observer. The level of interaction determines the level of feedback required to let the simulation appear realistic. For example, in a simulation of a race car the steering wheel is the primary interaction device. If the wheel is providing the steering angle to the simulation, the camera could be updated simulating a moving observer. Driving in this simple simulation of a car does not feel real; the “sense of being there” is quite low. Adding force feedback centering to the steering wheel would improve the feeling of driving a real car. Furthermore, the driver can tell from the haptic feedback alone whether he is going straight. Extending this idea even further, the force model of the steering wheel could include the speed of the

[3]Even then we have to know the exact point in time of certain events in order to add additional time offset in the program.

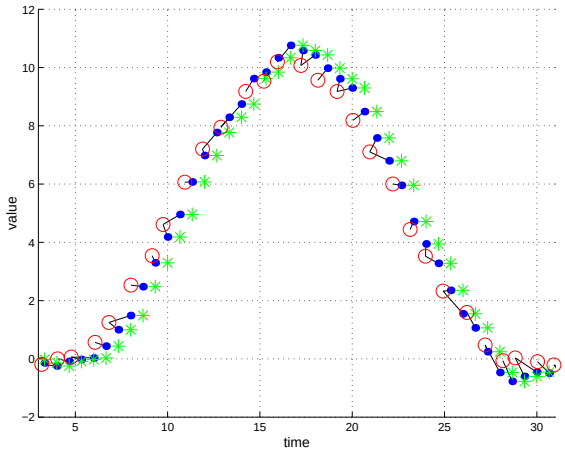
[4]Note: If the simulation provides no feedback to actions, it does not fulfill the given requirements for VR!

[5]Ignoring processes which have order effects and take history into account.

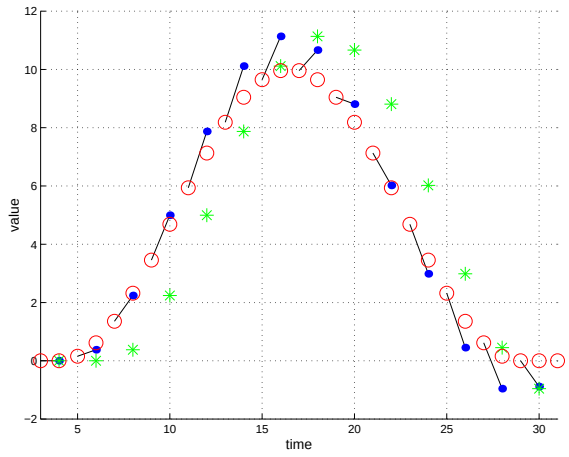
[6]A lot of “fun-rides” in Disneyland, for example, are well predefined and worked out for play back. If decisions of the observers are taken into account, alternative outcomes are defined and the observer works down a tree of decisions.



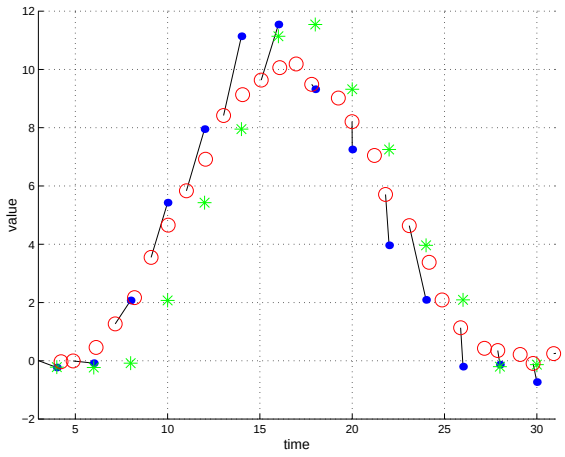
(a) Oversampling with factor of 1.5



(b) Oversampling with 1.5 + noise



(c) Undersampling with factor of 0.5



(d) Oversampling with 0.5 + noise

Figure 1: The extrapolation method can provide smooth paths even when the provided data flow is not smooth and in addition unsteady in time. Two examples show that the extrapolation works for under and oversampling relative to a given frequency. The left panels show data results for two frequencies without noise. The right panels show for the same frequencies the results for a situation where time and data are overlayed by random noise adding $^{+}/$-25% of the respective units. The one dimensional case can be generalized to serve all six degrees of freedom for camera or motion-platform data. The red circles indicate the transmitted position data on with bases the last velocity was calculated. The blue dots indicate the derived position connected with a thin black line to the data point they are based on. The green stars indicate a potential rendered image based on the blue position at the time the image would be displayed. Unsteady and changing frequencies normally cause “jumps” in continuous data when the stream is re-sampled with a fixed rate. In contrast, this extrapolation method predicts a future point based on position and velocity information. Even sudden changes of rate (due to incomplete data, for example) will not disturb the smoothness of the data. Due to the velocity prediction the algorithm overshoots, displaying behavior similar to low-pass filters.

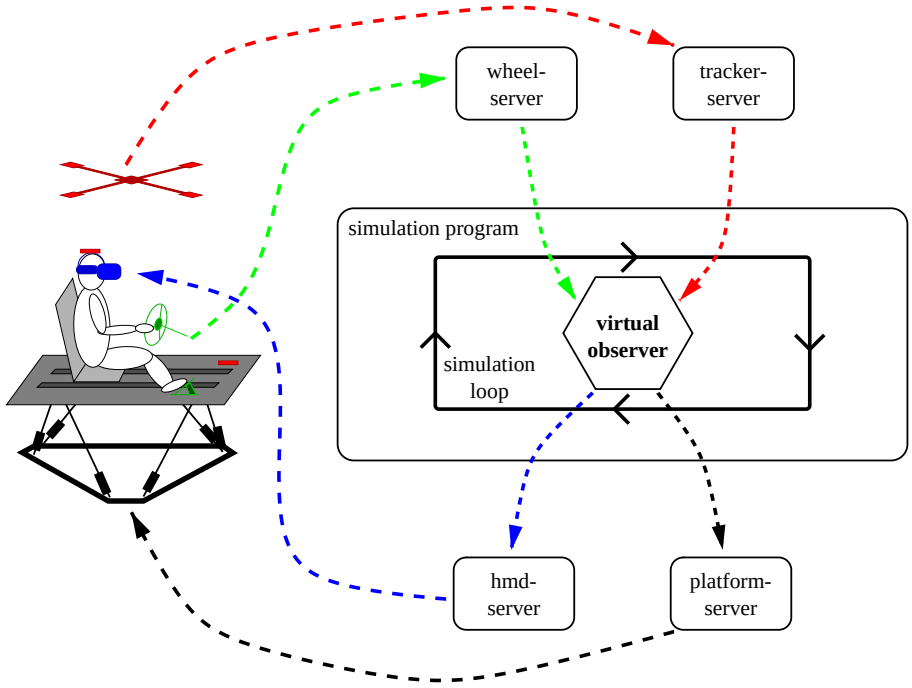


Figure 2: The closed loop simulation feeds back the actions of an observer to the modalities he experiences in the simulation. Every action is coupled to the reaction in the simulated world.

car: turning the tires on the spot would be harder at low speeds and so on. The information provided by the combination of both modalities (vision and haptics) is similar, but the coherence and synchrony makes the simulation more realistic. Including sudden jerks when leaving the road or velocity coupled noise, for example, would add information which could not be visually perceived. This example offers no systematic proof that synchrony and coherent feedback in different modalities make a better driving simulator. However, it makes a plausible suggestion of what could be gained by having those features. So far, there has been some evidence that presence, the "sense of being there", improves task performance in VR (Witmer & Singer, 1998). However, as Witmer and Singer pointed out, presence is negatively correlated with simulator sickness, which leaves the direction of causality between the two unclear.

4 Hardware

In general, the hardware is standard commercially available equipment, with the exception of the force feedback steering wheel[7]. For later ease of reference, the next sections include short descriptions of the different devices and their basic working principles and functions. The information is mostly provided by the manufacturer, but is all rephrased and simplified for the purpose of this technical report. Specific questions referring to technical data or functional details should be directed to the addresses given in von der Heyde (2001). A general overview of the Motion-Lab equipment is given in Figure 5.

4.1 Motion platform

The central item in the Motion-Lab is the Maxcue motion platform from Motionbase (see Fig. 7). It was built and designed after the Stewart platform principle: Two bodies are connected by six legs, which can vary in length (Fichter, 1986). One of the bodies is traditionally named base and the other platform. In our case the base is connected to the building letting the platform move by changes in the six cylinder lengths. The Max-

[7] It was designed and constructed in the institute's own workshop.

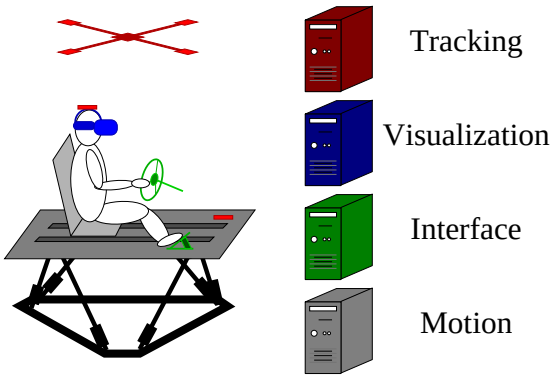
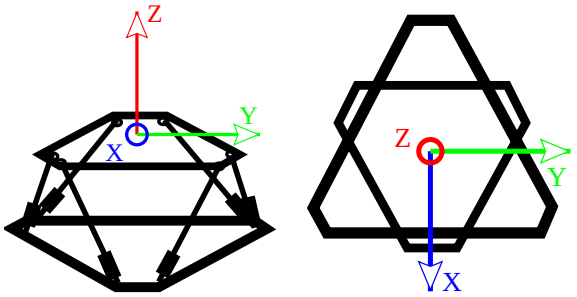


Figure 5: The Motion-Lab setup in its main parts consists of the motion platform with a seat for a human, interfaces for him/her to control his/her position in the virtual world, the visualization of that world presented by an HMD, and a tracking system to render the corresponding viewing angle for the measured head position. Each device is controlled by a separate computer to guarantee optimal performance.



(a) Front view (b) Top view

Figure 6: The coordinate system for the motion platform is at the same time the general coordinate system for the simulations for the whole lab.

cue motion platform has six electrically driven cylinders which are symmetrically arranged between base and the platform frame[8]. The platform is able to perform movements in all six degrees of freedom (DOF), so it can turn around three axes and move in all three linear directions independently. The coordinate system for the platform is identical to the coordinate system for the simulations of the whole lab: The X axis points away in front of the user sitting on the platform, and the Z-axis points upwards which completes the right hand coordinate system with the Y-axis pointing to the left of the user. Therefore, the rotations around the X-axis is called roll, the one around the Y-axis pitch, and the rotation around the Z-axis is called yaw (see Fig. 6). Normally those terms are used by pilots, but have been adopted here for the technical descriptions. For describing human orientation in space we use the same names for simplicity[9].

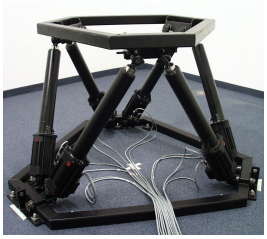


Figure 7: Maxcue motion platform

The actual control of the platform's movements is achieved in several steps (see Fig. 8). A data record which contains six numbers, one for each DOF, is given to the platform library at a rate between 30 and 100 Hz. Depending on the filter parameters, these values can be interpreted as accelerations, velocities or positions[10]. These values are passed on to the DSP board in the motion control host computer by the library provided by Motionbase. This board implements the digital filters in hardware with the given parameters. The filtered values are converted into cylinder lengths by the inverse kinematics of the platform. The cylinder lengths are derived from the given six DOF position by calculating the transformed frame mount points of the legs from the normal setup geometry with one matrix multiplication. The Euclidean distance from the base to the transformed leg positions on the frame is the length of the cylinders. Therefore, there is only one solution for the inverse kinematics. In contrast, the forwards kinematics is more complicated and probably not analytically solvable, but approximately solvable by a multidimensional Newton algorithm for any required accu-

[8] Besides a few restrictions on the position of the legs, the endpoints of the cylinders are arbitrary.

[9] Others prefer to use the terms tilt, nick, and heading

[10] The programming of the filters is subject to a nondisclosure agreement and therefore can not be discussed.

racy of the calculation. This calculation would, if needed, enable the library to recalculate the actual position of the platform for control reasons given the lengths of the cylinders. Nonetheless, this additional control is not yet implemented in the Motion-Lab Library.

There are many similar motion platform systems on the market, mainly being divided into two groups: The legs are either moved by electric motors or by hydraulic pressure. Pneumatic systems can be classified as hydraulic systems, since they share common features. The general advantage of hydraulic systems is the smaller size of the legs; they can generate higher forces with less technical effort. On the other hand, there is always a compressor needed for generating the force, which is usually very noisy and has to remain close by to allow rapid changes in pressure. The seals of the hydraulic cylinders have the duty of maintaining the pressure inside the cylinder and therefore add high friction to the cylinder. The result is very often a noticeable jump at the beginning of a movement. This can cause disturbances, especially at turning points of one or more cylinders. In contrast, the electric system can start movements more slowly and have smooth turning points. The resolution of the length control for one cylinder can be very high with the smallest step being 0.6 μm in our case. On the other hand, the small steps can cause micro-vibrations, which can be disturbing: The person sitting on the platform can notice the movement by the vibration before it is actually possible to feel the movement by visual, vestibular or proprioceptive cues. In our lab, those vibrations can be covered by very small changes in position driven by white noise which causes constant vibrations and sufficiently covers the actual onset of a larger movement. As the reader can see, each and every system has certain problems with the start of very soft movements. However, the systems also differ in the maximum frequency of movements they can perform. The electric systems are in general faster[11] since the latency of the hydraulic systems to react to small pressure changes is quite high.

[11] Our system is designed to perform active vibrations up to 25 Hz.

4.2 Head Mounted Display (HMD)

The visual simulation in the Motion-Lab is presented to the user via an HMD. An HMD combines, in principle, two small displays with an optic lens system, which enables the user to see the small displays at a very close distance while focussing to a comfortable distance. The displays are mounted together with the optic lense system inside a cover. The helmet can be placed on the head of the user like a bike helmet (see Fig. 9.c). When considering various models by different manufactures, several points have to be considered. The resolution and technology of the display is naturally important for the visual impression. Recently, LCD's became common and increased the possible resolution. However, because of the illumination decay of LCD, they sometimes present the picture too slowly and afterimages appear. CRT's on the other hand are available in higher resolutions but are considerably heavier. The optic lense system itself is equally important, since distortions and color changes could disturb the presented picture. The most important factor of an HMD is the field of view for the user (Arthur, 2000). Today's best helmets typically cover 40°-60° of horizontal visual field, whereas the human visual field covers more than 190°. The larger the field the more natural the view inside looks[12]. A small visual field, in contrast, can cause simulator sickness and may cause spatial disorientation. Another factor is the weight of the helmet which can cause fatigue. There is usually no external support for the 0.5 to 3.5 kg of an average device. A study by Cobb, Nichols, Ramsey, and Wilson (1999) summarizes the serious effects caused by HMD's as virtual reality-induced symptoms and effects (VRISE).

The helmet we chose to use in the Motion-Lab is the ProView XL50 produced by Kaiser (see Fig. 9). The two LCD's present a visual field of 40°x30° at a resolution of 1024x768 pixels. The refresh rate is fixed to 60 Hz which should be provided by the computer generating the standard XVGA signal. The weight of 980 g is relatively low such that the helmet can be worn for up to

[12] Some tasks are also known to require a larger field of view (e.g., driving a very large boat).

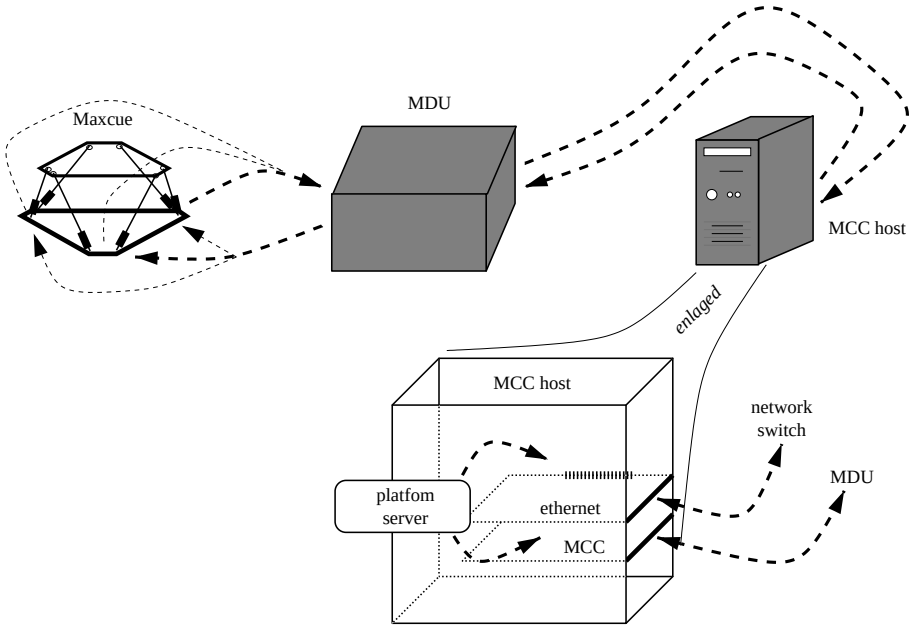


Figure 8: Information flow for the platform control: from numbers to positions. The host for the Motion Control Card (MCC) runs the actual server application. This application connects the input and output from the Ethernet with the MCC. On the MCC the platform positions get filtered and transformed into the actuator (leg) lengths necessary to move the platform. Those values are transfered to the Motion Drive Unit (MDU) where they are amplified to control the motors of the platform legs.

(a) Half front view (b) Side view (c) Subject with HMD

Figure 9: Kaiser Head Mounted Display ProView XL50

60 minutes without discomfort. Compared to projection systems, HMD's have the general advantage that they can easily be replaced by a newer model with better performance/resolution, lower weight and bigger field of view.

Other visualization setups are possible in the Motion-Lab, but have not been implemented yet. In principle, one could use a small LCD projector and a fixed screen both mounted on top of the platform. The projection would have to be carefully coupled with the performed motion of the platform in order to generate a good impression of a stable world. An HMD blanks out all vision of the exterior room, but having plain view around (especially seeing the platform itself) might result in other problems yet to be solved.

4.3 Force feedback steering wheel and analog control

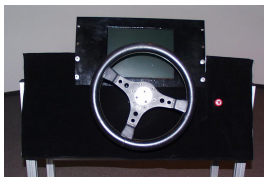


Figure 10: Force feedback steering wheel (here in the VE Lab)

The force feedback steering wheel is the primary input device for driving applications (see Fig. 10 for the setup in the VE Lab and Fig 20 for a picture on the motion platform). This custom built device is constructed to have maximum flexibility and is adjustable for heights, steering angle, and distance to the driver. A high force motor is controlled by an analog card who's signal is amplified and transformed for the motor-control (see Fig. 11). A potentiometer measures the current steering angle and enables a fine force control in a local feedback loop. Standard pedals can extend the functionality of the wheel and be used for breaking and acceleration. The current implementation was adopted from game pedals which were connected to the same analog card. The analog card (AT-MIO-10-16E) has more ports which can be used for further extensions. It can sample the data and provide analog output at rates up to 10 kHz. The local force feedback control can therefore be very rapidly and accurately controlled. The wheel is also usable in the other lab of the MPI in front of the cylindrical screen (Cunningham, von der Heyde, & Bülthoff, 2000b; Cunningham et al., 2000a).

4.4 Joysticks

Joysticks are commonly used in computer games. Therefore, many are constructed to be connected to the standard game-port of a soundcard. Joysticks are handheld devices which enable simultaneous analog control for multiple axes. Simple versions have two axes and more complex joysticks can have up to four axes. Normally, the stick is returned to neutral center position by springs. Modern versions have small motors driving the joystick back enabling the simulation of changing forces. Multiple joysticks can be used in the Motion-Lab as input devices to the simulation (see Fig. 12). The simplest (Fig. 12.a) has two separate analog axes and two buttons for digital answers. A more complex device (Fig. 12.c) has, in addition, two axes for special controls which should emulate functionality of a helicopter: The foot pedals are negatively coupled and the handle includes up and down movements as well as a digital button emulation for rotations. The last joystick (Fig. 12.b) combines the two axes of the first one with a horizontal turn axis and a large number of buttons. This device can give dynamic force feedback if the application addresses a special driver. In general, the control of the joystick is done via the game-port and therefore triggers an interrupt of the system. The maximum data rate is limited in the current implementation to 60 Hz.

4.5 Tracker

In order to take the body movements of the user in to account, it is useful to track these movements. This can be done by mechanical devices (like the joystick) or allowing free movements using other tracking devices. Several systems employing different methods are available on the market. Based on high frequency magnetic fields the Fastrak (Polhemus) system or Flock of Birds (Ascension) are the most commonly known. These systems are sensitive to metal in the direct environment and therefore not recommended in the Motion-Lab. Other systems measure the time differences of submitted and received ultrasonic sound signals with multiple microphones (CMS 70P from Zebris) performing triangulation calculations on the data to calculate position. Optical systems provide the best performance, but due to the need for

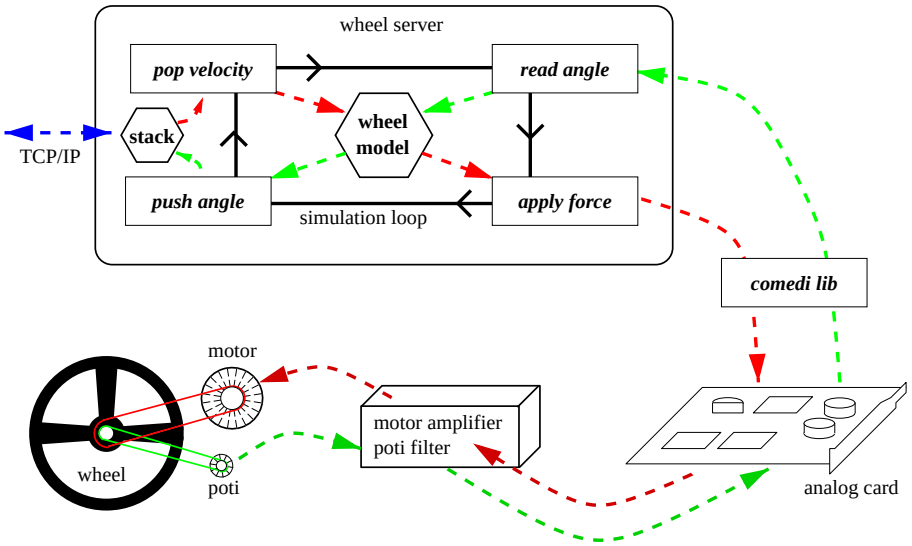


Figure 11: The control of the steering wheel: The wheel server gets data via the TCP/IP connection and updates its internal model of the steering wheel. Afterwards the actual angle is read and a force calculated based on the steering parameters. The force is applied to the wheel via an analog card and an amplifier. The steering angle is read out from the potentiometer coupled to the wheel. The angle is converted from analog current to digital values by the analog card. In the simulation loop of the wheel server, this value is transfered back to the stack and send of to the main simulation.

high speed cameras, also have the highest price (Optotrak from Nordern Digital Inc.).

For the Motion-Lab we use the IS600-mk2 tracking system from Intersense. This tracking device combines two principles for tracking motions: an ultrasonic system and inertial sensors. Both systems have several advantages which are combined to come up with a six DOF measurement for up to four tracking units. One tracking unit (see Fig 13.a) traditionally consists of one inertial cube and two ultrasonic sources (beacons)[13]. The inertial systems are updated at a rate of 400 Hz and therefore provide fast feedback for rotations, but not for linear accelerations. The device has a low acceleration detection threshold under which it cannot record movements. The inertial system is therefore susceptible to slow drifts. The ultrasonic subdevice, on the other hand, works on an absolute scale. Each beacon is triggered by an infrared LED flash and shortly afterwards produces an ultrasonic sound. This sound is recorded by four microphones located at the end of the cross bar mounted on the ceiling (see Fig 13.b). As the beacons are triggered in a sequential order, each additional beacon reduces the tracking speed[14]. An individual distance estimate is calculated, from the time difference between the infrared trigger and the first sound arriving in each of the four microphones. For each beacon, the four values are combined into one measured position in space. The positions of the two beacons of one tacking unit are combined with the data of the inertial cube to yield all six DOF's.

Our tracker has two tracking units which can, for example, be used for tracking the movements of the HMD and the platform in all six DOF's. The difference vector between the platform and the subject's head can be used to move a virtual camera for the VR simulation. Naturally, the tracking device can be used for other things as well. For example, one could track pointing movements of an arm or hand.

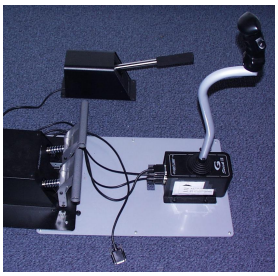
The communication between the tracking box (see Fig 13.c), which integrates the different measurements, and the simulation computer is done via a serial line which causes some additional

[13] Other combinations can be configured with a special configuration language via the serial line.

[14] A new version of the system overcame this limitation

(a) Standard model with two axes

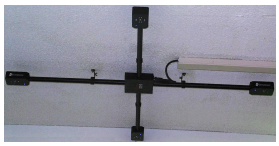
(b) Microsoft Sidewinder with force feedback



(c) Helicopter control

Figure 12: Collection of joysticks which could be connected to one of the game-ports and used for experiments.

(a) Tracking unit (two beacons + one inertial cube)



(b) Cross-bar

(c) Communication unit

Figure 13: The six DOF tracking device IS600-mk2.

latency. The overall latency of the system can be improved by separating the translational and the rotational signal. Since rotations cause bigger changes in the rendered picture, it is more important to integrate them with minimal latency. Luckily, the rotations are mostly based on the integrated signal of the inertial cubes which operate independently of the number of units at a high speed. The rate at which the system is currently used depends on the configuration and lies between 60 and 200 Hz.

4.6 Sound cards

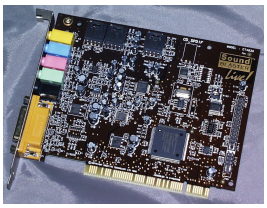


Figure 14: The Sound Blaster Live! is used for sound effects, speech synthesis, and for the connection of joysticks.

Sound is generated by a standard sound card (Sound Blaster Live! from Creative) which is shown in Fig. 14. The Linux driver is currently able to control the sound stream at a rate of 22kHz for both stereo channels. If needed, sound can be sampled in parallel at the same rate. Up to 32 sounds effects or speech outputs can be overlaid at the same time providing a complex auditory scene. We use multiple cards in the Motion-Lab to control speech and other sound effects. Different channels, for example, are used to control vibrations of force transducers (see section 4.8). In addition, the sound cards provide the game-port connector for the joysticks (see section 4.4).

4.7 Headphones

Figure 15: Aviation Headset HMEC 300 with high noise reduction

As sound is an important feature in VR simulations, it has to be carefully presented to the user. In addition, it is important not to let the user perceive sounds from the real world. Beside the disruption of the immersive feeling, external spatialized sound could provide auditory room context from the real environment. The Aviation

Headset HMEC 300 from Sennheiser (see Fig. 15) is used to provide sound to the user in the Motion-Lab. These kind of headphones are normally used by helicopter pilots to reduce the noise of the engine. These special active noise cancellation headphones effectively reduce the environmental noise during the simulation, and make the use of external sound sources as spatial references points (auditory landmarks) impossible. High frequency noise is passively reduced by special ear cushions. As the low frequency part of the noise cannot be reduced passively, active noise cancellation is used. The active noise cancellation uses the principle of canceling waves: Fitting the incoming noise, the systems adds the exact same sound with a temporal phase shift of 180° (opposite phase) so that the sound waves cancel out. In addition to the noise cancellation, the headset provides a microphone mounted on a flexible boom. In experiments where a verbal response from the subject is needed, sound can be recorded and provided to the operator.

(a) Amplifier SAM-200

(b) Tactile Transducers FX-80

Figure 16: The force transducers of the Virtual Theater 2 are used for high frequency vibration simulation.

4.8 Force transducer

Vibrations are sensed by the human skin. In vehicles, vibrations are often connected to motion. To simulate motion in VR we integrate special vibration devices into the system. In the Motion-Lab, vibrations can either be simulated by the motion platform or by the Virtual Theater 2 (VT2) from RHB which includes the amplifier SAM-200 and two Tactile Transducers FX-80 (see Fig. 16). Force transducers function like normal speakers, but without a membrane, transmitting the sound directly to the base plate of the transducer. One can compare them with powerful subwoofers, but force transducers do not generate a sound wave. The motion platform itself can simulate vibrations with high precision in independent six DOF but only up to 25 Hz. The force transducers on the other hand simulate vibrations from about 10 Hz up to 150 Hz. The direction of vibration is perpendicular to the mounting plate of the transducers and therefore only in one direction. Amplifying a normal mono sound source for low frequencies allows the simulation of car vibrations and other “natural” or technical noise realistically. The force transducers can also be used to cover the micro-vibrations from the platform effectively.

4.9 Computer and special graphics

There are several computers in the Motion-Lab with different duties. They are special, either in terms of their OS and library combination or for their special hardware and the corresponding library or both. Not all of the computers are used in all experiments, since it depends on the interface and devices used for interactions.

4.9.1 Sprout

This machine is running IRIX 6.5 and recently replaced an older machine running IRIX 6.2. Both OS’s are supported with different combinations of `o32`, `n32`, and `n64` library styles. IRIX is used in the lab for the driving dynamics, which are not included in this technical report. Furthermore, IRIX is used in the VE-Lab of the MPI for the Onyx2 computer displaying VR simulations on the 180° screen. Since the steering wheel could also be used in that lab, IRIX is one of the main clients for the steering wheel devices.

4.9.2 Cantaloupe

This is the Linux computer for the steering wheel control. It moves with the wheel between the two labs mentioned before. The high speed analog/digital card is built in as special equipment for the control of the steering wheel (see section 4.3). In addition, the computer can produce sound with the functions from the MotionLab sound scripts (see section 5.1.2, p. 19).

4.9.3 Cucumber

This Linux box is the main computer for most of the simulations. It also connects to the tracking system and to one of the joysticks. It hosts the same sound cards as mentioned before for Cantaloupe. The main simulation is not demanding in terms of calculational power, but in the sense of high reliability of timing: The main loop should run between 10 and 100 Hz depending on the precision and latency one would like to achieve in the simulation.

4.9.4 Borage

This computer hosts the Motion Control Card (MCC) for the motion platform and uses the library provided by Motionbase. In the beginning, this library was only available for Windows95 but was recently extended to WindowsNT. However, we decided to let the system run the old version, since we did not experience any complications[15]. The task is not very demanding, but constant timing for providing new data to the library, and therefore for the platform, has to be guaranteed.

4.9.5 Soy and Tofu

Both machines are identical in most of the technical data and are handled as twins in the MotionLab. They run WindowsNT since the graphics driver has the best quality for this OS[16]. The graphics system is designed to provide high resolution images with full screen anti-aliasing in 60 Hz for most of the virtual scenes used in the lab. The graphics power is provided by four graphics cards per machine connected by a specialized board to calculate the anti-aliasing. The graphics system is called Obsidian graphics and is delivered from Quantum3D in combination with the machines, called not without reason Heavy Metal.

[15] The system had once an uptime of more than 180 days!

[16] They also could run Linux, but the support from the manufacturer has its emphasis on WindowsNT.

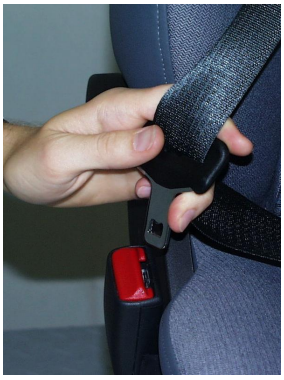
4.10 Security features

Some security features ensure the safe usage of the lab. It is obligatory to use the seat belt (see Fig. 17.a) any time the platform is operated. In the unexpected case of an emergency, the subject can use the emergency switch (see Fig. 17.c) to stop the platform at any time during the simulation. The switch is directly connected to the MDU (the amplifier) and activates the security function. Because of the unknown position of the platform at the moment when the security circuit is interrupted, it is not sufficient to turn off the actuators immediately. The platform could in this case sink to an oblique position due to the mass distribution. Instead, the platform is driven back to the park position (all cylinders short) and after a delay of two seconds after the detection of the emergency halt is physically switched off. The same security circuit can be disrupted by the light beam (see Fig. 17.b) which detects persons entering the simulation space around the platform or by the operator himself at the console. More switches can easily be added on demand.

4.11 Network and other devices

Beside all the specialized hardware, there are some general devices which are necessary to make the whole setup work. The computers are locally connected in the lab via a high speed switch (3com SuperStack II Switch 3300 with 12 switched 10/100BASE-TX ports). The switch is integrated into the campus net via fiber optics connections. Therefore, the equipment is theoretically usable from all over the internal net given the above mentioned OS platforms and the MotionLab Library (see section 5.6).

To handle several computers with one set of monitor, mouse, and keyboard, we use a PolyCon console switch. Due to limited space and simplicity reasons, this solution was chosen instead of a whole range of monitors and keyboards for controlling the different computers. Most of the current work could be done via the network, but especially for graphics and quick status overview it is



(a) seat belt for the subject

(b) light beam at the door makes sure no one enters the danger zone

(c) emergency break for the subject on the platform

Figure 17: Security features are used in the Motion-Lab to ensure a secure usage of the lab. The subject has to use the seat belt which is built into the seat (a). In case of an emergency, the subject can stop the platform at any moment without the help of an operator with the emergency break (c). The operator has a similar switch to stop the platform. The light beam (b) is stopping the platform as soon as a person is entering the close space around the platform.

more comfortable to have the actual console close by. The computers in the lab are secured against power failure up to several minutes by means of a Smart-UPS 2200. This device uses batteries to provide 220V power even when the power network fails. For safety reasons it is important to have the control computers running, even when the platform itself stops. The behavior of the motion platform during a simulation where just the control PC fails would be unpredictable.

5 Software

Since the software is a central part of the functioning system, it is necessary to describe the underlying principles which enabled us to achieve certain goals. This section of the report therefore presents the basic principles of the implementation of the Motion-Lab Library. They can either stand for themselves or – even better – be realized and thereby proven to work. In this case, it makes sense to talk about the concept and the realization together. A certain style of software development guided the implementation and concepts which will be introduced. Last but not least, this section documents which software was neccessary for the development of the experiments and programs running the lab. This project would not have been successful without a lot of different tools and libraries.

5.1 General software environment

The commercial operating systems (OS) used are Windows95, WindowsNT, IRIX 6.2, and IRIX 6.5, and were bought from the respective companies (Microsoft or Silicon Graphics). The low-level library (for Windows95) for controlling the platform was included in the delivery of the Maxcue motion platform from Motionbase. For the development of the simple 3D model of the second experiment (see section 5.7.3) the program Multigen was used. The rendering of the model on the specialized graphics systems (see section 4.9) involved the usage of several commercial 3D graphics libraries. At the moment, the Motion-Lab Library supports rendering with Performer from Silicon Graphics for IRIX, Vega which is distributed by Paradigm, and OpenGVS, a product of Quantum3D. Vega and OpenGVS are running mainly

on WindowsNT, but IRIX libraries are also available.

The other software parts of the system (open source or freeware) are freely available on the Internet. Mainly, the Debian GNU/Linux Distribution and ACE as a general base for the software development have to be mentioned. The Performer graphics library is also free for use on Linux. Since the freely available parts provided the base of this project, the author will consider making the Motion-Lab Library public under GNU Library General Public License with appearance of this technical report.

The Motion-Lab Library was designed and mainly implemented by the author. Under his supervision, Tobias Breuer helped with the implementation of smaller parts as documented in the source code. Documentation (apart from this technical report) is done in DOC++ which provides HTML and LaTeXversions of the C/C++ structures and class descriptions.

Before we go into the details of Motion-Lab software, the main tools and packages are introduced to the reader in case they are not known. Without some of these packages and programs, the development would have taken much more time and it might have been impossible for the author alone to complete the system and reach this high level of abstraction and perfection in the interfaces within just two years.

5.1.1 ACE - Application Communication Environment

Without ACE, most programming for different OS is much more difficult, error prone, and tiresome. System calls differ across OS in details and interface. When it comes to multi-threaded programming and socket communication, the programmer is forced to learn a lot of small differences for each and every OS. ACE, on the other hand, provides one single interface for most system calls, multi-threaded programming and socket communication. One has to learn only the specialties of ACE instead of those of 4-5 different OS. Since the realization is mostly done with inline statements[17], saving all the costs for additional function calls, ACE does not add significant overhead when the programs are run. ACE simplifies the realization of software for multiple OS concerning plain C++ code. When it comes to direct access to hardware, such as serial ports or even more special things like analog cards, ACE admittedly does not help any further.

5.1.2 sox and all the other well-sounding names

The sound is realized by a collection of ten small programs, each doing part of the job and solving one small problem. The main part for the actual replay is `sox` which works together with the kernel sound module. The program converts different sound formats into the format which can be played by the low level sound driver implemented in the kernel module. The script `play` wraps the more complicated and tiresome options of `sox` and makes the handling easier. For the replay of sound files, two more helpers allow playing a loop without sudden breaks in the sound stream. The buffering is done by `bag` and the repetitions are controlled by `repeat`. For the other application, speech synthesis, more scripts are required. Four filters modify the letter stream by speaking '@' as 'at' (done with `sed`), removing line breaks with `pipefilt`, replacing numbers with spelled-out numbers (realized with `numfilt`), and subst. abbr. w| the l. vers. they st. 4[18] (included in `preproc`). The actual translation of text with the corresponding phonemes is done in `txt2pho` so these can be pronounced by `mbrola`. Multiple speakers are available in the database for the pronunciation which allow, in addition, the usage of speed and mean frequency as independent parameters. In the end, the combination of `play` and `sox` plays the sounds for the given low-level sound driver. Both functionalities were summarized in two scripts by Michael Renner with some help of the author.

5.1.3 CVS - Concurrent Versions System

The CVS, like other version control systems, is a tool that provides a database which keeps old versions of files along with a log of all changes.

[17] Of course, only in those OS and compiler combinations which allow those statements.

[18] substituting abbreviations with the long version they stand for

It operates on the hierarchical structure of the file system containing version controlled files and directories. It enables multiple authors to edit the same files at the same time and tries to resolve conflicts, if possible. The single copy of the master source contains all information to permit the extraction of previous versions of the files at any time either by name of a symbolic revision tag or by a date in the past. There are versions of CVS available for all OS used in the Motion-Lab. All files (source, configuration, models, and documentation) are managed with CVS. Working with one master source code enables the Motion-Lab users to share latest versions and avoids keeping bugs in different versions of the same functions. Commonly used functionality therefore becomes more stable and powerful over time.

5.1.4 DOC++

The DOC++ documentation system generates both LaTeX output for high quality printouts as well as HTML output for comfortable online browsing of the source code. The program directly extracts the documentation from the C/C++ header file or Java class files. For the Motion-Lab, additional files are included in the documentation for the description of devices. The source code containing the DOC++ tags provides two versions of documentation: The complete developers documentation with all inside functions of the library and a user version as a subset that concentrates on the parts which are "seen" – that means usable – from the outside.

5.1.5 GNU-tools: gcc, gmake, emacs, and others

The famous collection of GNU-tools is the base of every Linux system. Furthermore, most of the tools are available for IRIX and some even for Windows. The compilers gcc and g++ provide useful hints when struggling with errors. Without gmake, the ACE package would not be able to compile easily. Finally, all the source code, documentation, and this technical report were written with the help of emacs. The useful aids and tools help a lot and make programming in an UNIX environment enjoyable.

5.2 Distributed programming for multiple OS

The above section gave an overview of which software was used in the development of the Motion-Lab Library and which software is still running hidden inside some scripts. However, having great libraries like ACE and powerful tools like CVS is not the complete story of how distributed programming for multiple OS becomes successful. In this section, some general guidelines introduce a more general concept than "take this and that and it will work".

5.2.1 What is the general flow of information in the project?

This question leads back to an old principle of software design in which one has to draw a lot of boxes with the information flowing between those boxes. Those diagrams help us to get the big picture of a process or program. Main sources and destinations of information need to be identified and put into the framework of informational flow. Surprisingly, programmers rarely do those drawings – but why?

Small projects tend to work out after a couple of attempts and redesign stages. Small projects grow slowly enough to forget about structure on the way, but tend to grow too fast to let real structure build up. Small projects start all over again, after reaching the point where "small" additions become more and more the purpose of a whole new project. The general mismanagement of all three assumed scenarios is the missing general flow of information, which should guide the project from the beginning. Therefore, one has to start with the question:

5.2.2 Where does the information come from and where should it go to?

If at least the points where information comes from and where it should go to are known, the parts in between very often become trivial. It is reduced to the question of conversion, recalculation, and managing information. Sometimes, it looks like the information is necessary at all points at once. In a distributed network, but also in a single computer, this leads to enormous data transfer which will slow the system down. The anal-

ysis should focus on the question of whether the complete set of data is necessary or if are there points where a smaller part would be sufficient. Following this question, one could come up with a structure of necessary data pools which might divide the project into different units/parts. This structure might be different from the one obtained from analysis of the source and destination of information.

5.2.3 How do special OS/library requirements split up the project?

Yet another structure might become clearer by looking into the need for special libraries or hardware which are involved in the project. As mentioned in the hardware section of this report, special devices often come with special libraries. Since most of those libraries are not binary compatible, they have to be used on the given OS platform or re-implemented on a different one. The re-implementation very often is made impossible by the manufacturer for reasons of fear: They fear the loss of knowledge and their position on the market by making interfaces documented and open to public[19]. Even if the interface is well documented, it will take a while to re-implement everything from scratch. Therefore, it is generally easier to take a look into the given libraries and design the structure of the project in parts around those libraries. The client-server concept (see section 5.3) provides an additional layer of abstraction.

5.2.4 Is it necessary to run the parts under several OS and on different computers?

One is lucky, if one can answer this question with a "*no*". On the other hand: Use the opportunity to stay flexible and gain the advantages of independence! A system which is based on parts that do not care about system requirements is more likely to be used by a lot of people. Not only the power of multiple users working on the same project, but also the power of parallelly working machines should be persuasive. The disadvantage of missing load-balancing in a situation where one computer does one job could be overcome by having several computers doing similar jobs and distributing the work load among them. Being able to use multiple computers for the same job, decreases the probability that none of the computers work.

5.2.5 Combine all the above structures to come up with a plan!

In the process of planing a new project that involves different hardware and software requirements, the above thoughts might help one come up with different solutions for the future structure. Depending on the importance of different requirements for a given project, one has to combine the results from the above questions. It could actually help to design the project in multiple ways and join those parts into a final concept which share the common structure of multiple solutions. With this approach one can almost be sure not to miss an important part which becomes obvious once started with the implementation.

5.3 Client - server architecture

The communication, library, and hardware structure of the equipment in the Motion-Lab suggested a client - server architecture based on the following requirements. Some of the libraries were explicitly designed for one OS and therefore, at least three different OS had to be used (Maxcue low-level access on Windows95, graphics like Vega/OpenGVS on WindowsNT and the driving dynamics[20] on IRIX). The control proposed by Motionbase for the Maxcue motion platform is based on UDP broadcast calls, which were unacceptable in our local network for two reasons. First and most import, the protocol had no security check and anybody could simply add data to the stream by opening a telnet like program on the specific port with the result of unpredictable behavior of the platform. The other reason was to reduce overall traffic and not to disturb others by using the platform inside our local network. In the

[19] As the development of graphics support in the Linux community has shown, the manufacturers might very well profit from the community by giving the interface to ambitious programmers and participate in distribution of their products.

[20] The driving dynamics done by Müller System Technik is not part of this technical report, but had to be considered in the design.

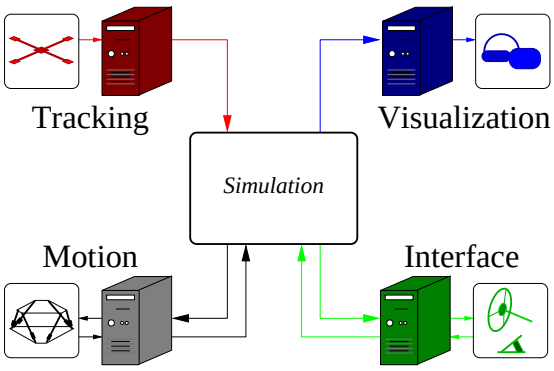


Figure 18: The general client - server framework for a distributed VR simulation.

process of solving the security issues, it became clear that a constantly running server controlling the platform and accepting only authorized connections, one connection at a time, would enforce a more deterministic behavior and control for security risks. Similar requirements had to be considered for the control of the steering wheel, since subjects in the experiments directly interact with the device and should not face any kind of risk. Even in cases were the VR simulation fails due to errors in the program, the server can still run and guarantee a safe shutdown of the device.

The idea of server programs controlling the different input and output devices is powerful not only for security issues. It also forces the implementation of a strict layer of abstraction between a client who wants to have some data from a device or send it towards the device, and the controlling server that implements the specific input and output behavior. Having this abstraction layer established not only for the platform and steering wheel, but also for all the other devices like joystick, tracker and visual display, the simulation logic becomes quite independent of special hardware and the actual physical realization. It is now possible to exchange the graphical rendering without affecting the simulation of a virtual observer, which is quite unusual for most VR systems. The advantage of this became clear when the actual machines were exchanged and we had to move from the Vega libraries to the OpenGVS rendering. The main simulation was not subject to any changes, only the graphical rendering had to be re-implemented based on the new library. This example illustrates the flexibility of this approach enabling the integration of new hardware, which very often comes with specialized libraries. The server implements the abstract device layer, based on the interface of the specific physical realization, and the clients just stay the same. This enables multiple users to program for the abstract interface rather than for a specific device, which enhances the overall usability of software. Even changes in the OS or the physical device do not affect the VR simulation.

It also became easier to add new services for all simulations. Since the main part, the direct handling of the devices, is outside the actual VR simulation of the user, the changes in the simulation program necessary in order to use additional services are reduced to a minimum[21]. Taking the client server concept together with asynchronous communication, it is easy to have devices work at different update rates. Each device can work at its necessary speed and provide either local feedback (like the steering wheel) or global feedback (like the visual display reacting on a turn of the head). The information flow is depicted in Fig. 18 for a simple VR simulation based on the client - server concept.

5.4 Use of templates

Having introduced the client - server architecture, it becomes obvious where one should use different classes and which functionality should be shared between different streams of information. In order to make the communication between the client and the server as simple and efficient as possible, one step for confining the communication was taken. The data packages themselves should contain only similar things in sense of data type lengths. The allowed types for the data inside data packages were fixed to `long int` and `double`, both containing four bytes and sharing the same conversion for changes between processor types[22].

[21] For example, it was possible to add head tracking into a simulation by adding less than 20 lines of C++ code into a normal simulation.

[22] see differences of big and small endians in the SGI ABI or the ntoh* man pages.

All those points do not connect to template discussions at first glance. Templates in C++ enable programming for unknown types to some degree. The concept is nearly as powerful as libraries are for sharing functionality for known types. It allows implementation of general functions in a type safe way at compile time[23]. In combination with inheritance, it becomes powerful, as the reader can see in section 5.6.2 where the realization of the different devices clients is discussed. Another example of efficient usage of templates can be seen in the stack implementation[24].

Connecting both ideas, it became fast and easy to send data packages over a TCP/IP connection with the above restrictions. General template functions were used to pack and unpack data packages and store the data on stacks. Since all data could be handled in the same way, there were no functions necessary for extracting special things like pointers of strings. Avoiding dynamic memory calls made the implementation efficient and safe to memory leaks. Extending the known data types with additional variables is easy since only initialization and print routines have to be changed due to their additional text output naming the data; the data communication routines stay untouched.

5.5 What is real-time?

Defining "real-time" is highly dependent on purpose and context. "Only reality runs at real-time" could be one statement, which could be opposed by some consideration about the brain of a bee and a CRAY high end computer with eight processors. Both the bee and the computer have roughly 10^6 neurons or transistors respectively. For the bee one "operation" is quite slow with $10^{-3}s$ compared to $6*10^{-9}s$ for the computer. Nonetheless, due to the high parallel execution in the bee's brain, the bee theoretically executes 10^{12} operations per second while the CRAY stays behind with only 10^{10} operations per second. What is real time for the bee and the CRAY? The bee behaves in the world with a considerably high latency, particularly if one compares it to the possible high precision of the CRAY. However, the speed of the bee's reaction is sufficient to do navigation, pattern recognition, social interaction and more specialized things. The CRAY computer is faster for very specific and simplified tasks but could not reach the complexity of the bee's perception and interaction with the world. This consideration holds for humans as well as for the bee. Looking into different modalities, the latency and time accuracy of perception differs widely. The time lag which would be acceptable, that means which would not be noticeable as additional offset to the "true" point in time, is different from the overall time resolution in each modality. Programming a VR setup, where things should behave "normally" and allow us to perform without previous training, has to consider both time constrains: The update rate and the latency. The goal is therefore not to define real-time in yet another way, but to provide sufficient fast interaction in the sense of update rate and latency.

A number of studies have shown either the neural latencies of the human system or the necessary update rate which should be provided by a simulation. For example, Dell'Osso and Daroff (1990) refer to several interactions between the vestibular and visual system. The latency for head movements which result in the vestibulo-ocular reflex appears to be less than 15 ms. The same study specifies that the latency for eye stabilization control is larger than 100 ms for field motion and more than 125 ms for corrective saccades for positional errors. The latency for extraction of optic flow seams to be greater than 300 ms (Berg, 1999). The eye stabilization mechanisms therefore react to changes in the vestibular system much quicker than to the perceived visual stimulus. The effect of apparent motion is visible for image changes faster than 18 Hz. Normally, update rates of 25 to 30 Hz are used for computer rendered pictures. The accuracy of audio localization also depends on the frequency of the provided stimulus. King and Oldfield (1997) described the necessary spectrum of the sound stimulus: "Results show that broadband signals encompassing frequencies from 0 to (at least) 13 kHz are required in order for listeners to accurately localize

[23] This is not true for void pointer concepts, which test class membership during runtime.

[24] The stack implementation is special for not keeping everything on the stack, as the concept normally suggests.

signals actually presented from a range of spatial locations". The human skin is known to be sensitive for vibrations faster than 2 kHz. For vestibular stimulation, update rates of 250 Hz were used to create a smooth path (Berthoz, Israël, Georgesfrancois, Grasso, & Tsuzuku, 1995). VR simulations have to match these numbers, and provide sufficiently fast update rates, and react to changes like the turn of the head with minimal latency. In addition, it is important to keep the time offset between stimuli presented to different modalities sufficiently small in order to not disturb the natural integration process.

5.6 Motion-Lab Library

The realization of the above concepts in the frame of a technical report will stay away from printing pages and pages of source code. The general implementational details are given without referring to actual code. The code, the actual data structures, and the respective documentation can be found at the author' website. All the actual C++ code compiles on all five OS platforms involved, as long as it does not concern special hardware libraries.

The overall structure of the Motion-Lab Library is simple. The library provides clients classes for all devices in the Motion-Lab. Those clients control the communication to the device servers, making the communication transparent to the user. In addition, some useful tools and functions for matrix and vector handling, and keyboard control are provided, but not discussed here in detail.

5.6.1 Stacks

Generally a stack is considered to be a structure which keeps information in a "last in, first out" (LIFO) fashion. The stack for the Motion-Lab communication works exactly with this principle, but with one unusual addition to it. There is only one "last" for output available and packages pushed earlier will be lost. In exchange, the stacks provide information about how often the last record had actually been popped. Of course, a template class is used for the realization of this concept, since the data are not touched at any point in the stack. In addition, the implementation is thread-safe for multiple readers and one writer without locking write or read access. Writing processes add new information and simultaneous reading processes are guaranteed to get the most recent data available. Internally, the stack class uses a ring structure for the storage of information.

This behavior is required for one simple reason. The asynchronous communication explicitly demands high speeds which may include dropping of whole packages. Locking would reduce efficiency, and always providing the newest package reduces latencies in the system. A queue concept would force the accepting process in a communication to check whether there is more information waiting to be processed. The stack concept on the contrary provides the most recent information available and drops other packages without the risk of increasing queue length.

5.6.2 Communication & devices

The communication itself is hidden (as computer scientists like to say, "transparent") to the user. It involves a collection of templates realizing the *send* and *receive* as *client* and *server* via a TCP/IP socket. Gathered as a virtual device, those templates implement the transparent communication with one specific server. As a result, the actual implementation of the abstract device for the user is done in the library by defining the data structure for the communication for one type of client. Therefore, the user does not have to worry about the actual template usage or class instantiation for the communication. The user is addressing a device by instantiation of the specific client for this device. Naturally, the server for that device has to run beforehand[25].

5.6.3 Sound scripts & functions

There are two simple scripts enabling the use of sound in programs running on Linux machines in the Motion-Lab. The corresponding functions for C++ programs are available in the Motion-Lab Library. There is not yet a sound server which could be used for programs running on different machines and OS. In general, the sound is realized

[25] Since those servers can run all the time, one should consider running those as daemons starting at normal system startup.

by a collection of small tools (see page 19) which are available for free on the Internet.

- `ml_play`: Replays all well known sound formats either as an infinite loop or a given number of times. Internally, the sound is buffered for the loop to guarantee break free replay even under high load of the computer.

- `ml_say`: Speaks a given string with German pronunciation. The string could include numbers and abbreviations. The speech synthesis is done online, so status messages or other feedback could be given to the user.

5.6.4 Device servers

At the moment, there are several device servers available, some running on multiple OS platforms, but most of them are confined to a specific one due to hardware and library constellations. In general, it makes sense to run the server on one specific machine, since the server has to connect to some physical device. Different ways of connecting the actual device are handled inside those servers. Additional servers can easily be implemented by using the existing servers as examples.

- platform:
  The control for the vestibular simulation runs on Windows95 and connects to an analog card with specialized library from the manufacturer.

- hmdvega, hmdperformer, and hmdgvs:
  The visual simulation runs on WindowsNT, Linux or IRIX and displays the rendered 3D model via the normal graphics output. On WindowsNT and Linux the output of the graphics card can be connected to the HMD. Stereo vision is possible by using two machines which synchronize their output.

- wheel:
  The force feedback steering wheel control runs on Linux with the help of an analog card and a library from the LinuxLabProject.

- joystick:
  It is currently possible for Linux to control up to two joysticks at a time via the gameport of one sound card.

- tracker:
  The tracker server reads out the serial port of a Linux machine and interprets the binary or ASCII output.

5.7 Applications

Several applications run in the Motion-Lab with different goals. A simple driving demo will be presented after explaining the concept of the virtual observer for VR simulations based on the client - server architecture. In addition, two experimental programs are sketched to give an impression of how things might look for open- and closed-loop programs. Nonetheless, the general structure of all programs is nearly identical since the tasks are quite similar.

The overall structure works around a central simulation loop as depicted in Fig. 19. The input devices provide information which is used to update the status of the virtual observer. Based on a time difference Δt in the main simulation loop the observer's movement status is updated describing a discrete version of what happens in the real world (velocity: $V_{t+1} = V_t + \Delta t * A_t$ and position: $P_{t+1} = P_t + \Delta t * V_{t+1}$). Naturally, one could add friction, wind resistance, surface slant and other factors here to slow down or accelerate the observer, based on the model of the world. For simplicity, those factors are assumed to slow down the observer (like sliding on a flat horizontal plain) a little bit and change, therefore, the velocity by a damping factor. After having updated the observer's internal status, the output devices get new data from the simulation. At that point, we can have a short time delay before we start all over again, in order to control the speed/update rate of this simulation loop.

5.7.1 Simple driving demo

The driving demo incorporats joystick and/or steering wheel control. Therefore, most of the above described equipment is actually involved in this simulation (see Fig 20). The car dynamics are kept as simple as possible, since it should soon be replaced by the professional one as mentioned earlier in this report. However, it is just a demo and not used for experiments in its current state. The general structure is exactly as described

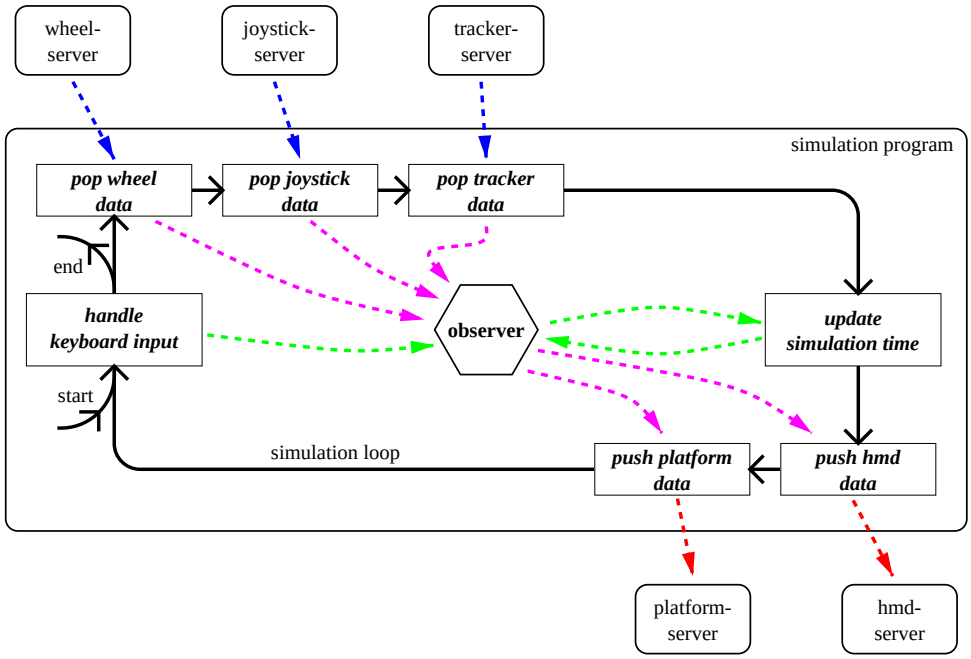


Figure 19: The general feed forward simulation loop shows the general input in the upper left-hand corner and the output in the lower right-hand corner. The virtual observer in the middle gets influenced by the input from steering wheel or joystick and the head tracker. Based on the internal representation of the observer's movement status output data are generated for the platform and visualization.

above. The virtual observer gets input from the different input devices and sends commands to the visualization (HMD server) and the vestibular simulation (platform server). The data for the visual simulation are the actual position and velocity of the observer. In contrast, the data sent to the platform could not be transformed into a movement of the actual distances in meters, but had to be reduced or scaled (see "motion cueing" in section 3.1). One simple solution is to send the actual speed of the observer as positional data for forward movements and negative pitch. If the observer gains speed, the platform pitches backwards to substitute for some of the linear forward accelerations. As a cue for higher velocity, vibrations are simulated with increasing amplitude. The combination of both results in a quite realistic feeling for driving without sudden changes in velocity. Since the platform can only rotate a certain angle, simulated turns have to incorporate the same principle. Adding some roll motion to simulate tangential forces enhances the realistic feeling. Nonetheless, psychophysical tests have to be performed to match data from real drives to simulated ones and to equalize both at the perceptual level.

Figure 20: Driving setup with force feedback steering wheel.

5.7.2 Experiment 1: Distance, velocity and acceleration judgements

Before we can start to program realistic driving simulations on the motion platform, we have to know which parameters are actually perceived by humans sitting on the platform. Therefore, an experiment was designed focusing on the question of perceptual parameters (see von der Heyde,

Riecke, Cunningham, and Bülthoff (2000) for detailed experimental design). Concentrating here on the software issues, it should be mentioned that this experiment was explicitly done with an open loop paradigm. The subjects had no influence on the performed movement, but had to report verbally on their perception of distance, velocity and acceleration. The simulation was, therefore, independent of user's input. The movements were predefined with Gaussian shaped velocity profiles by a parameter setting the controlling distance and the maximum acceleration. The positions for the movement were calculated beforehand and played back with the appropriate timing. This program shows that the client server concept can also be used in feed forward (open loop) conditions where no interaction is required.

5.7.3 Experiment 2: Holding balance, and coding vestibular and visual heading changes

The second experiment involves, in contrast to the first one, continuous feedback from the subject. It therefore uses a closed loop paradigm. All the details are described in von der Heyde (2001). Concerning the program itself, the structure is in principle quite similar to the driving demo. Because of the experimental conditions and different stages during the experiment, the conditions had to be scheduled based on the performance of the subject. Parameter files describe thresholds and dynamic changes in the level of difficulty for the task. The task itself was to stabilize the platform for roll movements based on vestibular cues. The visual simulation was not providing any roll information. The disturbances increase in speed and amplitude for higher levels of difficulty. In the end, the subjects had actually performed heading changes based on the paths they had learned. The relationship between visual and vestibular heading change was controlled and changed only in the test condition. Therefore it became possible to ask whether the vestibular or the visual turns where encoded in the learning stage. Changing those relationships is easily achieved in VR and would otherwise be very difficult to perform.

6 Summary of system characterization

In sum, the Motion-Lab implements a VR setup that enables psychophysical experiments with a variety of hardware in order to simulate multiple sensory inputs. The complex simulation is distributed across a network of specialized computers enabling input and output for intense interaction. Different experimental paradigms can easily be implemented with the Motion-Lab Library which effectively hides from the programmer problems that are imposed by the distributed systems approach.

The lab uses a network of standard PCs extended by special VR equipment. The different units of the networks are exchangeable and share multiple resources. The system can therefore be classified as distributed system. The general architecture realizes a client/server approach in which each hardware device is managed by a specialized server. The servers implement abstract devices, which efficiently hides the differences of various connected hardware. The multi-threaded library provides the clients for the abstract device interfaces. Therefore, these clients connect to the servers, hidden from the VR application programmers view. The bidirectional communication is asynchronous and done via TCP/IP with low latency. Specialized stacks circumvent problems with different frequencies which are imposed on the system by the demands of different sensory modalities. Smoothness of the data stream is established where needed by inter- or extrapolation methods. The library is available and used for multiple OS, hiding again OS specific interface differences of Windows95/NT, IRIX and Linux. Distributed development techniques were used for concurrent access by multiple programmers.

VR simulations are bound to include multiple senses. The goal typically is an immersive simulation which allows the user to interact with the environment with acceptable latency and high degree of realism. The observer's movements are usually unnaturally done in the simulation. Due to the lack of spatial updating, users do not feel that they are moving in space. In contrast, the present lab realizes realistic and immersive simulations for multiple senses. The latency between an ac-

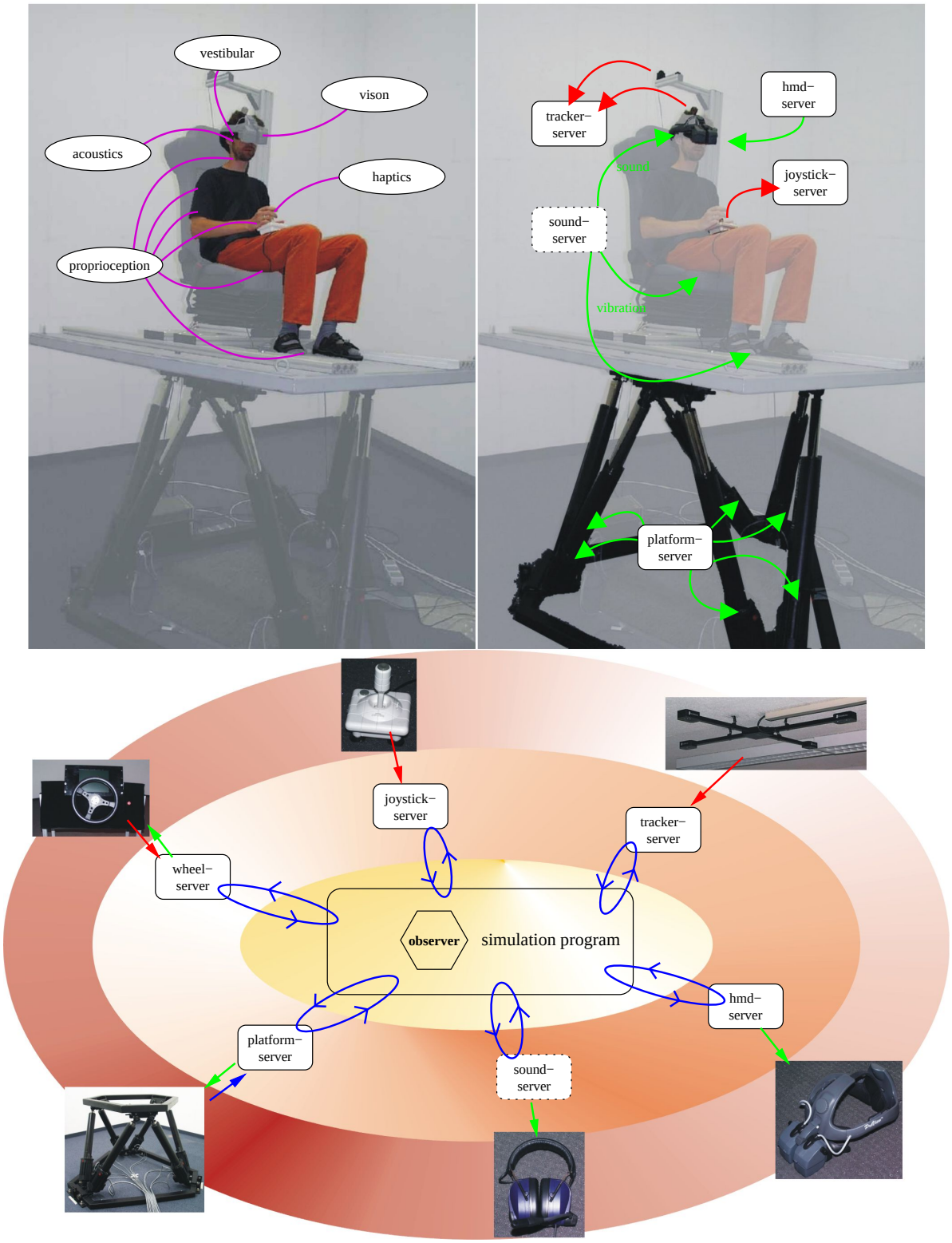


Figure 21: Overview of the Motion-Lab (see description in the text).

tion of a user and the feedback of the system is reduced by multiple layers of feedback loops within and between modalities. Specifically, vestibular, visual, acoustic, vibration, and haptic stimuli can jointly be used by the applications:

- The **vestibular** stimulation is realized by a six degree of freedom (DOF) motion platform with can perform high accelerations;
- Stereo **vision** is presented via a head mounted display with high resolution realized by different graphic libraries rendering the virtual scenery;
- **Acoustic** simulation is not yet presented in 3D, but already includes synthetic speech generation for multiple speakers and numerous stereo sound effects;
- The sound generation can be employed for low frequencies **vibration** stimulation;
- The simulation of **haptic** force feedback for the steering wheel delivers a realistic experience for driving simulations.

Other input devices can be used for typical VR interaction. Joysticks allow analog multiaxes control of quantities coupled to immediate changes in the virtual environment. Trackers for six DOF are used for the input of pointing movements as well as head tracking for the control of the virtual camera.

Figure 21 summarizes the client server architecture of the VR simulation in the Motion-Lab. The top part of the diagram shows the equipment of the lab as well as the senses of the human observer that are involved. The lower part depicts the implementing architecture on three levels: The inner software level of the VR simulation of the virtual observer, the outer hardware level of the VR equipment and the level in between formed by the distributed device servers implementing the abstract layer of I/O devices. The interaction of observer and virtual observer is realized by multiple feedback loops connecting different levels. The human observer perceives the virtual environment through multiple senses and interacts with the simulation via tracking, joysticks and the steering wheel. The virtual observer sends and receives data from the devices and simulates the VR environment as a discrete version of world. In sum, the Motion-Lab is usable for closed and open-loop experiments. Therefore, various psychophysical experiments in VR were made possible by this distributed system.

Acknowledgement

The author would like to thank Heinrich Bülthoff for his generous support at the institute. He is grateful to Bernhard Riecke and Douglas Cunningham for long fruitful discussions. Last but not least special thanks to Stephan Braun and Michael Renner for intense technical support.

References

Arthur, K. W. (2000). *Effects of field of view on performance with head-mounted displays.* Unpublished doctoral dissertation, Department of Computer Science, University of North Carolina, Chapel Hill. ([Online] Available: http://www.cs.unc.edu/ arthur/diss/)

Beall, A. C., & Loomis, J. M. (1997). Optic flow and visual analysis of the base-to-final turn. *Int. J. Aviat. Psychol.*, *7*(3), 201 – 223.

Begault, D. R. (1994). *3-d sound for virtual reality and multimedia.* Boston: Academic Press Professional.

Berg, A. V. van den. (1999). Predicting the present direction of heading. *Vision Res.*, *39*(21), 3608 – 3620.

Berthoz, A., Israël, I., Georgesfrancois, P., Grasso, R., & Tsuzuku, T. (1995). Spatial memory of body linear displacement: What is being stored? *Science*, *269*(5220), 95–98.

Bülthoff, H. H., Riecke, B. E., & Veen, H. A. H. C. van. (2000). Do we really need vestibular and proprioceptive cues for homing. *Invest. Ophthalmol. Vis. Sci.*, *41*(4), 225B225.

Burdea, G. (1993). Virtual Reality Systems and Applications. In *Electro'93 international conference* (p. 164 pp). Edison, NJ.

Burdea, G., & Coiffet, P. (1994). *Virtual Reality Technology.* New York: John Wiley & Sons. Inc.

Cobb, S. V. G., Nichols, S., Ramsey, A., & Wilson, J. R. (1999). Virtual reality-induced symptoms and effects (vrise). *Presence: Teleoperators & Virtual Environments, 8*(2), 169 – 186.

Cunningham, D. W., von der Heyde, M., & Bülthoff, H. H. (2000a). Learning to drive with delayed visual feedback. In H. Bülthoff, M. Fahle, K. Gegenfurtner, & H. Mallot (Eds.), *Beiträge der 3. tübinger wahrnehmungskonferenz* (p. 164). Max-Planck-Institute for Biological Cybernetics, Germany: Knirsch Verlag, Kirchentellinsfurt, Germany.

Cunningham, D. W., von der Heyde, M., & Bülthoff, H. H. (2000b). Learning to drive with delayed visual feedback. *Invest. Ophthalmol. Vis. Sci., 41*(4), S48.

Dell'Osso, L. F., & Daroff, R. B. (1990). Eye movement characteristics and recording techniques. In J. S. Glaser (Ed.), *Neuro-ophthalmology* (2 ed., pp. 279 – 297). Philadelphia, PA: J. B. Lippincott Company.

Fichter, E. F. (1986). A stewart platform-based manipulator: General theory and practical construction. *The International Journal of Robotics Research, 5*(2), 157 – 182.

Gilkey, R., & Weisenberger, J. (1995). The sense of presence for the suddenly deafened adult: Implications for virtual environments. *Presence: Teleoperators & Virtual Environments, 4*(4), 357 – 363.

Heilig, M. (1960, oct). *Stereoscopic-Television Apparatus for Individual Use.* US Patent No. 2,955,156.

Hendrix, C., & Barfield, W. (1996). The sense of presence within auditory virtual environments. *Presence: Teleoperators & Virtual Environments, 5*(3), 290–301.

Hlavacka, F., Mergner, T., & Bolha, B. (1996). Human self-motion perception during translatory vestibular and proprioceptive stimulation. *Neurosci. Lett., 210*(2), 83 – 86.

King, R., & Oldfield, S. (1997). The impact of signal bandwidth on auditory localization: Implications for the design of three-dimensional audio displays. *HUMAN-FACTORS, 39*(2), 287 – 295.

Mergner, T., & Rosemeier, T. (1998). Interaction of vestibular, somatosensory and visual signals for postural control and motion perception under terrestrial and microgravity conditions - a conceptual model. *Brain Res. Rev., 28*(1-2), 118 – 135.

Mergner, T., Siebold, C., Schweigart, G., & Becker, W. (1991). Human perception of horizontal trunk and head rotation in space during vestibular and neck stimulation. *Exp. Brain Res., 85*(2), 389 – 404.

Riecke, B. E. (1998). *Untersuchung des menschlichen Navigationsverhaltens anhand von Heimfindeexperimenten in virtuellen Umgebungen.* Unpublished master's thesis, Eberhard-Karls-Universität Tübingen, Fakultät für Physik.

Ryan, M. D., & Sharkey, P. M. (1998). Distortion in distributed virtual environments. In J.-C. Heudin (Ed.), *Virtual worlds 98* (Vol. 1434, pp. 42 – 48). Berlin – Heidelberg – New York: Springer-Verlag.

Viirre, E. (1996). Virtual reality and the vestibular apparatus. *IEEE Eng. Med. Biol. Mag., 15*(2), 41 – 44.

von der Heyde, M. (2001). *A distributed virtual reality system for spatial updating: Concepts, implementation, and experiments.* Unpublished doctoral dissertation, Universität Bielefeld – Technische Fakultät, Max-Planck-Institute for Biological Cybernetics, Germany. (in press)

von der Heyde, M., & Häger-Ross, C. (1998). Psychophysical experiments in a complex virtual environment. In D. J. K. Salisbury & D. M. A. Srinivasan (Eds.), *Proceedings of the third phantom users group workshop, mit artificial intelligence report no. 1643, mit r.l.e. tr no.624* (pp. 101 – 104). Cambridge: MIT Press.

von der Heyde, M., Riecke, B. E., Cunningham, D. W., & Bülthoff, H. H. (2000). Humans can extract distance and velocity from vestibular perceived acceleration. *J. Cogn. Neurosci.*, *1*, 77.

Witmer, B. G., & Singer, M. J. (1998). Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators & Virtual Environments*, *7*(3), 225–240.