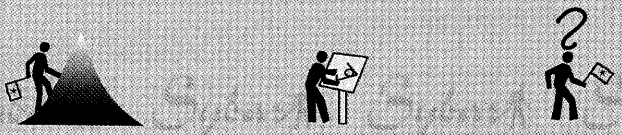
Sybar, a human motion analysis system for rehabilition medicine

Sybar

A Human Motion Analysis System for Rehabilitation Medicine



Edwin Hautus

Sybar,
a Human Motion Analysis System for Rehabilitation Medicine

Proefontwerp

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. M. Rem,
voor een commissie aangewezen door
het College voor Promoties
in het openbaar te verdedigen
op dinsdag 25 november 1997 om 16.00 uur

door

Edwin Hautus

geboren te Eindhoven

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. D.K. Hammer
en
prof.dr.ir. K. Kopinga

Copromotor:
dr.ir. C.WA.M. van Overveld

Druk: Universiteitsdrukkerij TU Eindhoven

Summary

The Sybar project is a designer's Ph.D project that deals with the development of a motion-analysis system for rehabilitation medicine, at the VU Hospital in Amsterdam.

Human motion can be analyzed by biomechanical measurement systems. There are a number of different methods to generate several types of data on human motion. Biomechanical analysis of patients with motion disorders can potentially give valuable information to physicians. However, existing biomechanical measurement systems that have been introduced in a clinical setting have had only limited success, because they have been designed for other purposes.

The goal of the Sybar project is to build a biomechanical analysis system specific for the clinical setting. We take the clinically accepted way of human observation as a starting point. Instead of providing large amounts of measurement data separately, the existing image of the patient is enhanced with additional information. We believe that by showing a video of a patient in combination with graphics measurement results, it becomes much easier to directly relate the data to the actions of the subject under investigation.

Sybar is developed using object oriented methods. In particular, the Object Modeling Technique (OMT) is used. From a software-engineering point of view, interesting aspects are:

- a clear system specification can not be determined without some form of rapid prototyping.
- the environment of the system is complex, and dynamic.
- the requirements for digital video and image processing are on the edge of what is technically feasible.

Summary 3

CHAPTER 1 Introduction 9

1.1 Sybar 9

1.2 Setup of this thesis 11

1.3 The chapter structure 13

CHAPTER 2 Human Motion Analysis 17

2.1 Short history of human motion analysis 17

2.2 Current clinical measurement methods 19

2.3 Problem statement: measurement methods for patient assessment 21

2.4 Our approach 23

CHAPTER 3 Requirements Definition 25

3.1 Function of the requirements definitions 25

3.2 Clear presentation of measurement data 25

3.3 User friendliness 26

3.4 Patient friendliness 27

3.5 Reliability 27

3.6 Maintainability 28

3.7 Resource constraints 28

CHAPTER 4 Requirements Specification 31

4.1 Function of the requirements specification 31

4.2 System boundaries 31

4.3 Types of measurements that are supported 32

4.4 The physician's view of the system 33

4.5 Maintainability requirements 39

4.6 Performance requirements 40

CHAPTER 5 Analysis Model 41

5.1 Function of the analysis model 41

5.2 The object model 41
5.3 The dynamic model 45
5.4 The functional model 46

CHAPTER 6 **Toplevel Design 47**
6.1 Introduction 47
6.2 Decomposition of Sybar 47
6.3 The measurement system 48
6.4 The computer system 50
6.5 Sybar software 51

CHAPTER 7 **Recording Subsystem 55**
7.1 System design 55
7.2 Object design 58

CHAPTER 8 **Display Subsystem 65**
8.1 System design 65
8.2 Object design 69

CHAPTER 9 **Kinematics: Systems and Algorithms 81**
9.1 Introduction 81
9.2 Modeling human motion 81
9.3 Detection of kinematics 83
9.4 Detection devices 83
9.5 Detection device for Sybar 86
9.6 Computer vision basics 86
9.7 General issues on retrieving human motion from images 89
9.8 Approaches without markers 90
9.9 Approaches that use markers 92
9.10 Classification of approaches 96
9.11 Approach for Sybar 97

CHAPTER 10 Kinematics from a Single View 99

10.1 Introduction 99

10.2 Chen and Lee's approach to 3D reconstruction 100

10.3 Problems with Chen and Lee's approach 102

10.4 Reconstruction using a linearized set of equations 104

10.5 Solving the equations using singular value decomposition 111

10.6 Results of our approach 114

10.7 Conclusion 121

CHAPTER 11 Kinematics Subsystem 123

11.1 Introduction 123

11.2 Kinematics subsystem design 123

11.3 The feature detector 126

11.4 The human model 132

11.5 Tracker 136

CHAPTER 12 Design Methodology 143

12.1 Design & design methods 143

12.2 Software design methods 145

12.3 Choosing a toolset 147

12.4 Toolset for Sybar 147

12.5 Design strategy 149

CHAPTER 13 Design Process 153

13.1 Subprocesses of the design process 153

13.2 People 159

CHAPTER 14 Evaluation 161

14.1 Evaluation of Sybar 161

14.2 Evaluation of the main goal: clinical usefulness 161

14.3 Evaluation of the system with respect to the requirement definition 163

14.4 Evaluation of the design methods 164

Object Oriented Methods & OMT 169

1.1 A very short introduction to object oriented methods 169

1.2 The object modeling technique 171

1.3 Two small extensions of OMT 174

Hardware Implementation 177

2.1 Measurement devices 177

2.2 Synchronization 177

2.3 Video recorder 178

2.4 Video digitizing and hardware compression 178

2.5 Data acquisition 180

2.6 List of devices and overview 180

Bibliography 183

Acknowledgments 191

Samenvatting 193

Curriculum Vitae 195

CHAPTER 1

Introduction

This chapter gives an introduction to the Sybar project, a designer's Ph.D. project for the Eindhoven University of Technology.

1.1 Sybar

A designer's Ph.D. is a post-graduate doctorate program intended for second phase design course graduates. In contrast with traditional Ph.D. projects, the goal of a designers Ph.D. project is not to do research which is relevant to the scientific community, but to prove one's abilities in developing a design and managing the design process.

The Sybar project deals with the development of a motion-analysis system for rehabilitation medicine at the VU Hospital in Amsterdam. The project took place at the department of Clinical Physics & Engineering of the VU Hospital. A number of employees of the VU Hospital participated in the project. Sybar is an acronym for 'Systeem voor bewegingsanalyse bij revalidatie', which is dutch for 'system for motion analysis in rehabilitation medicine'

1.1.1 Purpose

Human motion can be analyzed by biomechanical measurement systems. There are a number of different methods to generate sev-

eral types of data on human motion. Furthermore, there are many disorders that effect the human ability to move and that require treatment. Biomechanical analysis of these patients can potentially give valuable information to physicians. However, existing biomechanical measurement systems that have been introduced in a clinical setting, have had only limited success, because they have been designed for other purposes.

The goal of the Sybar project was to build a biomechanical analysis system specific for the clinical setting. The novelty to the approach is that the biomechanical measurements are seen as an enhancement of the normal working procedure of the physicians, and not as a completely separate procedure.

From a software-engineering point of view, interesting aspects are:

- a clear system specification can not be determined without some form of rapid prototyping.
- the environment of the system is complex, and dynamic.
- the requirements for digital video and image processing are on the edge of what is technically feasible.

1.1.2 Design description and process

A problem with describing the design is that there is no general consensus in computer science on what a software design actually *is*. However, there are a number of design methods that try to structure the design process. One of these methods, the Object Modeling Technique (OMT), is used as a basis for the design description in this thesis. A short introduction to OMT is given in Appendix A.

A characteristic of design-method books is that they often describe the design process in an idealized way. In practice, there is always a structured side and a creative side of the design process. The problem with creativity is that it is nearly impossible to control. A design process therefore in practice never completely follows a method. The Sybar project is no exception. OMT was used as a design language, however, the design process was often driven by creativity and intuition.

1.2 Setup of this thesis

The description of the design is separated into two parts: *analysis* and *design*. The analysis part describes the desired behavior of the system. The design part describes the way to achieve this desired behavior. Describing a design without giving the analysis does not make much sense, since there is then no way to verify that the design meets the requirements.

One of the purposes of a Ph.D. by design is to gain insight into the design processes of the engineering disciplines involved. For this reason, this thesis does not only describe the design itself, but also the process that leads to the design.

1.2.1 Analysis

The description of the analysis is separated in a number of parts. Each part describes the requirements of the system on a certain level of detail. Going from one level to the next, involves design activities. For example, to get from the requirements definition to the requirements specification, involves the design of the user interface. The description of the user interface is in itself a requirement for the design part.

We distinguish the following parts in the description of the analysis:

- *Domain analysis:* background information on the project
- *Problem statement*: the specific problem that is the purpose for building the system.
- *Initial approach*: the initial approach towards solving the problem.
- *Requirements definition*: a statement describing in global terms what the system is expected to provide.
- *Requirements specification*: a more detailed overview of the requirements, still using natural language.
- *Analysis model*: a model of the system, as it is required to function in its environment.

Although the design method OMT recognizes the top down approach in the design phase, it does not distinguish the subphases in the analysis phase. We think that requirements engineering is

such an important part of Sybar, that it requires a number of levels of detail to be described. We feel however, that this set-up does not deviate fundamentally from the suggested software life-cycle described in OMT. The terminology used for the analysis subdivision is taken from [69]

1.2.2 Design.

We distinguish the following parts in the design description:

- *Top level design*: global design of the system and the division of the system into subsystems.
- *System design*: design of each of the subsystems.
- *Object design*: design of the individual classes of a subsystem.

This setup follows the suggested setup of OMT.

Quite often in software design, the hardware configuration that the software is supposed to run on, is part of the requirements. However, in our case, determining the hardware configuration was an integrated part of the design task. Sybar can therefore be seen as a combined software and hardware design with a strong emphasis on software design. The overall hardware design is one of the first steps in the toplevel design. However, there are also hardware aspects to certain subsystems.

1.2.3 Process and evaluation

The design process is described in three parts:

- *Design methodology*: describes the selection of a design method for Sybar.
- *Design process*: describes the design process that led to the design.
- *Evaluation*: gives an evaluation of both the design and the design process.

1.3 The chapter structure

The chapter structure of the thesis is based on the setup described in previous section. The domain analysis, problem statement and initial approach, are described in a single chapter on *human motion analysis*. The system design and object design of the sub systems are described in a single chapter for each of the subsystems. The description of the kinematics subsystem is divided into three chapters.

In the following table, we give an overview of the contents of the chapters in this thesis.

TABLE 1 Guide to the chapters

	Chapter	Description
1	Introduction	This chapter gives an introduction to the Sybar project, a designer's Ph.D. project for the Eindhoven University of Technology. At this moment, you are reading this chapter.
Part 1: Analysis		
2	Human Motion Analysis	The first human motion analysis systems date back to the previous century. Over the years, new methods to measure and analyse human motion have been introduced. This chapter gives a short history of human motion analysis and gives the purpose of building a new motion analysis system.
3	Requirements Definition	The main goal for the Sybar project is the development of a system that aids physicians in patient assessment. In this chapter, the requirements are given that are critical for the success of the project.
4	Requirements Specification	This chapter gives the requirements specification of Sybar. First, the system boundaries are defined. Then, the types of measurements that are supported are given and the system is described from the viewpoint of the physician. Finally, maintainability and performance requirements are given.
5	Analysis Model	In this chapter, the analysis model is described, using the object modeling technique (OMT). This concludes the analysis phase.

TABLE 1 Guide to the chapters

	Chapter	Description
Part 2: Design		
6	Toplevel Design	This chapter gives the top level design of Sybar. The top level design describes the decomposition of the complete system into subsystems. The toplevel design consists of both hardware and software
7	Recording Subsystem	This chapter describes the recording subsystem, which is responsible for the acquisition and management of measurement data.
8	Display Subsystem	The display subsystem is responsible for the display of measurement results, and the interaction that takes place during a viewing session.
9	Kinematics: Systems & Algorithms	Human kinematics can be measured in a large number of ways. This chapter first gives an overview of the devices and algorithms that are used. Next, the kinematics detection approach for Sybar is chosen.
10	Kinematics Measurement from a Single View	This chapter deals with the possibility of retrieving 3D kinematics from a single camera view for Sybar. First, Chen and Lee's approach is summarized and discussed. Next, the 3D reconstruction is formulated as a minimization problem, to be solved with the SVD-technique. Finally, some test results of the approach are presented and conclusions are drawn.
11	Kinematics Subsystem	This chapter describes the kinematics subsystem. It uses the results of the research of the previous chapters. First, the system is divided into three subsystems. Next, a description of each the subsystems is given.
Part 3: Process & Evaluation		
12	Design Methodology	This chapter discusses the choice of a design method for Sybar. First, design and design methods in general are discussed. Next, the general software engineering methods are described. Finally, the choice for a particular method and design strategy for Sybar is motivated.

TABLE 1 Guide to the chapters

	Chapter	Description
13	Design Process	This chapter describes the design process of the Sybar project. First, the process is divided into subprocesses. Next, these processes are described, including some of the problems that were encountered. Finally, the project team and the contributions of the members of the team are described.
14	Evaluation	This chapter evaluates the Sybar project. Both the developed product and the design process are evaluated. Finally, some lessons learned during the project are described.
Appendices		
A	Object Oriented Methods & OMT	This appendix gives a short introduction to object oriented methods and the Object Modeling Technique (OMT). Furthermore, it introduces two small extensions of the notation that are used in the thesis.
B	Hardware Implementation	Most of the hardware that was to be used for the Sybar project was bought specifically for the project. This appendix describes the hardware components and the way they are connected.
	Bibliography	

CHAPTER 2

Human Motion Analysis

The first human motion analysis systems date back to the previous century. Over the years, new methods to measure and analyze human motion have been introduced. This chapter gives a short history of human motion analysis and gives the purpose of building a new motion analysis system.

2.1 Short history of human motion analysis

Human and animal motion has intrigued humanity, and artists in particular, from the beginning of our existence. Ancient paintings of running hunters and prey in caves are still testimony of this. The first important pioneer in scientific motion analysis is the Italian physician and mathematician *Borelli* (1608-1680). In his *treatise on animal motion*, he discusses the working of motion using models based on observation and mechanical laws. His work was unsurpassed for almost two centuries.

In the 19th century, the invention of photography lead to new possibilities in registrating motion. Photographer *E.J.Muybridge* was the first to study the possibilities of using a series of photographs, shot with a number of cameras, to study human and animal motion. In particular, his photographs of race horses are well known.

Interestingly enough, the invention of cinematography was a direct result of the desire to analyze human and animal motion. It was the

French scientist *E.J. Marey* (1830-1904) who, inspired by Muybridge, invented the movie camera and projector. His main interest was that of registrating animal and human motion. For this purpose, he also developed mechanical sensors to measure foot pressure. His goal was to give a quantitative description of movement (also known as a *kinematic* description), and he was the first to present measurement data of human movement using a graphic method.

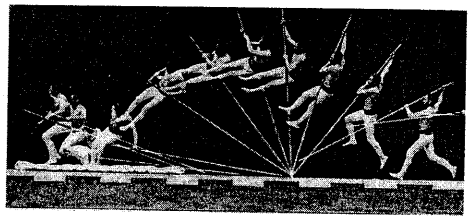


FIGURE 1
Motion analysis of pole vaulter using a photographic plate recording by Marey

Muscles are an essential part of the human motor system.The electrical activity of muscles can be measured, using a technique known as *electromyography*. The pioneer in this area is *Duchenne de Boulogne*, who improved existing measurement systems and studied the human muscle system in the 19-th century. Closely related to the measuring of electricity (kinesiological EMG) in the muscles, is the stimulation of muscles using electricity (clinical EMG). Both types of EMG were studied by Duchenne de Boulogne.

The first 3D-kinematic analysis of motion was performed by Braune and Fisher in 1891. Their method was based on Marey's photographic plates. They used four cameras and attached light tubes to the human subject. Preparing the subject took six to eight hours. All the 3D-calculations had to be done by hand and took several years.

FIGURE 2
Prepared subject in Braune and Fisher's motion analysis

Braune & Fisher also were the first to measure the forces and moments involved with the movement, also known as the *kinetics*. They did this by measuring inertial properties of body segments and combining these

with the 3D data. Braune and Fisher studied only one human subject.

Another important pioneer in the history of motion analysis is the russian scientist Nikolaj Bernstein. Using a systematic methodology, his research team studied a large number (800) of subjects. Like Braune and Fisher, Bernstein's data acquisition method was based on photogrammetry. Unfortunately, Bernstein's work was unknown for a long time because it was only published in the Russian language.

After World War II, a group of researchers at the University of California performed extensive research on human motion. Their goal was to gain insight in human motion in order to improve the design of artificial limbs. They used a number of measurement methods and devices, including an improved force plate device.

Since the California group did their research, the introduction of computer systems has made motion analysis much easier. A number of commercial motion analysis systems have become available. Furthermore, human-motion analysis labs have been established in universities, hospitals and research centers around the world. Human motion analysis systems are used in ergonomics, sports, the entertainment industry, ortopedics and rehabilitation medicine.

This concludes our short history of motion analysis. In the next section, we will give an overview of the measurement methods for motion analysis used today.

2.2 Current clinical measurement methods

In this section, we describe the types of measurements and measurement devices that are most frequently used in the motion analysis laboratories. More detailed overviews can be found in [11,13,14, 15]. Most analysis of human motion focuses on the movement of walking, because it is such an important motion in daily life. The analysis of walking is also known as *gait analysis*.

2.2.1 General movement parameters

In gait analysis, general movement parameters such as walking speed, step frequency, step length, etc. can be measured. For this, a stop watch or foot contact device is used.

2.2.2 Kinematics

Kinematics describe the movement of body segments. The historic studies by Marey, Braune and Fischer, based on photographic techniques, studied kinematics. Currently, there are a number of different ways that kinematics are measured: there are camera systems, acoustic systems, magnetic systems, goniometers, and accelerometers. These systems are described in greater detail in Chapter 9, which deals with kinematic detection systems and algorithms

2.2.3 Kinetics

Kinetics describe the forces and moments that cause motion. Kinetics follow from kinematics and inertial properties of model segments. Unfortunately, it is not easy to obtain these inertial properties. Therefore, kinetics of lower extremities are often measured with a force plate that measures the forces of the foot towards the floor. When combined with kinematics and inertial properties of leg segments, force plate data can be used to calculate moments in joints of the lower extremities.

2.2.4 Electromyography

The muscles are an essential part of the human motor system. Movement is achieved by the complex activity of contracting muscles that produce work. Since it is the activity of a number of muscles that causes joint rotations, the activity of individual muscles can not be determined from kinematics or kinetics. However, electrical activity of individual muscles can be measured, and this is known as electromyography (EMG).

Unfortunately, there is no direct relation between the electrical activity of the muscle and the force produced by the muscle. However, EMG can be used to determine at which point in time muscles are (relatively) active during a motion pattern, for example during a gait cycle. This is useful information to determine the cause of a

motion disorder. EMG is measured with electrodes that are applied to the surface or with wire electrodes that are applied directly into the muscle. Surface electrodes can only be used for superficial muscles and are not very specific, while wire electrodes give specific information on non-superficial muscles.

2.2.5 Energy expenditure

To perform a movement, the muscles in the human body contract, for which energy is needed. Energy expenditure is a measure for the efficiency of a person's motor skills. Energy expenditure is usually measured by determining the difference between the percentages of oxygen in inhaled and exhaled air. An alternative to measuring oxygen levels is the measurement of heart rate. However, this method is not very reliable, since the heart rate can also be influenced by other factors, such as mental stress.

2.3 Problem statement: measurement methods for patient assessment

The measurement methods described in the previous section each give information on certain aspects of human motion. An application of this information can be found in the treatment of patients with motion disorders. A distinction can be made between motion analysis in the context of *clinical research* and motion analysis for *patient assessment*. In the context of research, studies are done on data of populations of patients. The goal is to learn more about certain motion disorders. In patient assessment, a single patient is examined. Here, the goal is to determine the best treatment for the individual patient [7]. The measurement devices mentioned in Section 2.2 have found applications in clinical research. However, only few applications can be found in patient assessment [3,5,11,13].

There are several reasons for the lack of motion-analysis applications in patient assessment. First of all, there is a great diversity between individuals, both in normal and pathological cases. It is therefore difficult to draw conclusions from a single case. Furthermore, in case of permanently disabled patients, there is an additional problem. The goal of the treatment of permanently disabled patients is to improve motor skills to a level of functioning that enables the patient to perform the most important tasks of daily life.

The way this is achieved depends very much on the disability of the patient, and it may involve alternative ways of movement. This means that comparison with movement patterns of healthy persons is often not relevant.

A problem of a different nature is that physicians are not trained in interpreting results from biomechanical measurement devices. Physicians are also sometimes not convinced of the usefulness of a biomechanical analysis. As was noted by Woltring, this problem was already mentioned by H. Boerhaave in 1703: "Those who estimate the corporal forces through Geometric Calculus from mass, shape, and velocity, either theoretically or experimentally, are called Mechanicians (....). All the civil and martial arts recognize the utility of this science (...). Only the physicians mistrust her and judge generally, without further examination, that she can be of no use whatsoever" [9].

The lack of human-motion analysis applications for patient assessment is the topic of much debate in the biomechanical engineering community. One approach that has been propagated to introduce motion analysis, is the development of measures that indicate the likelihood that a patient has a certain disease. However, only a few results have been reported in this area [3].

One method that *is* accepted and used by physicians is that of *human observation.* A patient performs an exercise and is observed by the physician. The physician then reaches conclusions based on his education and experience. A simple enhancement is to record the patient on video, which enables the physician to view the patient in slow-motion [14]. In a sense, this can be seen as an application of Marey's invention of cinematography. Some work has been done to try to standardize this procedure for motion analysis [16].

In the field of biomechanics, observational analysis is often criticized [11,13]. The main objection is, that the assessment is determined largely by the individual observer. Furthermore, it is simply not possible for humans to see the causes of motion (the muscle contractions), and the forces involved. Research that has been done on the reliability of observational gait analysis, shows that it is indeed only moderately reliable [8,12,16].

2.4 Our approach

Design Decision: The approach we choose to follow is based on presenting biomechanical data together with video images that can be used for observational analysis. This way, we try to get the best of both worlds.

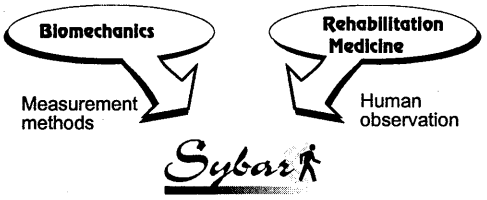


FIGURE 3
Sybar in relation to biomechanics and rehabilitation medicine

We take the clinically accepted way of human observation as a starting point. Instead of providing large amounts of measurement data separately, the existing image of the patient is enhanced with additional information. We believe that by showing a video of a patient in combination with graphics measurement results, it becomes much easier to directly relate the data to the actions of the subject under investigation. Furthermore, observational analysis can be instantaneously combined with interpretation of results from biomechanical measurements.

The approach is not completely new. Some experimental research on systems that combine video with data has been done with force plate data [1,2], and EMG data [6,19]. The Sybar project deals with the development of a system for use in a clinical environment that combines information from all relevant measurement systems.

Although we believe that this is a promising research direction, we do not claim to know in advance which data is most useful to physicians. Sybar therefore has to be a flexible system that evolves together with clinicians who have to learn to interpret the data. A technical opportunity that gives new possibilities in the development of motion analysis systems, is the recent availability of relatively cheap digital video equipment. Existing systems still use analog video. Using digital video gives new possibilities in the area of user interaction and display.

In the following chapter, this initial approach is expanded into a set of requirement definitions that we feel are essential in the development of a clinical application for motion analysis.

CHAPTER 3

Requirements Definition

The main goal for the Sybar project is the development of a system that aids physicians in patient assessment. In this chapter, the requirements are given that are critical for the success of the project.

3.1 Function of the requirements definitions

In this chapter, the initial approach is translated it into a set of requirements that are the basis for determining the more detailed requirement specification. The requirements are also used as guidelines in the design process. In the evaluation, we discuss whether Sybar succeeds in fulfilling the requirements definition.

3.2 Clear presentation of measurement data

In our society, one is frequently confronted with huge amounts of information of all kinds of different types. Most of this information does not interest us at all. However, to separate relevant information from useless information is no easy task. Furthermore, the combination of several pieces of apparently useless information may lead to the conclusion that this information is useful after all.

Sometimes, human capabilities to grasp information are not enough and a phenomenon called *information overload* occurs. Information

overload occurs when a person is presented with large amounts of unstructured data, and as a result has the idea that no conclusions can be made, although the information to make these conclusions is probably available. Whether information overload occurs, depends for a large part on the experience of the person involved.

Human motion measurement systems produce large amounts of data. Biomechanists are relatively well trained in interpreting these data. However, the end-users of Sybar are physicians. Physicians are not trained in dealing with these types of measurement data. If the information that motion analysis systems present is presented as a collection of raw data, physicians will surely suffer from information overload.

This leads to our first requirement:

Requirement 1: *Measurement information is to be presented in a way that is useful for physicians.*

3.3 User friendliness

User friendliness and human-computer-interaction are becoming more and more important in the design of computer systems. User friendly systems allow the user to work more efficient and with more pleasure. Unlike in the past, unfriendly systems are now often simply rejected by users. Projects can fail because of lack of attention to user interfaces.

The following types of users in clinical motion analysis systems can be distinguished:

1. The physicians: these are the end-users of the system. Physicians usually have little experience with computers. They also have little time to learn complicated interfaces.
2. The lab operators: these operate the data acquisition equipment. Lab operators also prepare measurements data for presentation to physicians. Lab operators are more used to working with complicated equipment.

Considering the price of a physician's time and his inexperience with computers, it is obvious that user friendliness to the physicians

is essential in the design of Sybar. The basic user task for physicians is that of navigation through the measurement data. This leads to our next requirement:

Requirement 2: *Navigation through measurement data should be simple and intuitive for physicians.*

The lab operators deal with the data acquisition and data processing necessary for the presentation. Although more effort from lab operators in learning the system can be expected, it is clear that lab operators should also find the system easy to use, especially considering that they will be the users that use the system the most.

In a clinical environment, results have to be available quickly. Therefore, it is essential that the process of data acquisition and processing does not take too much time.

Requirement 3: *The lab operators should find the system easy to use and should be able to perform data acquisition and data processing quickly.*

3.4 Patient friendliness

The patients on which motion analysis is performed are not users. However, it is obvious that they are to be taken into consideration. Patients usually have some sort of motion disorder and often cannot move very easily. Therefore, the strain that measurement system have on patients should be kept to a minimum. Another standard requirement for medical equipment is that the measurement system is safe to use.

Requirement 4: *The measurement system has to be patient friendly and safe.*

3.5 Reliability

Since the goal of the project is to deliver a system for daily clinical use, it is expected that physicians will over time become more and more dependent on it. Therefore, it is essential that the system is reliable.

Requirement 5: *The system should be reliable*

In particular, two aspects are relevant:

- It is important that no patient data can be lost.
- It is important that the information presented to the physicians is correct

Reliability in terms of system crashes and 'mean time between failures' is not really an issue. Of course, a system that crashes a lot can never be user friendly and therefore violates the other requirements anyway.

3.6 Maintainability

There are several types of software maintenance [69]:

- *Corrective maintenance* is concerned with fixing errors in the code.
- *Adaptive maintenance* is concerned with adaptation of the software to new circumstances, for example new hardware.
- *Perfective maintenance* is concerned with meeting new requirements.

All three types of maintenance will be necessary for Sybar. In particular perfective maintenance will be necessary in order to continue the improvement of Sybar by evaluating the physicians use of the system. Adaptive maintenance is anticipated because of the developments in the area of digital hardware: new video cards with better performance will improve the quality of the video images.

Requirement 6: *The system should be maintainable.*

3.7 Resource constraints

The amount of resources available to accomplish the design is limited.

- *Budget constraints:* no particular budget has been assigned to the project, however, each purchase needs to be approved of by management. The budget constraints is particularly relevant during hardware design and implementation.

- *Human resources*: the amount of people available for the project depends on which specific skills are necessary. Again, each request for human resources needs to be approved of by management.
- *Time*: the time span for the Ph.D. project is 2 years and 9 months.

These resource constraints do not lead to specific requirements, but they do have an impact on the design.

CHAPTER 4

Requirements Specification

This chapter gives the requirements specification of Sybar. First, the system boundaries are defined. Then, the types of measurements that are supported are given and the system is described from the viewpoint of the physician. Finally, maintainability and performance requirements are given.

4.1 Function of the requirements specification

This chapter is a more detailed description of the requirements, using the requirements definition as a basis. In the requirements definition, we managed to avoid design decisions. This chapter, however, contains a large number of design decisions, some of which were determined at the later stages of the project. The fact that analysis and design activities can not be completely separated is discussed in the evaluation, see Section 14.4.3.

4.2 System boundaries

System boundaries define what is considered to be part of a system and what is considered to be part of the environment of a system.

The complete 'Sybar Lab' consists of:

- A measurement system that performs the data acquisition and the data processing

- A viewing station that allows the measurements to be viewed
- A lab operator that performs the measurement.

Note that the lab operator is considered to be part of the system. The reason for this is that the lab operator has a supportive task, and no functional requirements of his own. Therefore, the system will not be described from the view point of the lab operator in this specification.

The environment of the system consists of the physician and the patient.

4.3 Types of measurements that are supported

Design Decision: The choice of measurement devices that are supported, has been based on availability in the motion analysis lab at the VU Hospital. Sybar supports the use of video recording, EMG measurements and force plate measurements. Furthermore, kinematics can be retrieved from the video.

Sybar is a general human motion analysis system. The most important type of analysis in rehabilitation medicine is the analysis of walking gait. However, the analysis of other exercises, like getting out of a chair or jumping, is also possible with Sybar.

4.3.1 Kinematics

An important way to describe human motion, is by means of a kinematic description. Kinematic data consists of a model of the human body and a description of its behavior through time. There is a distinction between 2D kinematic and 3D kinematic descriptions. Since human motion is in principle a motion in three dimensions, 3D kinematics are to be preferred. Furthermore, many gait pathologies involve movements in the frontal plane, and require 3D analysis [7]. However since the acquisition of 3D data is much more complex, sometimes a simple 2D kinematics analysis is used.

Design Decision: Sybar should at least support 2D analysis. Support for 3D analysis is very desirable and should be investigated.

There are many kinematic detection systems, with many years of development effort and, in general, an emphasis on accuracy of the data. In chapter 9, the approach for Sybar is determined. However, at this point, it can be said that the accuracy of the kinematics detection is not as important as the visualization and ease of use of the overall system.

4.4 The physician's view of the system

From the viewpoint of the physician, the system consists of a computer screen, a keyboard, and a mouse interface.The physician can look at the results of the motion analysis measurement, as prepared by the lab operator. At this point, the reader is reminded of the two relevant requirement definition: *clear presentation of data* (req.def. 1) and *easy navigation though data* (req.def. 2). This section translates these requirements into a user interface. The design of the user interface consists of a large number of design decisions. Only the more interesting ones have been indicated in the side line.

4.4.1 General screen lay-out

Design Decision: The user interface uses the standard 'desktop-metaphor'[69]. The screen lay-out consists of a menu and a number of windows, depending on the chosen *display configuration.* The windows are either *dialogs,* to set parameters and options, or *displays.* Displays show the measurement results in a certain way. Only one patient exercise can be watched at a time. There is a global *time state,* that influences what the displays show.

Figure 4 shows an example of a screen lay-out.

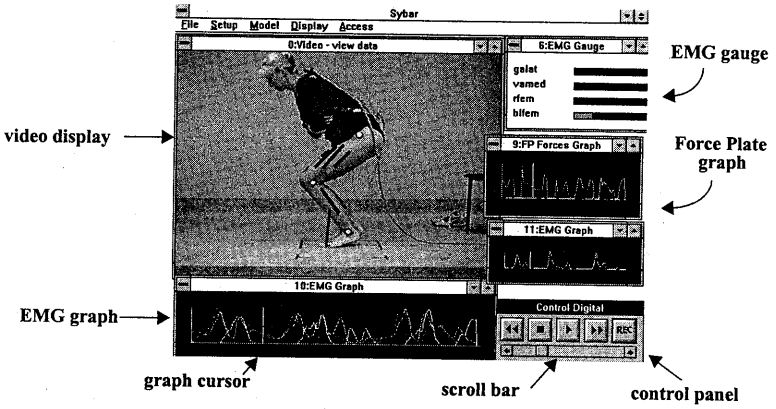
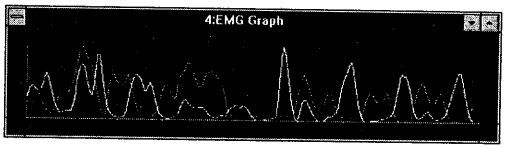


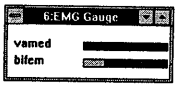
FIGURE 4
Example of a screen lay-out

The following displays are available:

- *video displays* display the video. The video displays shows the video image closest to the time state. The video display is described in Section 4.4.2.
- *graphs displays* show measurement data (either EMG or force measurement) in a graph. The time axis is always horizontal. The vertical axis shows measurement results. A cursor (vertical line) shows the place of the time state on the time axis.



- *gauge displays* show measurement data values at the time state in the form of a gauge. These are only available for EMG.



- *3D kinematics displays* show a view of the kinematics of (part of) the patient, using a stick diagram model, which can be

viewed by position a virtual camera. The kinematics display is described in Section 4.4.3.

Any number of displays can be shown at a time, and because displays are windows, they can be moved and scaled to any position. This gives the possibility of choosing a display configuration, depending on the measurement data that is available, or the information that the physician wants to focus on.

The video display and the kinematics display are now described in greater detail.

4.4.2 The video display

Design Decision: Since our system is based on observational analysis (see Section 2.4), the most important display is the video display that shows the patient. Most attention of the physician will be directed towards this display. Therefore, the easiest way for the physician to interpret the additional measurement data is when it is displayed in a meaningful way *inside* the video image.

Force plate and EMG can be shown in the video display:

- Force plate data represents the force vector of the foot on the force plate. This is shown as a 3D line in the video image. A shadow indicates the projection of the vector on the floor.
- EMG data represents (electrical) muscle activity. It can be shown as a moving gauge at the location of the muscle. To display the measurement at the location of the muscle makes sense, since this way it is easy to see which measurement corresponds with which muscle (no names are necessary).

There has been some discussion in the Sybar project on how to display the EMG-levels. Alternatives to the 'moving gauges' are:

- Colored muscles, where the color indicates the value. The problem with color-scales is that they are harder to interpret than gauges [76].
- Vertical gauges. Using vertical gauges makes it easier to compare different gauges, since they are always aligned. However, the image becomes less clear because the gauges no longer

respect body contours, and the correspondence to muscle locations is less clear.

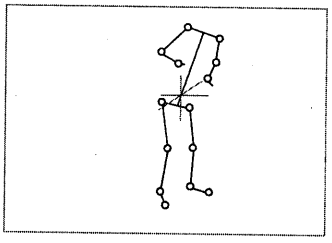
The most important camera view is the sagital view, where the patient is seen from the side as in Figure 4. However other views, like the coronal (frontal) view, are also supported.

4.4.3 The 3D kinematics display

The purpose of the 3D kinematics display is to give a visualization of the 3D kinematics from a view point that can be selected by the user, see Figure 5. The user can move the camera on a sphere that has its center at the gravity center of the human model. The gravity center of the human model, is defined as the average of all model points. The camera is automatically directed towards the gravity center. This is a design decision that favors ease of use over flexibility: the user can not look in any direction he wants. It is assumed that the only interesting direction to look at is the center of gravity. By moving the mouse, one can move the camera over the sphere.

By moving the mouse up and down in combination with the control key, it is possible to change the radius of the sphere. By using the tab key, the focal distance can be changed. Muscles are again shown as gauges, and the force vector is shown as a 3D vector.

FIGURE 5
Kinematic display that shows stick figure model

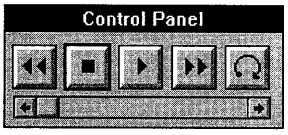


Design Decision: The chosen navigation mechanism in the 3D world favors ease of use over flexibility: the user does not have the freedom to choose the viewing direction. It is assumed that the user want to look at the stick figure and therefore the view is automatically directed towards the stick figure.

4.4.4 Navigation through time

Design Decision: The displays all depend on the time state of the system. Therefore, the most important type of navigation is *navigation through time*. For navigation through time, a video recorder-metaphor is used. It is possible to play, rewind, forward and stop a recording of a measurement with a user interface element that is similar to a remote control of a video recorder, see Figure 6. If a recording is playing, the time state is automatically adjusted. Updates are performed as quickly as the hardware permits. There is a scroll bar that represents the time axis. Moving the scroll bar cursor sets the time state to the corresponding time. There is also a 'cycle button'. If this button is pressed, the video is continuously played from the start to the end and over again.

FIGURE 6
The Control Panel



Another way for the user to change the time state is to click in a graph. The time state is then set to the point in time corresponding to the point on the time axis where the user clicked. This feature is useful to quickly navigate to certain points in time, based on information from graphs. For example, the situation at a peak in a graph can be examined by clicking on the peak.

4.4.5 Navigation through data

The user has the option to focus on a selection of the measurement data. First of all, the contents of a display can be chosen. The data consists of a number of *channels*, each representing a series of measurements in time with a single sensor. One or more channels can be shown in a graph. In the video display, the force plate and EMG annotation can be switched on or off. Furthermore, by moving and scaling windows, the screen can show a selection of the available displays.

During their lifetime, displays are either visible or hidden. If the users wants to focus on part of the data, a display can be hidden. Displays that are hidden can be made visible again. A user can also

end the life of a display by deleting it. Deleted displays can not be restored, but new displays can created at any time.

Manipulation with displays is performed via a display configuration dialog, that shows the created displays, and allows operations on them, see Figure 7.

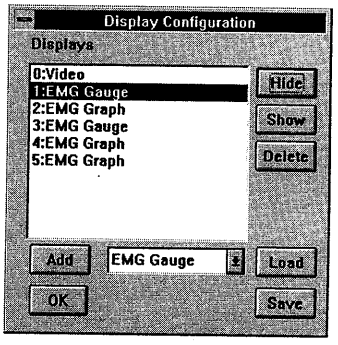


FIGURE 7
Dialog to change the display

Another way to focus on a selection of the measurement data is to focus on particular EMG channels by hiding others. EMG channels can be hidden by clicking on EMG bars in a gauge display. The selected EMG is then no longer shown in any of the displays. The gauge shows a grey bar for a hidden channel. By clicking on a hidden channel, it becomes visible again.

4.4.6 Navigation through measurements

Navigation through several measurements is performed via *sessions*. A session represents the state of the system, as viewed by the user. A sessions consists of the measurement data corresponding to a single video and the display configuration chosen by the user.

A session can be stored on disk and reloaded. Via loading and saving sessions, it is possible to quickly view information from several measurements. Sessions can also be prepared by the lab operator.

4.4.7 Single-document interface

Design Decision: An important restriction is to allow only a single recording to be viewed at a time. This is known as a single-document interface (SDI)

Developing a SDI application has the advantage that:

- it is easier to design (keep it simple)
- there can be no confusion on which displays correspond to which recording, since there is always a single recording

A multiple document interface (MDI) has the advantage that:

- different recordings can be compared directly
- it is more general: MDI can always be turned into SDI

The final argument that lead to the decision was that it would be impossible to implement a MDI because of video hardware constraints: it is not possible to show two videos at the same time.

4.5 Maintainability requirements

One of the requirements from the requirements definition is that the system should be maintainable (req.def. 6). The problem with maintainability is that it is difficult to predict what kind of maintenance will be necessary. However, some remarks about maintainability can be made:

- The design of Sybar should support the addition of future measurement devices. It is very likely that new measurement equipment will be introduced at the rehabilitation department. It should be relatively easy to incorporate these into Sybar.
- The addition of new displays should also be supported in the design. One of the most obvious user requirements that will keep changing is the functionality of the displays. Therefore, changing and adding display possibilities to the system should be straightforward.
- The design must be hardware independent whenever possible. In particular, it is anticipated that new video cards will be used in the future.

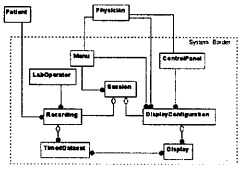
In contrast to the other requirements in this chapter, these requirements are on the design of the system, and are therefore not directly visible to the users.

4.6 Performance requirements

The video of the patient must be shown in such a way that the physicians find it useful (req.def. 1). A high frame rate obviously is desirable since it gives a more detailed view of the movement. However, there are technical limitations in the frame rate that can be achieved with video hardware. A frame rate of 25 frames per second is enough for analysis of most human motion. This frame rate corresponds with Standard PAL video. The analysis of fast sport movements and high frequency shaking may require a higher sampling rate (to avoid aliasing), however these types of analysis are not the primary focus of Sybar. It is therefore specified that the data in Sybar should be available with at least 25 frames per second. This requirement is important for the hardware design.

CHAPTER 5

Analysis Model



In this chapter, the analysis model is described, using the object modeling technique (OMT). This concludes the analysis phase.

5.1 Function of the analysis model

This chapters describes the analysis model of Sybar, using the Object Modeling Technique (OMT). The analysis model is the model that describes what the system should do. It consists of three views of the system: the object model, the dynamic model and the functional model. A short introduction to object oriented development and OMT can be found in the appendix.

Sybar is described from the viewpoint of the physician. This means that the acquisition and processing of measurements is not modeled at this stage: it is considered to be part of the design.

5.2 The object model

A key aspect in object oriented development is the identification of classes of objects. The classes of objects and their relations are described in the object model. Furthermore, the object model also shows relationships between objects, and the relation of the system with the environment. The environment of Sybar consists of the physician and the patient.

Figure 8 gives a global version of the model, without attributes or operations.

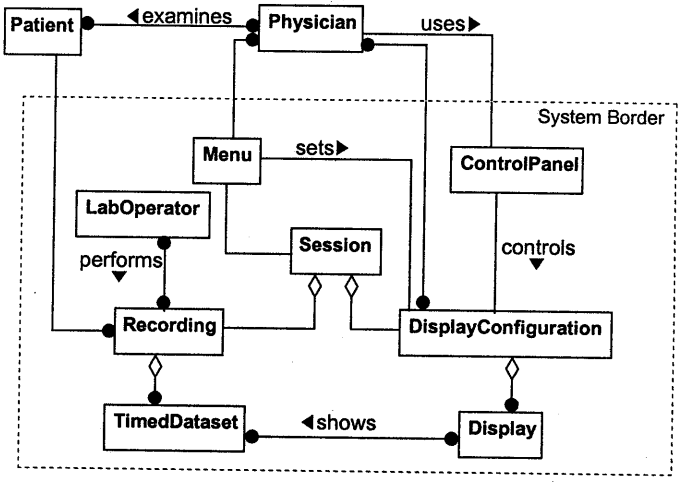


FIGURE 8
Object Model

5.2.1 Patient, Physician and Lab Operator

There are three human objects in our model. The *Patient* is the subject of the motion analysis. The *LabOperator* performs the measurement and preparation of data. Since the *LabOperator* has a supportive function, he/she is considered part of the system. The *Physician* uses the system to view measurement results.

5.2.2 Recording and DisplayConfiguration

The main purpose of Sybar is to display measurements on patients in the context of motion analysis. For this reason, we introduce two classes that play a vital role: The *Recording* class and the *DisplayConfiguration* class. A *Recording* manages the data in a measurement. A *DisplayConfiguration* takes care of the visualization of the measurement.

A *Recording* object manages the datastreams of a measurement on a *Patient*. A *Recording* consists of a number of *TimedDatasets*. A *TimedDataset* is an object that contains a number of time dependent values of a datastream. The *TimedDataset* class is an abstract class:

only descendant classes can be instantiated. A *Recording* is controlled by the *LabOperator.*

A *DisplayConfiguration* object consists of a number of *Display* objects. These objects display part of the data from a *Recording* in a certain way. Like the *TimedDataset*, the *Display* class is an abstract class. The *DisplayConfiguration* also contains a dialog that enables the user to add, delete, show and hide displays (see Figure 7).

5.2.3 Session

A *Session* consists of a *Recording* and a *DisplayConfiguration.* Sessions can be stored and retrieved. This makes switching between measurements easier. A session can also be retrieved in part: for instance, only the *DisplayConfiguration* can be loaded. This makes it possible to re-use a *DisplayConfiguration* for another *Recording.*

5.2.4 ControlPanel and Menu

There are two classes that deal specifically with user interaction between the *Physician* and the system: the *ControlPanel* and the *Menu.*

The *ControlPanel* is comparable to a remote control of a video recorder or stereo: it contains stop, play, fast forward, fast backwards and slow motion buttons (see Figure 6). It is the primary way to navigate through time. When the time is changed, this influences the *DisplayConfiguration.*

The *Menu* is a window-menu that is used for simple user interactions:

- Creation of additional displays
- Loading and saving of *Sessions*
- Inititiation of *DisplayConfiguration* dialog

5.2.5 TimedDatasets

There is a *TimedDataset* for each of the data types described in Section 4.3 of the requirements specification:

- *EmgData*, contains information about electrical-activity in the muscles.

- *FpData*, contains information about forces, as recorded by the force plate.
- *VideoData*, shows a video of the patient performing an exercise.
- *Kinematics,* describes the movement of the patient.

5.2.6 Displays

The following *Displays* are available (see Section 4.4.1):

- *EmgGauge/EmgGraph*, displays EMG in dynamic bar diagrams or graphs.
- *FpDisplay*, displays force plate data.
- *VideoDisplay*, displays a video of the patient, together with EMG, force plate and kinematics.
- *Kinematic Displays*, displays kinematics of the patient.

The *Display* objects are synchronized when they are displaying data. For this purpose, all the *TimedDatasets* have a common time scale. The *TimedDatasets* and *Displays* and their connections are shown in the object model of Figure 9.

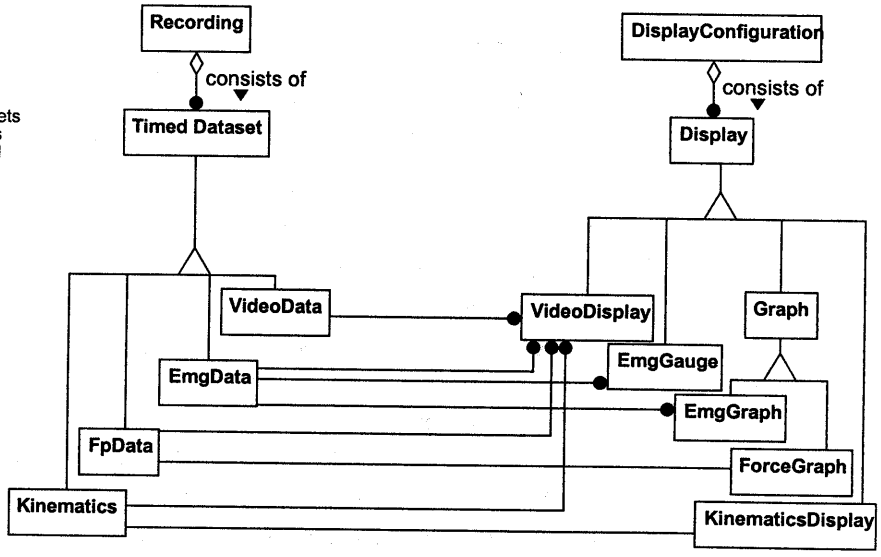
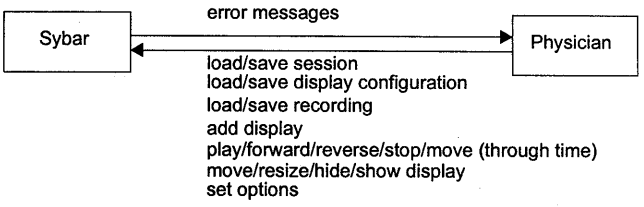


FIGURE 9
TimedDatasets and Displays object model

5.3 The dynamic model

The Dynamic model describes the desired dynamic behavior of objects by means of state diagrams and the event flow diagram. The event flow diagram for Sybar can be seen in Figure 10. It shows the types of events that can be expected from the environment. Since the display of information is not considered an event, the only events that are given back are possible error messages.

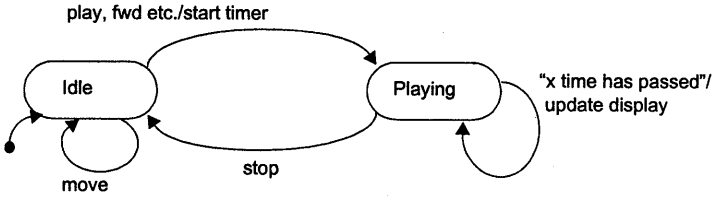
FIGURE 10
The event flow diagram



The dynamic model of the individual classes is simple and will only be described for the *Display* class.

5.3.1 The Display class

A *Display* has two states. It can be either idle or playing (playing includes rewinding, slow-motion etc.). The object can go from idle to playing when certain buttons are pressed by the user. In the idle state, a still frame is shown. When a move message is received, a different frame is shown. When a display is playing, it updates itself with a certain update frequency.



5.4 The functional model

The functional model describes data value transformations and dependencies. Figure 11 describes the input and output values of the system as a whole.

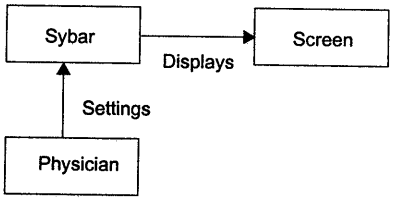


FIGURE 11
Input/Output

Figure 12 shows the data flow diagram for Sybar. The *Physician* loads and saves sessions, sets options and sets the time state. A *Session* consists of a *Recording* and a *DisplayConfiguration*. The display process uses the recording, display configuration and time state to display a measurement on the screen.

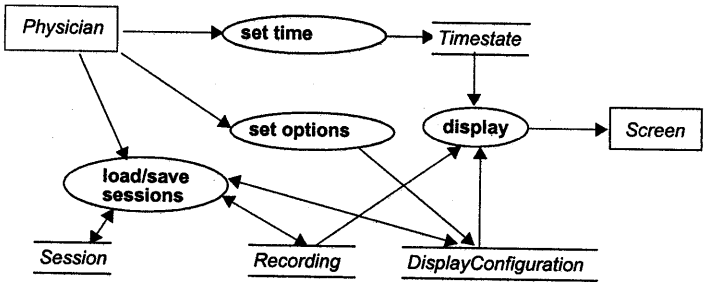


FIGURE 12
Data flow

CHAPTER 6

Toplevel Design

Toplevel System Design

This chapter gives the top level design of Sybar. The top level design describes the decomposition of the complete system into subsystems. The toplevel design consists of both hardware and software.

6.1 Introduction

The first step in system design is the decomposition of the system into subsystems. Sybar consists of both hardware and software. We first give a hardware design, then we give a global design of the software system. The hardware subsystems consist of standard components and are described in Appendix B. The software subsystems will be described in the following chapters.

6.2 Decomposition of Sybar

At the top level, Sybar consists of the following two parts:

1. A *measurement system*, consisting of a set of measurement devices: an EMG measurement system, a force plate, and one or more video camera's.
2. The *computer system* that digitizes measurements, determines kinematics from video images and displays measurement results.

3. A lab operator controls the measurement

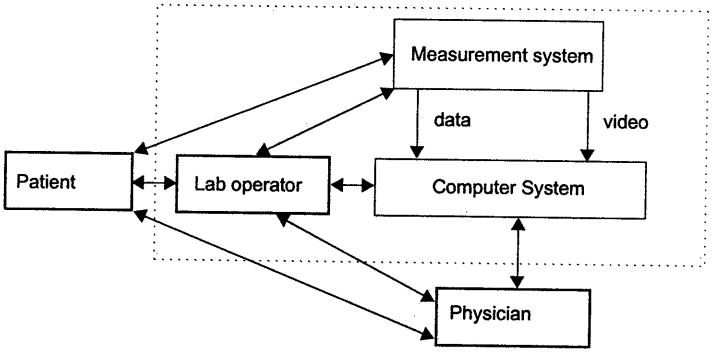


FIGURE 13
Top level subsystems

6.3 The measurement system

The measurement system consists of a motion lab with measurement equipment.

1. One or two video cameras and a video mixer
2. An EMG measurement system
3. A force plate measurement system
4. A synchronization module
5. A video recorder to temporarily store the video

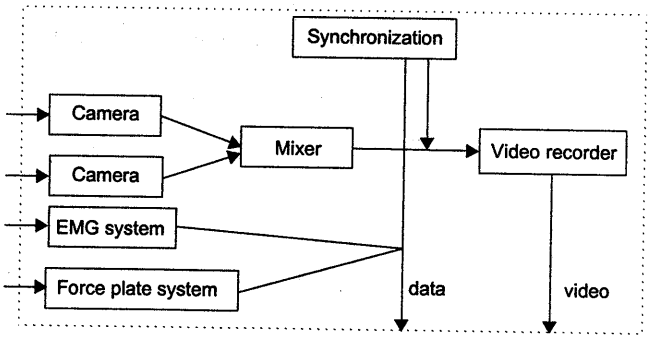
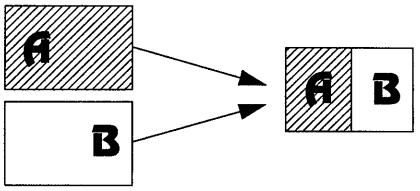


FIGURE 14
Measurement system

6.3.1 Support for multiple camera's

Design Decision: The motion analysis lab supports the use of a multiple camera recording. This is done by combining the images of two cameras into a single video image, using a video mixer device. The video mixer takes two video signals as input and produces a single video as output. The output video shows both input videos in the original size and resolution. However, in order for the two input videos to fit, part of the video is left out:



The advantage of combining several camera images in a single video image, is that only a single video recorder is needed. Furthermore, video digitizers can only deal with a single video signal at a time. By combining video signals, the camera streams can be digitized in a single digitizing session.

The alternative to using a single video channel, is obviously to use a channel for each camera.The advantage of this alternative is that no resolution is lost in combining the video images. However, this approach was considered not to be feasible given the budget for the project and the hardware currently available on the market.

6.3.2 Synchronization

It is important that the Sybar software can show measurement results in a synchronized way. For example, the user must be able to see which EMG values correspond with a video image. The function of the synchronization module is to send a synchronization signal with the measurement data and the video data. This synchronization signal is recorded during the digitizing of both the measurement data and the video signal, and is used by the software to synchronize the various types of data.

6.3.3 Video recorder

Design Decision: During a measurement, the video (including the synchronization signal) is recorded on a standard video recorder. An alternative would have been to digitize the signal directly from the camera. However, digital video requires large amounts of disk space, and we therefore choose to first store the video on analog tape. A selected time period can be chosen at a later stage after the measurement has taken place.

This design decision is also motivated by the reliability requirement: if the computer system for some reason fails, there is always the video recorded image to fall back to. The video recorder is a backup for the digitized data.

6.4 The computer system

The computer system runs the Sybar software. It has dedicated hardware to digitize measurements signals. In the following section, the Sybar software is described in terms of software subsystems. In the following chapters, each of the subsystems is described in detail. No assumptions are made as far as multiple processing or networking is concerned.

Design Decision: The use of a secondary processing system for the specific purpose of kinematics detection was considered, but it was concluded that such an approach was not useful because the decompressing and transporting of the digital video would take more time than the processing itself.

6.5 Sybar software

6.5.1 Software subsystems

Figure 15 gives a division of the Sybar software into subsystems.

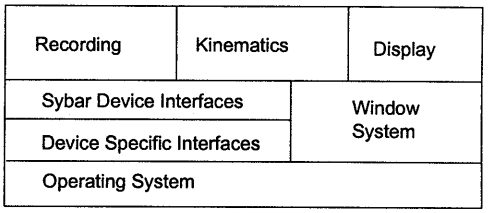


FIGURE 15
Decomposition of the Sybar software into subsystems

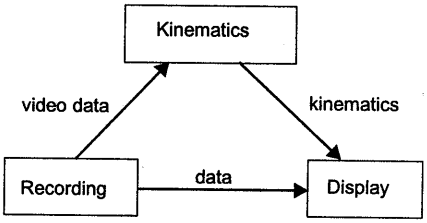
The subsystems which are shown in bold are part of the Sybar software. The other subsystems consist of general purpose software used by Sybar. Each subsystem can only use functions from lower levels, not from higher. Objects from different subsystems in one level can communicate with each other, although they are more likely to communicate with objects from the same subsystem. We now describe the subsystems in short.

- **Recording subsystem**: takes care of acquiring the data provided by the measurement system and the video provided by the video recorder.
- **Kinematics subsystem**: retrieves kinematics from the video data
- **Display subsystem**: takes care of interactive display of measurement results
- **Sybar device interfaces**: deliver general interfaces to hardware devices, independent of the specific device used.
- Device specific interfaces: deliver interfaces to specific devices, as defined by manufactures.
- Window system: provides user interface and window functions.
- Operating system: provides low-level operating system functions.

Figure 16 shows the data-flows between the top-level subsystems. The diagram shows that the recording system is a server for the other systems: the recording does not require input from the other

subsystems. The display is a client of the other systems, and the kinematics subsystem can be seen as a data processor.

FIGURE 16
Data-flows between top level



Design Decision: We have chosen to separate data and the display of data. This seems a natural division of tasks, and indeed is quite common in the object oriented world. The alternative, to let data objects take care of their own displays also has advantages, and is in a sense more object oriented. However, there are also displays that combine data from several measurement data objects. For this reason, we decided to combine all displays in a display subsystem. The recording and kinematics subsystem provide support for the display system, including operations for objects to 'draw themselves'.

Design Decision: Another decision is that user interface aspects are *not* put into an independent subsystem. Putting user interface aspects in a single subsystem has the advantage that porting the application to other user interface platforms is easier, since the other subsystems then stay unchanged. However, we have decided to divide the user interface aspects over the subsystems, since every subsystem requires quite a lot of user interaction. Whenever possible, user interface aspects have been put in separate classes. For instance, dialogs to manipulate object attributes are always separate objects. One argument for not completely separating the user interface is that it was not expected that the software would have to be ported to a second platform. Therefore, the maintainability requirement is not endangered by this decision.

6.5.2 Session class

A session consists of a recording and a display configuration, see Section 5.2.3. The *Session* class is not part of any of the subsystems. It could be considered a subsystem on its own, however

because it consists of a single class it is not described as such. The function of the Session class is to direct user actions such as saving and storing data to the specific subsystems.

6.5.3 Synchronization in software

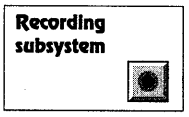
As was mentioned in Section 6.3.2, the measurement and video data is to be displayed in a synchronized way. Both the measurement data and the video data contain a synchronization signal which is stored in the recording subsystem. The display subsystem sends requests to the recording subsystem for data corresponding to a certain point in time. The recording subsystem ensures that all the objects use the same time scale. When a measurement is played back at normal speed (as required in the specification by Section 4.6) the display subsystem uses the operating system timer functions to periodically generate updates.

6.5.4 Description of the software subsystems

In the following chapters, the Sybar software subsystems are described in further detail. The description of each subsystem starts with the system design, followed by the design of the individual classes. The kinematics subsystem is the most complex subsystem, and is divided into three chapters. Chapter 9 gives an overview of existing kinematic systems and algorithms. In chapter 10, a particular approach is researched. Chapter 11 gives the final kinematics subsystem design.

CHAPTER 7

Recording Subsystem



This chapter describes the recording subsystem, which is responsible for the acquisition and management of measurement data.

7.1 System design

The recording subsystem has to following basic functions:

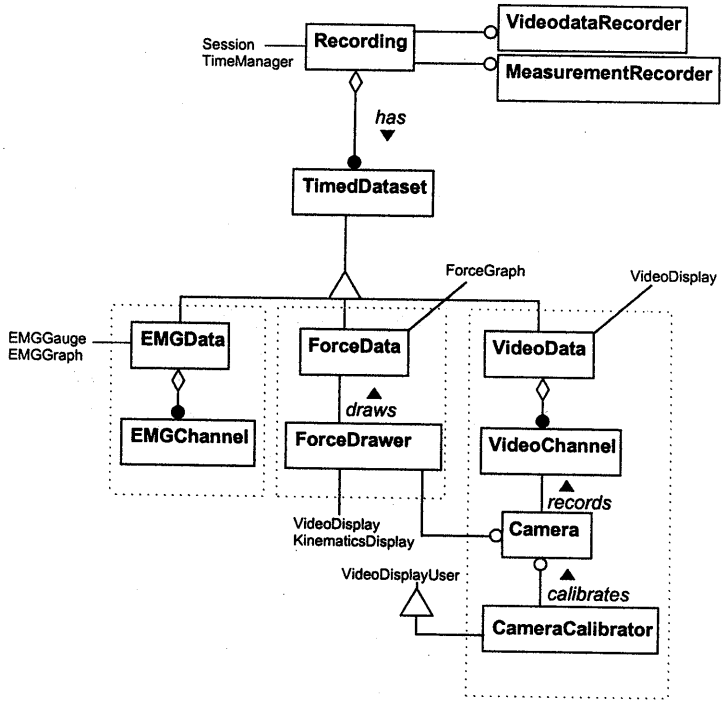
1. data acquisition of measurement data through external devices
2. storage and retrieval of the data
3. providing services for the display and kinematics subsystems to access this data

The most important class in the recording subsystem is the *Recording* class. Initially, a *Recording* is empty. A *Recording* can acquire a number of *TimedDatasets*. Each *TimedDataSet* contains information of a certain media (such as video, EMG, or force data). Each *TimedDataSet* has operations to store and retrieve its data. A *Recording*'s *TimedDatasets* are used by other objects. For example, An *EMGGraph* from the display subsystem uses a *TimedDataset* containing EMG data. All the *TimedDatasets* use the same time scale. This enables the display subsystem to show the data in a synchronized way (see Section 6.5.1).

Figure 17 shows the recording subsystem object model, which is based on the recording part of the object model described in Figure 9, Section 5.2 of the analysis. Associations that are beyond the boundaries of the subsystem are shown, but the classes are shown in a smaller font without boxes.

A difference with the analysis model is that the *Kinematics* dataset has been left out. The kinematics data are determined and managed in the kinematics subsystem.

FIGURE 17
Recording subsystem object model



The specializations of the *TimedDatasets* correspond to the measurement types described in Section 4.3 of the requirements specification. The following *TimedDatasets* are available:

- *EMGdata*: an *EMGdata* object consists of a number of *EMGChannels*. Each channel corresponds with one EMG electrode connected to a muscle.

- *Forcedata*: a *ForceData* object describes the force vector as measured by a force plate device. The *ForceDrawer* draws the force vector in a given display, using a *Camera* object to make the projection from 3D to 2D. The force drawer is used by several display objects. It is part of the recording subsystem because it is not a *Display* by itself: it provides the service of drawing forces *inside* a *Display*.
- *Videodata*: a *Videodata* object contains digitized video. Other objects, in particular the *VideoDisplay* from the display system, send commands to the *Videodata*, such as *move, play* and *stop*. A *Videodata* object can store video information from several *VideoChannels*. Each *VideoChannel* is recorded by a video camera.

  To perform 3D motion analysis and to draw force vector information at the correct location in the video, information regarding the position of the camera is necessary. Camera parameters can be obtained via a method called *camera calibration*. In Section 9.6, the process of camera calibration is described in the context of kinematics detection. At this point, we introduce a *Camera* object that contains the parameters that determine the viewing transformation, and a *CameraCalibrator* object that takes care of the calibration process.

Design Decision: The *Videodata* is acquired by digitizing from a video source. For this purpose, the *Recording* uses a separate object, a *VideodataRecorder*. The EMG and force data are acquired using a data acquisition board. The *Recording* uses a *MeasurementRecorder* to take care of the user interaction necessary for the data acquisition. By separating the recorder devices from the recording itself, it becomes easier to replace these recorders in the future.

Design Decision: We choose to use the pin-hole camera model, described in Section 9.6.1. This means that lens distortions need not to be considered. Furthermore, we use the straightforward linear calibration method described in Section 9.6.2, although this method is known to be somewhat sensitive to noise. We choose *not* to use the most advanced and complex methods, since accuracy of 3D data is not our highest priority.

7.2 Object design

7.2.1 Recording

Recording
record_video
record_ measurement
load
save
give_emgdata
give_forcedata
give_videodata
give_time_int
empty

The *Recording* manages the measurement and video data. The *record_video* and *record_measurement* operations start the acquisition process of video and measurement data. Furthermore, a *Recording* can be stored and retrieved from disk. This is achieved by calling corresponding operations from each *TimedDataset*. The *Recording* provides access to each of the *TimedDatasets* by means of the *give_emgdata*, *give_forcedata*, and *give_videodata* operations. The recording's begin and end time are given via the *give_time_int* operation. The time values are initially generated by the synchronization module (see Section 6.5.3).

Finally, the recording can also be emptied. A *Recording* is emptied by removing each of the *TimedDataSets*.

7.2.2 TimedDataSet

TimedDataSet
begin_time
sample_ frequency
set_begin_time
set_sample_ frequency
give_samplenr
{load}
{save}

A *TimedDataSet* represents a measurement of some kind of data via a sampling process. It is an abstract class: only its children can be instantiated (see Section 1.1 of the appendix on object oriented methods).

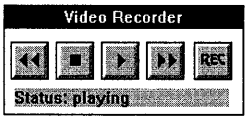
A *TimedDataSet* has a *begin_time*, indicating the time of the first sample. Furthermore, it has a (constant) *sample_frequency*. These two values can be set with the corresponding set operations. The *give_samplenr* operation computes the sample number given a time value.

The *load* and *save* operations are abstract functions to store and retrieve data from disk.

7.2.3 VideodataRecorder

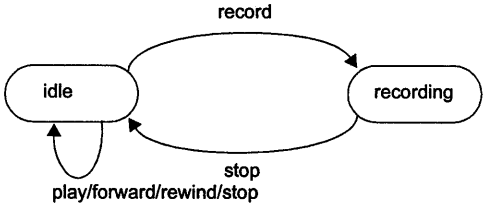
Videodata Recorder
play
forward
rewind
stop
record

The *VideodataRecorder* is the object that takes care of digitizing an analog video signal from a video recorder. It has the following appearance:



As can be seen, the *VideodataRecorder* is similar to the *ControlPanel* from the display system. The difference is that the control panel controls the time state of the *Recording* during a viewing session, while the *Videodatarecorder* controls a 'real-world' video recorder. Since the two classes belong to two different subsystems, and are similar only in appearance, we have chosen to not let them inherit from a common superclass. The record button starts the digitizing of the video.

Although pushing the play, forward or rewind buttons changes the state of the real world video recorder, the state of the *VideodataRecorder* object is not changed by these operations, except for the status line in the user interface element. However, recording does change the state, since only pressing the stop button is allowed:



The *VideoDataRecorder* uses the *VCRDriver* object from the sybar device interfaces subsystem (as described in Section 6.5.1). This object, which is not described in further detail, takes care of controlling the (real world) video recorder via software. The *VideoDataRecorder* also uses the *VideoDigitiserDriver* from the device layer to generate a window containing a 'live view' of the video. The recording of the video is performed by the *VideoData* object.

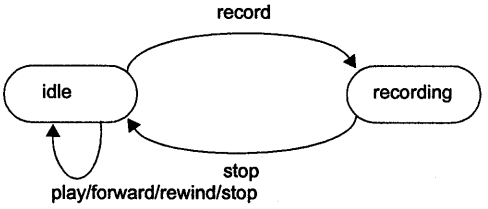
7.2.3 VideodataRecorder

Videodata Recorder
play
forward
rewind
stop
record

The *VideodataRecorder* is the object that takes care of digitizing an analog video signal from a video recorder. It has the following appearance:

As can be seen, the *VideodataRecorder* is similar to the *ControlPanel* from the display system. The difference is that the control panel controls the time state of the *Recording* during a viewing session, while the *Videodatarecorder* controls a 'real-world' video recorder. Since the two classes belong to two different subsystems, and are similar only in appearance, we have chosen to not let them inherit from a common superclass. The record button starts the digitizing of the video.

Although pushing the play, forward or rewind buttons changes the state of the real world video recorder, the state of the *VideodataRecorder* object is not changed by these operations, except for the status line in the user interface element. However, recording does change the state, since only pressing the stop button is allowed:

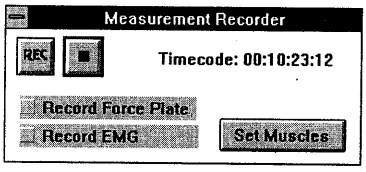


The *VideoDataRecorder* uses the *VCRDriver* object from the sybar device interfaces subsystem (as described in Section 6.5.1). This object, which is not described in further detail, takes care of controlling the (real world) video recorder via software. The *VideoDataRecorder* also uses the *VideoDigitiserDriver* from the device layer to generate a window containing a 'live view' of the video. The recording of the video is performed by the *VideoData* object.

7.2.4 MeasurementRecorder

Measurement Recorder
set_muscles
set_force_plate
set_emg
record
stop

The *MeasurementdataRecorder* is responsible for measurement of all the data, except the videodata. It has the following appearance:



Before a measurement is performed, the user selects the number of channels to be recorded, and in case of EMG measurements, provides the muscle names by selecting from a list of standardized names. These names are also used to identify EMG data in the in the display subsystem. There are record and stop buttons to start and stop the measurement operations.

The state diagram is similar to the *VideodataRecorder* state diagram, except that play, rewind and forward events do not exist.

7.2.5 EMGData

EMGData
add_channel
hide_channel
show_channel
set_value
give_value
set_color
give_color
set_name
give_name
load
save
empty
is_empty

An *EMGdata* object manages the EMG data of a measurement. EMG data is measured in a number of channels. Channels can be added with the *add_channel* operation. Channels can be 'hidden' or 'shown'. Hidden channels are not shown in any display.

Samples from channels are initially set with the *set_value* by the *MeasurementRecorder* operation and can then be retrieved by other (display) objects with the *give_value* operation. Values are retrieved by giving a time value. To convert the time value to a sample number, *EMGData* uses the *give_samplenr* from its parent the *TimedDataSet*.

An *EMGChannel* has a color, that is used for display and a name.The color and name of a channel can be set and retrieved with the corresponding *set* and *give* operations. The *EMGdata* object is emptied with the *empty* operation. To check whether the object contains any data, the *is_empty* operation is available.

7.2.6 EMGChannel

EMGChannel
name
color
hidden
hide
show
set_value
give_value
set_color
give_color
set_name
give_name
load
save

EMGChannels are containers for a single channel of EMG data, corresponding to the measurement of a single EMG electrode. Channels can be hidden or shown. If a channel is hidden, this means the user does not want see that particular channel in any display.

EMGChannels have a color and a name, corresponding to the muscle that was measured. The reason that the colors are stored with the data, is that they need to be consistent: a channel should have the same color in all the displays.

The color and name of a channel can be set and retrieved with the corresponding *set* and *give* operations.

7.2.7 ForceData

ForceData
set_value
give_value
set_color
give_color
load
save

The ForceData is a *TimedDataset* that manages data as recorded by a force plate device. The data consists of a vector and a starting point of the vector. Unlike EMG data, the number of channels is constant: 3 channels describe the x,y,z location of the starting point and 3 channels describe the 3 vector components. It is therefore not necessary to introduce a force data channel class.

Like the *EMGChannels*, the force data channels have a color. These are used by the *ForceGraph* class to draw single components from the force vector.

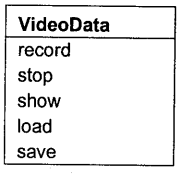
7.2.8 ForceDrawer

ForceDrawer
color
width
normalized_length
scale
set_options
give_options
draw

The *ForceDrawer* can draw a 3D force vector in a given 2D display. It is used by the *VideoDisplay* and the *KinematicsDisplay*. The *ForceDrawer* has several options: the color, width and normalized length can be set. Furthermore, a scale factor can be set to indicate the size of the drawing. The options can be set with the *set_options* operation and retrieved with the *give_options* operation.

The *ForceDrawer* gets its measurement data from the *ForceData* object. The *ForceData* object delivers 3D data in the world coordinate system. The 2D location in the image coordinate system is determined by calling the *project* operation from a given *Camera* object.

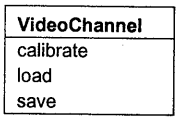
7.2.9 VideoData



The *Videodata* is a *TimedDataset* that records and manages video data. The *VideodataRecorder* sends a *record* message to start digitizing and a *stop* message to stop. The retrieval of data consists of showing images in a display. As a result, the operations are slightly different from the *ForceData* and *EMGdata* objects. The *show* operation displays a video image in a display at a given time. The frame number is calculated through the *give_samplenr* operation from the parent *TimedDataset*. In this case, a sample corresponds with a video frame.

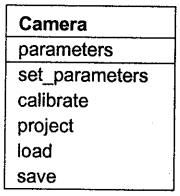
In Section 6.3.1, it is mentioned that a digitized video can be a combination of several camera images. We therefore introduce a set of *VideoChannels*, each corresponding to a camera view. It is not possible to view a single channel separately. However, the channels do need to be calibrated separately in order to perform 3D analysis.

7.2.10 VideoChannel



A *VideoChannel* corresponds with a stream of images produced by a video camera. To perform a 3D analysis, the parameters of the camera need to be determined. Camera information is stored in a *Camera* object. Each *Videochannel* therefore has its own *Camera*.

7.2.11 Camera



A *Camera* is an object that projects 3D points on to an image plane using the camera model of Section 9.6. A *Camera* may correspond to a real world video camera. There are, however, also 'virtual' cameras which can change position via user interaction.

Virtual cameras are used by the *KinematicsDisplay* (see Section 8.2.12) and use the *set_parameters* operation to set the viewing parameters. The parameters of cameras that have real world counterparts are determined by a *CameraCalibrator*, via the operation *calibrate*.

The *project* operation projects a point in 3D world coordinates to 2D using (EQ 1) and (EQ 2) from Section 9.6.1.

7.2.12 CameraCalibrator

CameraCalibrator
3D world points
image points
set_world_point
mouse_click
calibrate

In Section 9.6.2, a straightforward method for camera calibration is described, based on a camera model with 11 parameters. It is mentioned that using this camera model, a camera can be calibrated with six points whose world and image coordinates are known. The world coordinate points can be set with the *set_world_point* operation. The image points are set by mouse clicks.

The *CameraCalibrator* determines the 11 parameters of the camera model. The solution to the set of equations is calculated, in a least square sense, using the SVD algorithm (see 10.5.1). Although in principle the camera calibration can be done with six points, more accurate results are obtained when a larger number of points is used.

Design Decision:

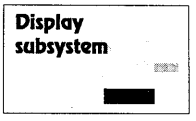
For the purpose of calibration, a special calibration object has been constructed. This rigid calibration object has a number of markers whose coordinates are known. For the lab operator, camera calibration involves positioning the calibration object, making a camera image of the scene, and indicating the location of the calibration points in the image. The image coordinates can be entered via clicking on the video image, which is presented for this purpose. One this has been done, the camera calibration is performed automatically.

For the presentation of the video image, the *CameraCalibrator* uses a *VideoDisplay* object from the display subsystem. For this purpose, *CameraCalibrator* inherits from *VideoDisplayUser*. The reason for introducing a special *VideoDisplayUser* class in explained in Section 8.2.10. Being a *VideoDisplayUser* allows the calibrator to receive mouse click messages from the *VideoDisplay*. Furthermore, results of the calibration process can be shown in advance by drawing the calibration points in the video display.

In order for the camera calibration to take place, the camera should be stationary. A moving camera can not be calibrated with the *CameraCalibrator*.

CHAPTER 8

Display Subsystem



The display subsystem is responsible for the display of measurement results, and the interaction that takes place during a viewing session.

8.1 System design

The main function of the display subsystem is to visualize measurement data in a way that is useful for physicians (as described in Section 3.2). The measurement data consists of:

1. Video, EMG and force data provided by the recording subsystem.
2. Kinematics provided by the kinematics system

Furthermore, as described in Section 3.3, the visualization is interactive and should be easy for novice computer users.

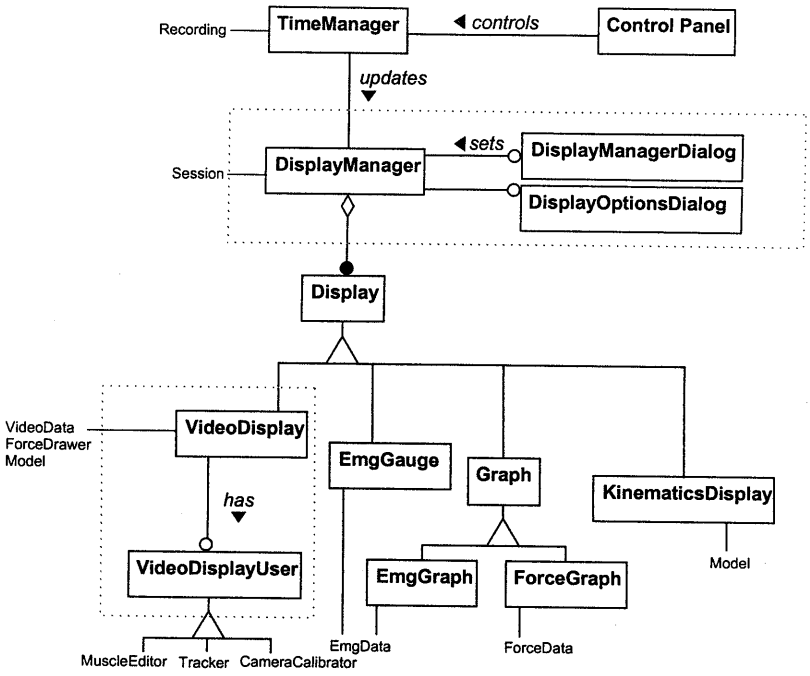
The display subsystem uses the recording subsystem and the kinematics subsystem. From the recording, the specializations of the *TimedDataSet* are used to display the correct measurement data. From the kinematics system, the *Model* is used to display kinematics data.

We again start by giving the complete object model. The display system object model is an detailed version of the display part of the

analysis model described in Section 5.2. The elements from this model and their relations are described in this chapter. The individual objects are described in the next section.

The object model for the display subsystem is shown in Figure 18. It shows the classes concerned with the display, and their relations.

FIGURE 18
Object model for display subsystem



8.1.1 Managing the display configuration

In the Section 4.4.5 of the specification, it is stated that the user of Sybar must be able to choose his own display configuration. Furthermore, the user must be able to change the display configuration at any time. It should also be easy to extend Sybar with additional *Display* classes (see Section 4.5). At this point, we extend the idea of a display configuration, described in Section 5.2.2 of the analysis, to a *DisplayManager*. The *DisplayManager* can create and delete *Display* objects. Furthermore, it can *hide*, *show*, *update*, *store* and

retrieve Display objects. The user operates on the *DisplayManager* via a *DisplayManagerDialog* window.

8.1.2 Timing and control

The user of Sybar must be able to view the recording in way similar to a video recorder (see Section 4.4.4). This means that the recording is in certain position. For this, the term *time state* was introduced in Section 4.4.1 of the analysis. The user can manipulate the time state via the *ControlPanel*. The *ControlPanel* consists of the standard *play*, *forward* and *reverse* buttons. It also has a scrollbar for direct feedback of the position and quicker access. There is also the possibility to repeatedly replay a recording via the *loop* button.

As was mentioned in Section 6.5.3, the recording system has the information that is necessary to synchronize the different data streams. The display system, however, has to take care of generating the updates when a recording is shown to the physician.

For the timing aspects of the display system we introduce a special class: the *TimeManager*. The *TimeManager* keeps track of the time state. When the time state is changed, the *TimeManager* sends an update message to the *DisplayManager* and to the *ControlPanel*. When the recording is played, the *TimeManager* periodically sends update messages, until the end of the recording is reached. The *ControlPanel* also needs to be updated because of the scrollbar feedback.

Design Decision: An alternative to a single object (the *TimeManager*) that controls the time, is that each *Display* takes care of updating itself periodically. This corresponds more directly to the dynamic model of the displays in the analysis. The advantage of a separate object to control the time is that it is easier to abstract from the operating system and that it requires less communication with the operating system clock functions. In some operating systems, including the one Sybar has been implemented for, there is an upper limit to the amount of timers that can be set. We have therefore chosen to put timing into a single object, the *TimeManager*.

A result of this decision is that the dynamic model of the *Display* class is simpler than described in the analysis phase. The *TimeManager*'s dynamic model (see the next chapter, object design) is now similar to the *Display*'s dynamic model from the analysis phase.

Design Decision: The *ControlPanel* is very similar to a *Display* object. The *ControlPanel* needs to be updated when the position of the recording changes (for the scroll bar). Furthermore, some display objects have the possibility to change the position too (in particular the *Graph* displays). However, we have chosen to keep the *ControlPanel* separate (i.e. not let it inherit from *Display*), since there are some differences between the two: The *ControlPanel* should be visible all the time and can not be deleted. More fundamentally, the *ControlPanel*'s main function is to let the user manipulate the time state, while the *Display's* main function is to show information.

8.1.3 The displays

The *Display* class is an abstract class with abstract operations that every child of *Display* should have. This means that only *Display*'s children can be instantiated. The following *Display* classes are available: *EmgGauge*, *EmgGraph*, *ForceGraph*, *KinematicsDisplay* and *VideoDisplay*. These classes correspond to the display types described in Section 4.4.1 of the specification. The *Displays* are largely independent. They receive messages from the *DisplayManager*. Furthermore, for the display of actual data, they are associated to objects from the recording subsystem. Display options for the *Displays* can be set by the user via the *DisplayOptionsDialog*.

The *Emggraph* and the *Fpgraph* both inherit from *Graph*. The *Emggraph* and *Forcegraph* provide the functions to give data values (*give_value),* and to determine the color of the graphs (*give_color*). All the other aspects of maintaining a graph display are taken care of by the *Graph* class.

The *VideoDisplay* is a relatively complex *Display*. It uses support objects from other subsystems: the *ModelDrawer* from the kinematics subsystem draws EMG data in the video image for each video channel using the *Model* (see the kinematics subsystem). The *ForceDrawer* from the recording subsystem draws the force plate data into the video image, using a calibrated *Camera* object for each video channel (see the recording subsystem).

8.2 Object design

8.2.1 TimeManager

TimeManager
time state
speed
play_start
status
frequency
stepsize
play
move
loop
stop
reset
give_status
step_next
step_previous
set_stepsize

The *TimeManager* takes care of timing issues. The *TimeManager* is associated with a *Recording*. The user can navigate through a *Recording* via the *time state*.The time state is stored in the *TimeManager*. The *TimeManager* provides services to other objects to change the time with the operations *play*, *move, loop* and *stop*. The time state is measured in milliseconds. The *TimeManager* can play at different speeds. The *speed* can be set via a parameter of the play operation. A negative speed results in a recording that is played backwards, a speed of one is the normal speed. The *reset* operation resets the time state to the start of the *Recording*, the *begin_time*.

Another attribute is the *status* of the *TimeManager*. The status can be either playing or idle. The status can be requested using the function *give_status*. If the *TimeManager* is playing, it automatically calls the update functions of the associated *DisplayManager* and *ControlPanel frequency* times per second and it updates the position. The frequency can be changed with *set_frequency* operation. If the end of the recording is reached, the *TimeManager* stops the updating process.

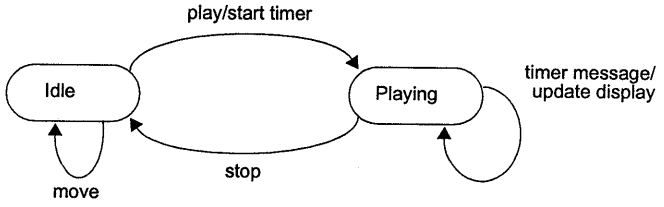
In order to generate periodic updates, we use a timer function provided by the operating system. As a parameter to the update functions, the time state of the recording is given. Since other objects only have to deal with these time state values, the operating system timer calls are encapsulated in the *TimeManager*.

The attribute *play_start* stores the time when the playing started. It is used to calculate the time state when an update is sent during playing. The time state is calculated by requesting the current time from the operating system. The time state needs to be recalculated because a single processor multi-tasking operating system can not guarantee that a function will be given control at exact time points. This means that although the frequency of updates may be lower than the specified frequency (i.e. some frames may be skipped), the individual updates of the display are as accurate as possible.

The *TimeManager* plays a recording within the time interval provided by the *Recording*.The operations *step_next* and *step_previous*

can be used to move a single frame. The length of a single frame can be set with *set_stepsize*. For example, if the frame rate is 25 frames/second, the step size is set to 40 milliseconds.

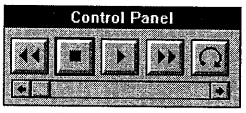
The dynamic model of the timer consists of two states:



8.2.2 ControlPanel

The *ControlPanel* is a user interface object with the following appearance:

ControlPanel
move
play
stop
forward
reverse
loop
update
save
load



The main purpose of the control panel, as described in Section 4.4.4, is that it allows the user to navigate through time. Navigation through time is achieved by manipulating the time state, via messages to the *TimeManager*. The messages are send as a response to user actions via the scroll bar or via one of the buttons.

The *ScrollBar* is a separate object that sends *move*-commands to the *ControlPanel*. The *ScrollBar* also functions as a feedback for the position of the recording: during playing, the *ScrollBar* is updated to the current position. No further description of the (low-level) *ScrollBar* class is given.

If any of the buttons, *play*, *stop*, *forward*, *reverse* or *loop* are pressed, the corresponding message is send to the *TimeManager*. In order to keep the scroll bar up to date, the *ControlPanel* has an *update* function which is called by the *TimeManager*. Finally, the screen location of the *ControlPanel* is stored and retrieved together with the display configuration. For this purpose, the *ControlPanel* has *save* and *load* operations.

8.2.3 DisplayManager

DisplayManager
dispnr
create_display
show_display
hide_display
delete_display
update
save
load
open_dialog
empty

The *DisplayManager* takes care of managing the display objects. Management includes creating, showing, hiding, and deleting objects. The *create_display* operation is used to create new displays. A parameter is used to indicate which type of display is to be created. Furthermore a unique ID, *dispnr*, is given in order to be able to distinguish the displays. All other parameters are determined in the constructor of the specific display object.

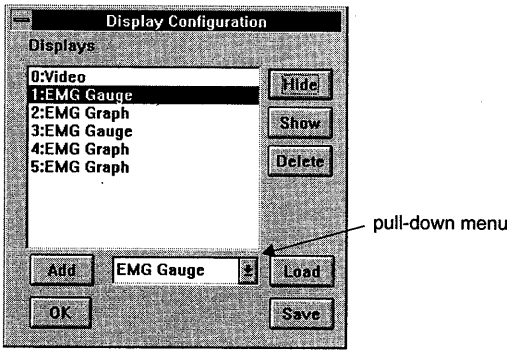
When a recording is being played, the *DisplayManager* broadcasts an update message to all the *Display* objects. The *Display's update* operations have the time state (provided by the *TimeManager*) as a parameter. This means that neither the *DisplayManager*, nor the *Displays* need to keep track of the time. The *DisplayManager* can also be requested for an update by other objects via its own *update* operation. This operation is called when the user changes display options, and the displays need to be redrawn.

A display configuration (including window locations) can be saved or loaded. The *DisplayManager* receives user-messages via the *DisplayManagerDialog* class. This dialog is created with the operation *open_dialog*, which is called by the *MainMenu*. The *empty* function is provided to clear the entire display configuration.

8.2.4 DisplayManagerDialog

The *DisplayManagerDialog* is the direct interface between the user and the *DisplayManager*. It has the following appearance:

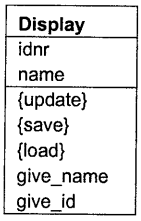
DisplayManager Dialog
add_button
delete_button
hide_button
show_button
ok_button
save_button
load_button



The main purpose of the *DisplayManagerDialog* is that it allows the user to navigate through data, as described in Section 4.4.5.

The *DisplayManagerDialog* shows the list of the current *Display* objects. New displays can be created by selecting a type from the pull-down menu and pressing the *Add* button. Displays can be deleted, hidden, or shown by first selecting an element from the list and then pressing the relevant button. For the list, a *ScrollList* (library) class is used. Display configurations can also be saved or loaded.

8.2.5 Display



The *Display* class is an abstract class. This means that only children of the *Display* class can be instantiated. These children provide implementations for the abstract operations of Display (abstract operations are shown in braces in the diagrams). *Displays* are used to show data from *TimedDataSets* from the data acquisition subsystem. *Display*'s children have to provide the following services:

- A *Display* should have an *update* function, which will be called whenever the display needs to be updated.
- A *Display* should be able to *save* and *load* itself, as part of the display configuration.

Displays are created by the *DisplayManager*. Information on what type of Display should be created is provided by the user in two ways:

1. via the *DisplayManagerDialog*. In this case the user can select options in an additional dialog that is performed during the construction of the display.
2. via a display configuration file. In this case the display must read its own parameters and options from the file.

The *give_id* operation returns the *idnr*, and *give_name* the *name* of the *Display*. The *idnr* is a unique number given by the creator of the *Display*.

8.2.6 Graph

Graph
scale
origin
x_axis
y_axis
n_channels
{give_value}
{give_color}
{load}
{save}
update
resize
mouse_button
pix_to_time
time_to_pix

A *Graph* is a *Display* that shows measurement data in the form of a graph. The graph display is specified in Section 4.4.1. The *Graph* class is also an abstract class. It can show a number of channels of integer data. The *n_channels* attribute gives the number of channels to be displayed.

The abstract operation *give_value* gives a value in the graph for a given channel and a time. Furthermore, the abstract operation *give_color* gives the color to be used for drawing the data channel. There are also operations to *load* and *save* the *Display* for storing and retrieving the display configuration. Using these abstract operations (provided by the children), the *Graph* class takes care of the *update* requests from the *DisplayManager*.

The size of a graph display can be changed by the user via the *resize* operation, dealt with by the window system. The *Graph* has attributes that determine the scale, and the location of the *origin*, the *x-axis* and *y-axis* of the graph. When the user resizes the display, the *resize* operation is called by the operating system. The *resize* operation then recalculates the scale and origin attributes.

The *mouse_button* operation takes care of mouse clicks of the user in the graph. As specified, the mouse click is interpreted as a command from the user to move the time state to the position where the mouse was clicked. For this purpose, the *Graph* is associated with the *TimeManager* object. When a mouse click message is received via the operating system, the screen location is transformed to a time state value, and the *move* operation of the *TimeManager* is called.

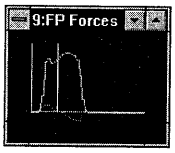
For conversions between graph coordinates and time state values, the operations *pix_to_time* and *time_to_pix* are used.

8.2.7 ForceGraph

ForceGraph
give_value
give_color
save
load

Forces are measured by a force plate device. The *ForceGraph* is a *Graph* that shows graphs of the x,y and z component of the force

vector through time, in the coordinate system of the force plate. All three channels are shown in the graph:

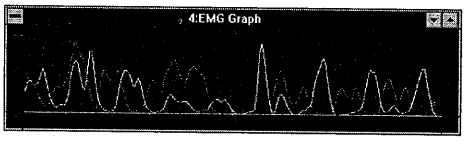


The operations *give_value* and *give_color* are performed by calling the corresponding operations from the *ForceData* object of the recording subsystem. If the force data is not available, the graph is left blank. Since there are no further options that the user can set, a dialog for initialization is not necessary. Also note that the *Force-Graph* does not have an update operation: updates are taken care of by the *Graph* class.

8.2.8 EmgGraph

EmgGraph
give_value
give_color
save
load

EMG data provides information on electrical activity in muscles:



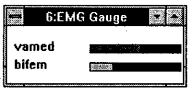
A number of channels of EMG can be measured in a single recording. The *EmgGraph* is a *Graph* that shows a number of channels of EMG in a single graph. The choice of channels to be shown in a single graph display is up to the user. For this purpose, a dialog is performed during the creation of *EmgGraph*. The muscles that are selected by the user are shown in the graph. It is not possible to change the selection of the muscles, once a graph is created.

The operations *give_value* and *give_color* are performed by calling the corresponding operations from the *EmgData* object of the recording subsystem. The channels that are not available are not drawn in the graph.

8.2.9 EmgGauge

EmgGauge
update
mouse_button
save
load

The *EmgGauge* display, specified in Section 4.4.1, shows gauges (bars) that indicate the current values of a number of EMG channels:



When the *TimeManager* is playing, the EMG gauges therefore show changing levels of muscle activity. As with the *EmgGraph*, the choice of the channels is up to the user and the data is provided by the *EmgData* object. The colors are also provided by the *EmgData* object. This ensures that an EMG channel has the same color in all the displays.

Next to displaying EMG values (via the *update* operation), the gauge also shows the correspondence of muscle names with muscle colors. A final feature of the *EmgGauge* is that individual EMG channels can be 'shown' or 'hidden' by clicking on them, see Section 4.4.5. The *mouse_button* operation translates this user action to a request for the *EMGData* object to hide or show a channel. Hidden channels are not shown in any display, except the *EmgGauge*, were they are shown as grey bars. Hiding channels allows the user to temporarily focus on a selection of the EMG channels. This feature was added with the 'easy navigation through data' goal in mind (see requirement 2 in Section 3.3).

VideoDisplay
options
default_user
update
resize
mouse click
set_options
give_options
user_request
user_release
save
load

8.2.10 VideoDisplay

As described in Section 4.4.2 of the specification, the *VideoDisplay* is the most important display of Sybar. It shows the video of the patient, annotated with measurement data. The *update* operation calls the *update* operations of the objects responsible for drawing a part of the annotation: the *ModelDrawer* and the *ForceDrawer*.

The *VideoDisplay* can be resized by the user, via the resize functions of the operating system. The *VideoDisplay* receives and interprets these window messages via the *resize* operation. The user can also set some options via a small dialog that is started by double clicking the mouse: kinematics, force data and EMG data can be turned on or off for the display.

The *VideoDisplay* is used by other objects: the *MuscleEditor (see* Section 11.4.3*)*, the *CameraCalibrator (see* Section 7.2.12*)*, and the *Tracker (see* Section 11.5.7*)*. These objects need to receive mouse events, and they also need to draw in the display (for feedback). A single display window is used for more purposes because bringing up a new window for each purpose would be confusing to the user. However, the control of the *VideoDisplay* needs to be managed in some way.

Design Decision:

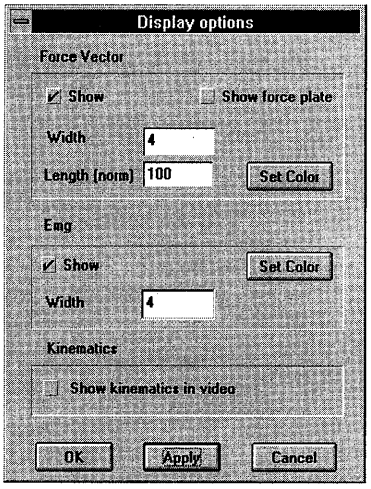
One option would be to let the *VideoDisplay* determine which object should have control, and send the messages to this object. This approach leads to a large number of unnecessary 'if' statements and associations. A better (truly object oriented) approach is to define a new abstract class: The *VideoDisplayUser*. The *VideoDisplay* only knows that events have to be sent to this *VideoDisplayUser*. The classes that need to use the display can then be made children of *VideoDisplayUser*. This is the approach that we have followed.

The *MuscleEditor* is the default user and *CameraCalibrator* and *Tracker* objects can request to gain control via the *user_request* operation. Control is released via the *user_release* operation. The default user is stored in the *default_user* attribute.

8.2.11 DisplayOptionsDialog

DisplayOptions Dialog
ok_button
apply_button
cancel_button
set_color_force
set_color_emg

The *DisplayOptionsDialog* allows the user to set options for the displays:



The following options are available:

- The color and width of the EMG and force vector annotations can be set.
- The normalized length of the force vector can be set.
- The force vector and EMG visualization can be switched on or off.
- The kinematics in the video display can be shown or turned off.

The force vector is expressed in Newton, which has no geometrical meaning. Therefore, although the length of the vector is relative to the measured force, the absolute length of the vector has no meaning in terms of pixels in the image. The parameter that determines the length of a vector in pixels can be set by the user, and is called the *normalized length*.

The user interaction of the dialog itself is the responsibility of the window manager. When the *OK* or *Apply* button is pressed, the values from the dialog are send to the *VideoDisplay*. The difference between *OK* and *Apply* is that the dialog remains open when *Apply* is pressed, and that the dialog is closed when *OK* is pressed. The

Set Color buttons bring up a standard color selection dialog. When a color is selected, it is send to the corresponding objects. To realize the options, the *DisplayOptionsDialog* communicates with the *ForceDrawer and EMGData* from the recording subsystem, and the *ModelDrawer* from the kinematics subsystem.

8.2.12 KinematicsDisplay

KinematicsDisplay
alfa
beta
radius
update
mouse_event
key_event
resize
load
save

The *KinematicsDisplay*, specified in Section 4.4.3, uses the *Model* (see Section 11.4.2) to draw the kinematics via the *update* operation. It has its own *Camera* object (see Section 7.2.11) which determines the point of view. The camera is located on a sphere and set via two angle parameters: *alfa* and *beta*. The angles alfa and beta can be changed by first pressing and then moving the mouse horizontally (to change alfa) or vertically (to change beta), as in Figure 19. The center of the sphere is the average of all the model points (the gravity center).

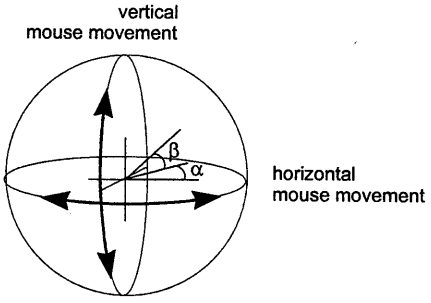


FIGURE 19
Navigation via mouse dragging

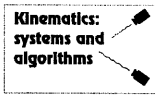
Further interaction is possible via the keyboard:

- Pressing <shift> constraints the movement to either horizontal or vertical movement (the largest movement is chosen).
- Pressing <ctrl> in combination with vertical movement changes the focal distance of the camera, which gives the possibility to zoom in and out.
- Pressing <tab> in combination with vertical movement changes the *radius* of the sphere, which gives the possibility to move closer or further away from the center.

The camera orientation is always directed towards the gravity center. The *KinematicsDisplay* can be resized, loaded and saved.

CHAPTER 9

Kinematics: Systems and Algorithms



Human kinematics can be measured in a large number of ways. This chapter first gives an overview of the devices and algorithms that are used. Next, the kinematics detection approach for Sybar is chosen.

9.1 Introduction

Kinematics describe the motions of the human body, or part of the human body, through time. Kinematics are used to analyze human motion, both in medical applications and in other fields. Kinematic descriptions are always based on a human model. There are a number of devices and methods that are used to detect human kinematics.

9.2 Modeling human motion

The human body is an immensely complex system. Any description of human motion is necessarily a simplification. Therefore, a description of human motion requires some kind of model. A very common simplification is to describe human motion by assuming that the body consists of a number of connected *rigid body parts*. A rigid body part consists of particles that have a fixed distance to each other. The rigidity assumption is based on the fact that the human skeleton indeed consists of relatively rigid bones.

A very common rigid model is the *stick figure model,* also known as the *linked segment model.* In the stick figure model, the complex joints between bones are modeled via simplified joint models (usually hinge joints), and bones are represented by line segments. Furthermore, the number of bones (segments) is reduced. A very common stick figure model is the 15 segment model of Figure 20. The spine, a complex part of the human skeleton, can not easily be modeled with hinge joints.

FIGURE 20
Stick figure model

Sometimes, cylinders or other solid shapes are used to represent segments consisting of both bones and soft-tissue parts. This means that it is assumed that entire body segments are rigid. However, soft-tissue parts of the body are not rigid, and therefore these models have to be used with care. Solid models are useful for kinematics detection by means of model matching. This is the topic of Section 9.8.

Modeling human motion involves two kinds of parameters:

1. Position-independent parameters. For example, segment lengths are usually assumed to be constant during motion, and are therefore position independent.
2. Position-dependent parameters. These are the parameters that determine the posture of the model. For example, most models have angles as position dependent parameters.

For some applications, a 2D model is used. Applications with 2D models usually assume that the motion of each body segment is restricted to a fixed plane.

Once kinematics are available, they can be visualized in various ways. For example, the segments of a stick figure model can be visualized by drawing lines or cylinders. This chapter, however, deals with the detection of kinematics, and not with the visualization of kinematics.

9.3 Detection of kinematics

The detection of the kinematics of human motion is a difficult problem. The dynamic position of the skeleton, on which the rigid human models are based, cannot be detected easily. A radical solution to this problem was used by the California group in the fifties (see Section 2.1), where metal pins were attached to the skeleton near the joints under local anesthesia.

A more practical, and very often applied method, is the use of markers, attached to the skin. Joint positions are estimated from these marker positions. A problem with the marker approach is the fact that the skin moves relative to the skeleton. Marker approaches are usually used in combination with stick figure models. Some alternatives to the marker approach have been suggested in the field of computer vision These will be described in Section 9.8.

The following section describes the types of devices and systems used for the detection of kinematics.

9.4 Detection devices

There are a number of devices, based on various principles, that are used for the detection of kinematics. In this section, we give an overview of the most often used types. Every system has its own advantages and disadvantages. There are also differences in accuracy. In the specification of commercial marker systems, the accuracy of the marker detection is given, not taking into consideration the errors that are introduced by skin movements. Commercial marker systems claim accuracies within the range of 1 mm. However, skin movements of markers attached to anatomical landmarks during walking are in the range of 10-30 mm [18].

9.4.1 Camera based systems

Camera based systems register analog or digital 2D images. These images can be used to detect kinematics. It is very difficult to detect kinematics directly from camera images. Therefore, the commercial systems require the subjects to wear markers.

Two types of markers are used: active markers and passive markers. Active markers send light themselves, while passive markers only reflect light. Active markers are easier to detect, whereas passive markers are easier to use, since no electronic wiring has to be attached to the patient. Some older (passive marker) systems require a human operator to locate the position of the markers in the images. Most modern systems are able to perform marker detection semi-automatically, using image processing techniques.

Another problem that has to be solved with the marker approach is the *correspondence* or *identification* problem. It is not enough to detect the markers in the image, they also have to be matched with actual model points: for example a knee marker has to be identified as being a knee marker, not a hip marker. In the cases where the motion is complex or when a large number of markers are used, this can be a difficult problem.

The following camera systems can be distinguished:

- *Photogrammetry systems* (photo-camera): the first kinematic systems were based on photogrammetry and cinematography. Some are still in use today because of their high accuracy. They usually require a human operator for marker detection.
- *Opto-electronic systems* (television/video): these systems are much cheaper and easier to use than photogrammetry systems. Detection of markers is performed either manually or automatic.
- *Infrared systems*: there are both active and passive systems that work with infrared light. The active systems use markers that send a flash of infrared light one at a time. An infrared camera then determines the average location of all incoming infra red light. An advantage of this approach is that the identification of markers is performed automatically, since the markers are detected one at a time in a specific order. A disadvantage of this approach is that it is sensitive to reflections of markers on the floor or other objects [13].

In general, camera devices deliver images that need to be processed for acquiring kinematics. The computer processing of camera images is the topics of the field of *computer vision.* Section 9.7 deals with the retrieval of kinematics from camera images.

9.4.2 Opto-electronic scanners

The Coda 3 system is an opto-electronic scanner. It scans the measurement area using a very fast moving plane of light, controlled via mirrors. Retro-reflective markers reflect colored light, and are detected. The colors are used to identify the detected markers. Because every marker has to have its own color, only a limited number of markers can be used (in the Coda 3 system there is a maximum of 12 markers). Three scanners are used to determine 3D coordinates [13].

9.4.3 Acoustic systems

Acoustic systems register active sound producing markers. Several microphones are used to determine the 3D position.

9.4.4 Electrogoniometers

Electronic goniometers are devices that are attached to the body and measure angles of joints. Goniometers give a limited description of the kinematics: most systems only measure knee angles. Furthermore, electrogoniometers are difficult to calibrate [11].

9.4.5 Inertial sensors

Accelerometers and solid state gyroscopes are electrical devices that are attached to the body and measure acceleration. They are relatively small, and therefore much easier to use than goniometers. From the acceleration, kinematics can be determined via integration, although no absolute angles can be determined. Because of technical limitations, these systems are not widely used [13].

9.4.6 Magnetic tracking devices

Another method to determine the 3D coordinates of a marker is by means of a magnetic tracking device [10]. The 3Space Isotrak system from McDonnell Douglas uses low-frequency magnetic fields which are send by active markers and detected by sensors. The system is sensitive to metallic objects in the scene.

9.5 Detection device for Sybar

Sybar uses video for the visualization of human motion. There are several reasons why we prefer to use the video information that is already available to detect kinematics over using an additional kinematic detection device. Using an additional kinematic detection device causes the measurement system to become a lot more complex, since such devices have to be integrated with the other measurement devices. Complex measurement settings are user-unfriendly for the operator, and are also less patient-friendly.

Furthermore, using existing commercial systems has the disadvantage that:

- they are not 'open': it is very difficult to integrate them together with Sybar in a single system.
- they are expensive. Not using them reduces the total cost of Sybar.

Design Decision: We therefore choose to develop a proprietary kinematics detection system, based on the image processing of video images.

We first discuss some basics of computer vision. Next, we review the research on kinematics detection that has already been done. Finally, we choose an approach for Sybar.

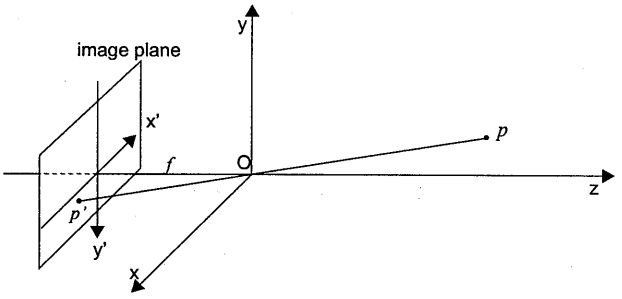
9.6 Computer vision basics

The field of computer vision is concerned with the recovery of useful information about a scene from its two-dimensional projections [75]. Computer vision deals with digital images, that are acquired by digitizing analog images directly from a camera or via a recording device. Digital images consist of two dimensional arrays of color or grey values. Points in these arrays are called *pixels*. In addition to image (or pixel-) coordinates, an arbitrary image and view point independent coordinate system, known as the *world coordinate system*, is often used.

9.6.1 Camera model

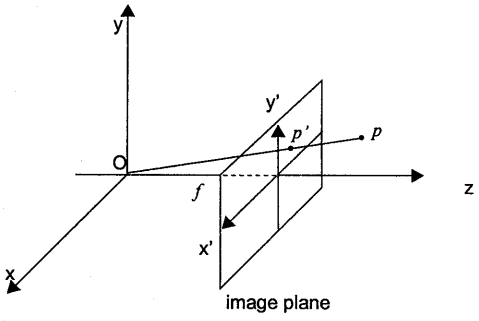
For images generated with cameras, the *pin-hole* model is usually used. A point p is projected via the centre of projection to the image plane point p', as in Figure 21.

FIGURE 21
Camera model: image plane behind center of projection



This model does not take lens-distortions into consideration. Also note that the 2D coordinate system of the pixels is reversed with respect to that of the camera. For this reason, it is customary to assume that the image plane is in front of the centre of projection instead of behind, as in Figure 22.

FIGURE 22
Camera model: image plane before center of projection



Using a world coordinate system, the following parameters determine the image produced by the camera:

- camera position in space (X_0, Y_0, Y_0)
- camera orientation given by a rotation matrix R, with elements r_{ij}
- focal distance: f

Given a point (X,Y,Z) in world coordinates, its projection (x,y) is given by

$$x = \frac{1}{f} \cdot \frac{r_{11}(X-X_0) + r_{12}(Y-Y_0) + r_{13}(Z-Z_0)}{r_{31}(X-X_0) + r_{32}(Y-Y_0) + r_{33}(Z-Z_0)} \quad \text{(EQ 1)}$$

$$y = \frac{1}{f} \cdot \frac{r_{21}(X-X_0) + r_{22}(Y-Y_0) + r_{23}(Z-Z_0)}{r_{31}(X-X_0) + r_{32}(Y-Y_0) + r_{33}(Z-Z_0)} \quad \text{(EQ 2)}$$

9.6.2 Camera calibration

Computer vision algorithms often assume that the camera parameters are given. There are also special *camera calibration* algorithms to determine the camera parameters. Camera calibration algorithms need 'world knowledge' to determine the camera parameters. Usually, this world knowledge is provided by means of a calibration object with a known structure. An image is then taken of the calibration object, and the camera parameters are determined from the calibration image.

The rotation matrix has three degrees of freedom. Therefore, the pinhole camera model has seven parameters: three for the camera position, three for the orientation and one for the focal distance. It is however not possible to determine these 7 parameters solving linear equations. It is therefore customary to use 11 parameters, based on regrouped versions of (EQ 1) and (EQ 2).

$$x = \frac{L_1X + L_2Y + L_3Z + L_4}{L_9X + L_{10}Y + L_{11}Z + 1}, \quad y = \frac{L_5X + L_6Y + L_7Z + L_8}{L_9X + L_{10}Y + L_{11}Z + 1}$$

The parameters $L_1, L_2, \ldots, L_{11}$ are called the DLT (direct linear transformation) parameters. A disadvantage of using more parameters then necessary is that the algorithm that is rather sensitive to noise on the input ([61]). However, using this redundant parameterization, we avoid non-linear equations.

Each calibration point, which is given in 3D world coordinates and 2D image coordinates leads to two equations. Therefore, to determine the 11 DLT parameters with linear equation solving, at least 6 points are necessary.

More advanced camera models also take lens distortions into consideration, which leads to more complex calibration algorithms. An overview of camera calibration algorithms is given by Tsai in [61].

9.6.3 3D reconstruction

Since the mapping from a world coordinate point p to an image coordinate point p' is a projection, it is not possible to reconstruct p solely from p': a point in 2D can be the projection of any point on the line through p-p'. To reconstruct a point in 3D, more information is necessary. This information can come either from having more projections of the same 3D point (for example by using more cameras or by using mirrors) or from having knowledge of the object that the point to be reconstructed is part of, in the form of a model.

9.7 General issues on retrieving human motion from images

The retrieval of human-motion kinematics from images is a topic that has received a lot of attention in the computer vision field. It is a very difficult problem, and a general solution has not yet been suggested. A distinction can be made between algorithms that use markers to determine the kinematics and those that retrieve kinematics by matching the image with a so-called matching model.

There are a number of issues that are common to all the algorithms. These will be described in this section. First, we describe the approaches that do not use markers. Next, we describe the approaches that do use markers.

9.7.1 The scene

Human motion takes place in a certain environment, also known as the *scene*. Algorithms that do not use markers usually require that the scene is static: the human subject is the only thing that moves in the scene. Another common requirement is that there is enough contrast between the subject and the background. Some algorithms require 'featureless' backgrounds (empty scenes).

9.7.2 Number of views and point of view

Some algorithms use multiple views, obtained via a number of camera's. Others retrieve kinematics from a single view. Processing of images also depends very much on the point of view of the camera. Two views are often used: the sagital (side) view and the coronal (frontal) view.

9.7.3 Motion and occlusion

Humans are capable of an enormous variety of motions. Most algorithms are restricted in that only a small subset of motions is allowed. Some algorithms focus on walking gait, others more on gymnastic motions. Another distinction that can be made, is that between algorithms that analyze individual images (posture detection) and algorithms that analyze motion through a sequence of images. A re-occurring problem is that body parts (including markers) are frequently obscured by other body parts, a phenomenon that is known as *occlusion*.

9.7.4 Appearance

Most algorithms make assumptions on what the subject looks like. Some algorithms require that the subject wears a certain outfit. A number of algorithms use the marker approach, where markers are attached near joints. Algorithms that are not based on markers, require the subject to wear tight clothes.

9.7.5 Models

Kinematics retrieval algorithms use a human model for their kinematic description. Both stick figure models and models consisting of solid primitives are used. Sometimes, a 2D model is used. There are also algorithms that result in the retrieval of only a part of the human body, for example only the upper part.

9.8 Approaches without markers

In this section, we discuss algorithms that do not require markers to be attached to the subject. Most of these algorithms work in two steps. First, some 'low-level' image processing is performed. Next,

the processed images are matched with an algorithm-specific human model, which we call the *matching model*. In Section 9.10, a classification of methods is given in Table 2.

9.8.1 2D stick figures of 'gymnastic motions'

Leung and Yang describe an algorithm that determines 2D stick figures of humans performing certain 'gymnastic motions' [29]. Their algorithm first determines an outline of the human for each image. These outlines are matched with a 'ribbon' model, from which the stick figure is determined. A single view is used for each image sequence. The point of view is chosen optimally for each motion.

9.8.2 2D stick figures of a walking person

Several approaches have been suggested for retrieving 2D stick figures from a sequence of images of a walking person [26,27,28]. Guo, Xu and Tsuji describe an algorithm that uses an image processing filter that determines a skeleton from a person's silhouette. This skeleton is matched with the stick figure model using neural networks [26,27]. Niyogi and Adelson determine an 'armless' stick-figure by searching for patterns in the spatio-temporal volume. This spatio-temporal volume is the complete XYT-space consisting of a number of images of one or more walking persons [28]. The algorithms both use a single view, namely the sagital view of the subject.

9.8.3 3D models of 'gymnastic motions'

In an early article on human motion retrieval, O'Rourke and Badler describe an algorithm that detects simple gymnastic motions from a coronal view [30]. It is based on a prediction-interpretation loop. Positions of body segments are predicted and the predictions are compared with images. Furthermore, constraints are used to rule out certain positions. The matching model used is based on 600 intersecting spheres. The algorithm was only tested on simulation data.

Akita describes an algorithm that was tested on real data [32]. In his algorithm, particular gymnastic motions, that have to be described in advance, are analyzed. The algorithm first determines outlines of figures, and next determines the positions of the body parts using the image outlines and a predicted outline. The matching model is based on generalized cones.

9.8.4 3D model of a walking person

Rohr describes an algorithm to retrieve a 3D model of a walking person from images using the sagital view [35]. First, an algorithm determines the difference between an image and its predecessor. Using this change detection filter, the part of the image that belongs to the walking person is determined.

Rohr uses a model that not only describes the human body, but also incorporates a model of human walking. The model of human walking is based on averages obtained from medical data of healthy persons. Using Kalman-filter prediction and comparison of predicted values with image values, the parameters of the model for each frame are determined. The matching model is based on cylinders.

9.8.5 3D model of upper body motion

Gavrila and Davis describe an algorithm that tracks upper body motion [36]. Their algorithm is unique in the sense that it uses a multi-camera approach without markers. It is again required that the scene is static. The matching model is based on 'tapered superquadrics'.

The algorithm searches the parameter space of the model using a heuristic approach (*'best-first'*), starting at a predicted value (based on previous frames). The similarity between model views and the actual images is determined with a image processing technique called *chamfer matching*.

9.9 Approaches that use markers

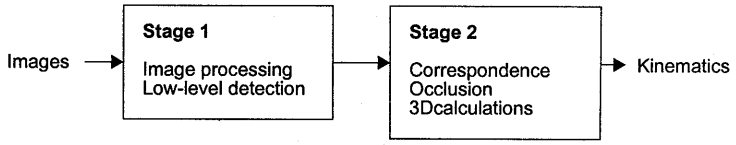
Commercial human motion analysis systems use markers that are attached to the human body. Most marker systems are model independent: the fact that the markers are attached to a mechanical system (e.g. a human subject) is not used. Usually, a two stage approach is followed:

1. Low-level detection of markers. The result of this stage is a set of marker-coordinates for an image. The first marker systems required a human operator to indicate the locations of the mark-

ers. Most modern systems can perform the detection automatically.

2. Combination of low-level results into a description of 2D or 3D trajectories of model points. For this purpose, the markers detected in stage 1 need to be matched with model points (correspondence). In some systems, there is a one-to-one relation between markers and joints. Other systems use sets of markers from which joint positions are calculated, to get a higher accuracy. Some systems are capable of correcting errors of stage 1 in stage 2. Some systems are also capable of estimating positions of temporarily invisible (occluded) points.

FIGURE 23
general approach for kinematics detection with the aid of markers



9.9.1 Low-level detection

Automatic low-level detection algorithms usually first convert the digitized grey or color images to a binary image using a *threshold* operation. A threshold operation sets a grey value pixel to 1 if it has a greater intensity than a given value T and to 0 if the grey value of that pixel is smaller than T. The value T is called the threshold.

The idea behind using a threshold algorithm is that markers generally have the highest grey level intensity in images. In the next step of the marker detection, an algorithm that searches for markers is applied to the thresholded binary images. The search algorithms search for shapes that resemble markers, for example using a method known as *template matching* [75]. In template matching, a template of the object is compared with a part of the image. The dissimilarity between the template and the image is usually defined as the sum of the squared differences between intensities of image points and template points. It is possible to achieve sub pixel accuracy by also testing template positions located in between pixels.

Although several improvements to this general approach have been suggested, none of the algorithms is perfect. Depending on the quality of the input images, markers can be missed, and other

objects can be incorrectly identified as being markers. Furthermore, markers that are not visible can obviously not be detected.

9.9.2 2D model using a single camera

From the results of low-level detection, a 2D model can be determined. For this purpose, the marker coordinates in each image need to be identified as being model points. This is known as the *correspondence* problem. Correspondence can be solved with tracking algorithms. These algorithms determine the trajectories of model points, using the fact that marker motion is smooth [43]. Some tracking algorithms also deal with occlusion [50,51]: invisible points can be estimated by interpolation between known values.

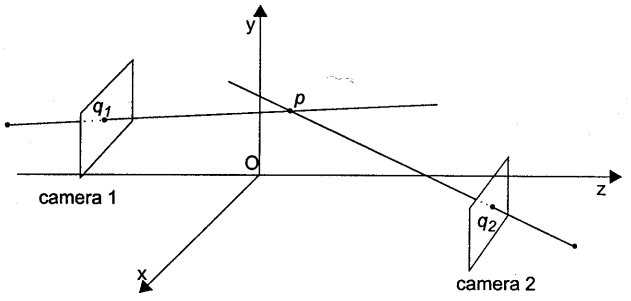
Errors in detection can be corrected by a human operator, resulting in semi-automatics systems. Occlusion can also be solved by a human operator.

Ferrigo and Gussoni describe an algorithm that determines a 2D model for several sport activities. They use a single leg, single arm model and a sagital view to determine correspondence and solve (limited) occlusion [24].

9.9.3 3D model using multi camera's

Most motion analysis systems are able to determine the 3D position of a marker by combining the results of several camera's. Assuming that the camera parameters are known, and that marker detection for each image has been performed without errors, it is possible to reconstruct the 3D position of a marker with two cameras, without using any additional information. Using the camera model of Figure 22, Figure 24 shows that a point p that is projected onto two camera image points q_1 and q_2, can be reconstructed by calculating the intersection of the rays from the centers of projection through the respective image points.

FIGURE 24
reconstruction of marker coordinates with two camera's

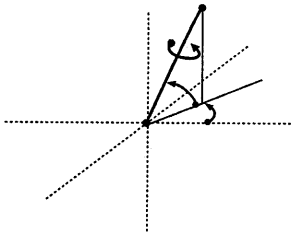


In practice, the camera model is not perfect and the marker position q_1 and q_2 can not be calculated without some errors. Therefore the two rays will not intersect and an approximation has to be made, for example the midpoint of the midperpendicular of both lines is taken as the intersection [9, 56].

Occlusion is often solved by using more than two cameras, to make sure that the marker is visible in a least two images. The correspondence problem has to be solved for every camera view. It is possible to solve the correspondence for each view separately. An alternative is to solve the correspondence by considering the data of all the views simultaneously. If the correspondence is correct, there is a 3D point that projects onto each of the pixels by applying the corresponding viewing transformation. In the case of imperfect data, the projections are close to the pixels.The 3D point can be reconstructed from the pixel data, by generalizing the 2D midperpendicular approach. For each pixel, the difference between the projection of the reconstructed 3D point and the pixel can be determined. The sum of these differences is an indication for the validity of the match.

A rigid body segment in 3D has six degrees of freedom: three for the position, and three for the orientation. The systems described so far, determine only two parameters for the orientation. The possible rotation of a segment around its own axis, as in Figure 25, is not considered.

FIGURE 25
The orientation of a segment has 3 degrees of freedom



When more than two markers are used for a single body segment, it is possible to determine the rotation of a segment around its own axis. The calculation of the position and orientation of a segment in 3D from a set of points on the segment, has been the topic of the studies reported in [55,57,58,59]. In particular, the accuracy and sensitivity to noise in marker data is addressed in these articles.

9.9.4 3D model using a single camera

Chen and Lee describe an algorithm to determine 3D kinematics of a stick figure from a sequence of single view images [33,34]. They assume that low-level marker detection is performed correctly. Furthermore, they assume that the correspondence problem and the occlusion problem are solved. Their algorithm uses model information and constraints to reconstruct 3D kinematics for a stick figure model. The algorithm was tested on synthetic data.

9.10 Classification of approaches

This section summarizes the previous two sections by giving a classification of approaches with and without markers to retrieve kinematics from images. The first row in Table 2 gives the literature reference of the approach. In two cases, the approach can not be directly related to a single publication:

- The detection of kinematics with markers using a 2D model and a single view, described in Section 9.9.2. This approach is shown in as [*] in Table 2.

- The detection of kinematics with markers using a 3D model and multiple cameras, described in Section 9.9.3. This approach is shown as [**].

TABLE 2 Classification of approaches to retrieve kinematics from images

Reference	[29]	[26],[27]	[28]	[30]	[32]	[35]	[36]	[*]	[**]	[33],[34]
Scene	static	static	dynamic	empty	empty	static	empty	dynamic	dynamic	dynamic
number of views	1	1	1	1	1	1	several	1	several	1
point of view	several	sagital	sagital	several	coronal	sagital	several	several	several	coronal
type of motion	gymnastic	gait (walking and running)	gait	gymnastic	gymnastic, prescribed	healthy gait	any upper part motion	any	any	gait
human appearance	different intensity from background	different intensity from background	no restrictions	equal to model	different intensity from background	no restrictions	different intensity from background	markers	markers	markers
model	2D stick figure	2D stick figure	2D stick figure	3D spheres	3D generalized cones	3D cylinder	3D tapered super quadrics	2D stick figure	3D stick figure	3D stick figure
part of body retrieved	whole	whole (armless when running)	armless	whole	whole	whole	upper part	whole or part	whole or part	whole
tested on real data	Y	Y	Y	N	Y	Y	Y	Y	Y	N

9.11 Approach for Sybar

Two requirement definitions that are important with respect to the kinematics subsystem:

- Ease of use for the lab operator (see Section 3.3)
- Patient friendliness (see Section 3.4)

An ideal kinematics subsystem is transparent to both the lab operator and the patient. Therefore, an approach that uses the video images that are recorded for the visualization, would be ideal.

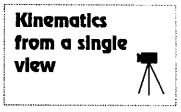
These video images are usually made with a single camera and do not contain markers.

However, when one reviews the algorithms described in the section on approaches that do not use markers, it is quite clear that these algorithms are still largely experimental. There are a number of reasons why they are not suitable for our application. First of all, we are primarily interested in 3D kinematics detection (see Section 4.3.1). Therefore, approaches that produce 2D stick figures are not suitable [26,27,28,29]. Furthermore, the algorithm should be able to detect kinematics in walking gait, since it the most important type of analysis for rehabilitation medicine (see Section 4.3). Therefore, approaches that do not deal with gait can not be used [30,32]. Finally, the motion patterns of patients typically do *not* resemble historical averages from healthy persons. Therefore, Rohr's algorithm is also unsuitable [35].

Design Decision: We therefore choose to use an approach that uses markers. The attachment of markers is a relatively small burden on the patient. Using such an approach, it is possible to choose either a single or a multiple camera approach. A single camera approach is to be preferred from the lab operator's point of view, since it means that no additional video equipment will have to be operated: the video that is recorded for the visualization can be used. In the following chapter, we therefore investigate the possibility of using a single view approach for Sybar. We use Chen and Lee's work as a starting point [33,34].

CHAPTER 10

Kinematics from a Single View



This chapter deals with the possibility of retrieving 3D kinematics from a single camera view for Sybar. First, Chen and Lee's approach is summarized and discussed. Next, the 3D reconstruction is formulated as a minimization problem, to be solved with the SVD-technique. Finally, some test results of the approach are presented and conclusions are drawn.

10.1 Introduction

Human motion takes places in three dimensions. Therefore, a human-motion analysis system should preferably be able to generate and work with 3D data. The commercial systems that provide 3D kinematics use a multi-camera approach. Multi-camera motion-analysis systems are expensive and cumbersome. In the Sybar project, a single camera video is already available for the purpose of visualizing the patient's movement. An approach that uses only this video information is therefore to be preferred, since it does not require any additional equipment.

In the field of computer vision, a lot of research has been done on acquiring 3D data from single-view images. In the previous chapter, we presented an overview of the kinematics detection literature and we investigated the possibility of using a single camera approach to retrieve 3D data for Sybar. It was concluded that the marker

approach followed by Chen and Lee [33,34], is the most suitable for Sybar. We will now describe their algorithm is greater detail.

10.2 Chen and Lee's approach to 3D reconstruction

In Chen and Lee's papers, an algorithm is described that reconstructs 3D stick figures from single view images, using the marker approach. In this approach, markers are attached to the human body. Joint positions are determined from these markers.

In Chen and Lee's algorithm, a number of assumptions regarding the input are made. It is assumed that the complete set of projected body joints that form the stick figure is available in each of the images. This means that:

1. Low-level marker detection has already taken place.
2. The *occlusion* problem is not considered: markers can become temporarily invisible if body parts obscure the view. Chen and Lee assume that this does not occur (or that perfect estimates of obscured marker points are available).
3. The *correspondence* problem has been solved. This means that each projected point is already identified as being the projection of a certain stick figure model point.

Furthermore, it is assumed that:

4. The length of all rigid segments is known.
5. Six points on the head are available. The head is considered to be a rigid object. The camera is calibrated (see Section 9.6.2) using the head as a calibration object.

In Section 9.6.3, it is mentioned that is not possible to reconstruct a 3D point p solely from its 2D projection. However, additional model information, in the form of segment lengths, can make this reconstruction possible. Assume that p and q are projected points of respectively P and Q. If p, q and P are given and the length l of segment PQ is given, then two candidates for Q can be reconstructed, by casting a ray from the centre of projection through q. The points of intersection of this ray with the sphere with radius l which has P as its centre are the candidates. This is shown in Figure 26.

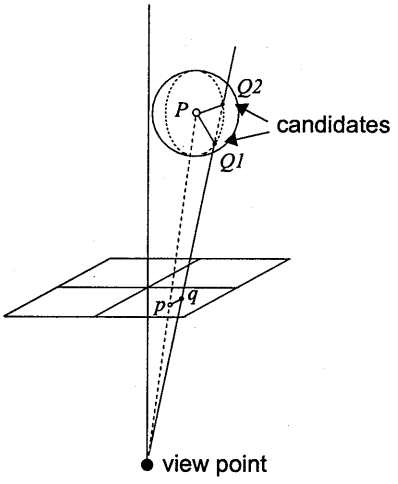


FIGURE 26
Reconstruction candidates

By choosing the correct candidate and repeating the procedure for the connected points in the stick figure, the entire model can be reconstructed. The choice between the two candidates will be called the *hither-yon* choice. As a starting point, the neck, which is one of the six markers located on the head, is used. If there are n model points outside of the head, there are n hither-yon choices to be made. This leads to 2^n possible stick diagram configurations, that should comply with the detected 2D marker positions.

The method of choosing the correct configuration is based on eliminating incorrect configurations, until only the correct configuration is left. Configurations are eliminated by checking a number of constraints:

- *Angle constraints*: the angles that segments can make in joints are limited. Chen and Lee define a number of angles in the human model, and use constraints on these angles to eliminate impossible configurations.
- *Distance constraints*: points on rigid objects have a fixed distance to each other. When more than two markers are attached to a single rigid object, this leads to additional distance constraints. In Chen and Lee's stick figure, this is the case at the pelvis. Furthermore, in invalid configurations, the rays that are cast as in

Figure 26, can miss the sphere. This can also be seen as a violation of distance constraints.

- *Collision-free constraints*: it is not possible that segments intersect with each other. This is expressed in the collision-free constraints.
- *Correctness rules:* Chen and Lee use additional information on gait, as performed by healthy persons. This is done in the form of a number of knowledge rules. For example, it is assumed that it is not possible that both arms are in front of or behind the torso simultaneously. These rules restrict the algorithm to gait of healthy persons.
- *Smoothness of motion:* In [33], constraints that can be applied to a single position of the stick figure are defined. In [34], the algorithm is extended to include information from multiple frames to eliminate incorrect configurations. The constraint that is added is based on the fact that human motion is relatively smooth: configurations that would result in unsmooth motion are not allowed.

Using these criteria, Chen and Lee find unique configurations for their simulation data.

10.3 Problems with Chen and Lee's approach

Chen and Lee's algorithm basically deals with the case were the data obtained by the marker detection system is perfect. However, in practical circumstances, there are a number of complications:

- The stick figure is a very simple model of the human body. The human body's behavior is more complex and cannot be mapped exactly to the stick figure model. In particular, the behavior of the spine is more complex than can be modeled with a simple stick diagram.
- The skeleton, on which the stick figure is based, is not visible. Markers, placed on the skin, only approximate the location of joints. The accuracy particularly suffers from skin movements relative to the underlying bone during movement.
- Segment lengths can not be determined exactly.
- Marker detection is not perfect due to limitations of image resolution and image noise.

- Markers can be completely invisible due to other body parts that obscure the view: the occlusion problem.
- It is impossible to attach and detect six visible points on the head outside of a simulated environment.
- It is very difficult to acquire a first correct frame, necessary to apply 'smoothness of motion' information.

Chen and Lee only give results for simulation data, and from the above it is obvious that a practical implementation is anything but trivial. It is also quite clear from the above, that errors in the input for the algorithm *will* appear. Chen and Lee have not given an analysis of the robustness of their algorithm.

We believe that their method is very sensitive to errors in the input. This follows from the following example: the result of an error in segment length information.

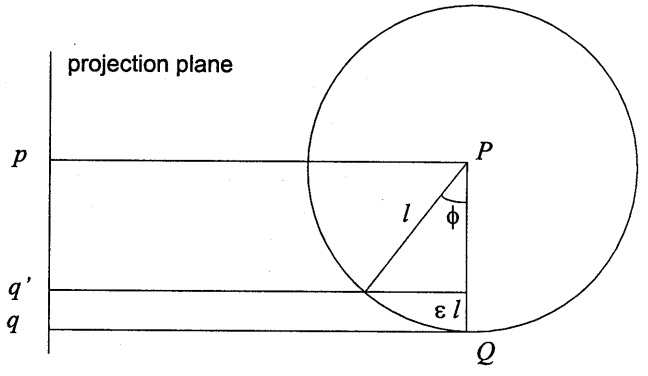


FIGURE 27
Error caused by incorrect segment length information

For simplicity, we assume a parallel projection and we consider the case where a segment is parallel to the viewing plane. Segments that are parallel to the viewing plane are quite common. For instance, in the sagital view, both arm and leg segments are (almost) parallel to the viewing plane all of the time during a healthy person's gait cycle.

Let ε be the relative error in length l, caused by marker misplacement and marker detection errors, and let ϕ be the resulting error in angle.

Then $\cos\phi = (1 - \varepsilon)$.

The most significant error in marker systems is the movement of the marker relative to the joints as a result of skin movement. These errors are in the range of 10-30 mm. Considering a body segment of 40 cm, errors can be in the range of 5%. Assuming an error of 5% ($\varepsilon = 0.05$), this leads to an error of 18.2 degrees in ϕ!

The case where the segment is parallel to the viewing plane is the worst case. If the segment is perpendicular to the viewing plane, an error in segment length does not cause an angle error.

The algorithm is also sensitive to other types of errors, such as errors in joint projections. Another problem is that of accumulation of errors: the reconstruction method causes errors to propagate through the reconstruction, since new points are constructed using previous ones that contain errors.

10.4 Reconstruction using a linearized set of equations

Chen and Lee focus on finding the correct configuration in their tree of configurations for each frame. However, in practical circumstances, it is not possible to acquire perfect input data. Input data will contain errors. As we have shown in the previous section, simply ignoring the input errors can lead to unacceptable errors in the output.

This essentially changes the discrete problem of eliminating illegal configurations from a tree into a continuous minimization problem of finding a solution with a minimal error, using a suitable cost function. We will now define the problem in terms of a continuous minimization problem. We are no longer looking for the 'correct' configuration from a discrete set of configurations. We want to find the configuration that has a minimal cost from an infinite set of configurations.

The following information is available for the cost function:

1. The set of 2D projections of 3D model points for each frame.
2. An estimate of the length of each body segment.
3. The fact that human motion is smooth, in other words there is limited movement between two adjacent frames.
4. The fact that the human body is limited in what it can do. Certain positions are physically impossible.

We drop the assumption that six model points on the rigid head are available, since this assumption is very unpractical. Furthermore, we will not use Chen and Lee's 'correctness rules', since these only seem to apply to healthy gait. We do assume that the camera parameters are known (see Section 9.6).

For computational reasons, it is preferable that the cost function can be given as an overdetermined set of linear equations that have to be solved, using the least-squares criterion. We will now first try to represent the information as a number of linear equations. Next, we will show how the optimization problem can be solved using singular value decomposition.

10.4.1 Projection information

Definitions:

N = the number of points in the model

M = the number of segments in the model

The relation between a point in 3D world coordinates and its projection in 2D image coordinates is given by and from Section 9.6.2. In the case of 3D reconstruction, the DLT parameters and the 2D projections are given, and the 3D originals are the unknowns.

We can rewrite the projection equations as:

$$X(L_1 - L_9x) + Y(L_2 - L_{10}x) + Z(L_3 - L_{11}x) = x - L_4 \qquad \textbf{(EQ 3)}$$

$$X(L_5 - L_9y) + Y(L_6 - L_{10}y) + Z(L_7 - L_{11}y) = y - L_8 \qquad \textbf{(EQ 4)}$$

where X,Y,Z are the unknowns. We now combine all $3N$ unknowns (three for each point) into the unknown vector $\boldsymbol{x}$.

We can then combine the $2N$ projection equations in one set of equations:

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad \textbf{(EQ 5)}$$

where A is a $2N \times 3N$ matrix, and $\boldsymbol{b}$ consists of the right-hand sides of the equations 3 and 4.

10.4.2 Adding segment length information

Segment length information can best be added by making a change in variables. Instead of describing the reconstruction problem in terms of 3 unknown cartesian coordinates for each marker point, we will now describe the problem in terms of a 3D point for one particular marker point, and 2 angles for the remaining marker points. The 3D point is a the starting point for the reconstruction, and will be called *the root.*

An alternative approach is to add the segment length information by adding additional equations for each length. However, this does not reduce the amount of unknowns and therefore we choose to make a change in variables.

Changing to these variables reduces the number of variables from $3N$ to $2(N-1)+3$. Changing variables also means that the projection equations **(EQ 5)** have to be rewritten in terms of the new variables. This will be the topic of the remainder of this section.

We start by showing how the cartesian coordinates can be reconstructed from the root and angles variables.

Let P and Q be 3D model points connected via a segment PQ of length l that has P and Q as end points. Then the 3D vector representing the segment PQ can be written as a function $f \in \boldsymbol{R}^2 \rightarrow \boldsymbol{R}^3$ with angles α, β as argument:

$$f_l(\alpha, \beta) = l[\cos\alpha\cos\beta, \sin\alpha\cos\beta, \sin\beta] \qquad \textbf{(DEF 1)}$$

where α and β are defined as in Figure 28.

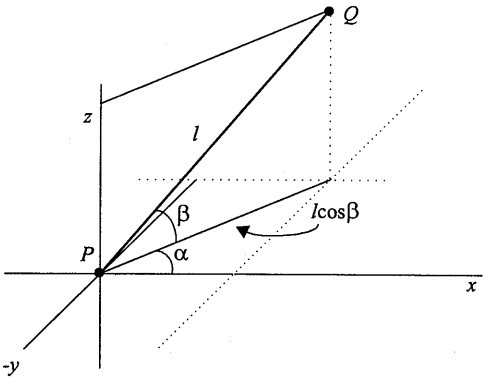


FIGURE 28
Definition of angles

The angles are α and β defined relative to the world coordinates.

The stick diagram that represents the human model is a graph, where vertices correspond to model points and edges correspond to segments:

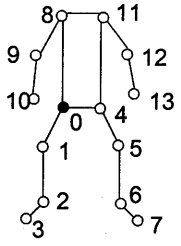


FIGURE 29
The stick diagram as a labeled graph

In Figure 29, we have labeled each vertex in the graph with a number, indicating the sequence number of the vertex in the reconstruction process. The root has number 0. We now define r as the reconstruction row: the row that starts with the root, followed by the other vertices in the order that they are to be reconstructed according to the labeling. For the i+1-th element, we will use the notation: r_i. Therefore, the root is r_0. Note that the labeling is chosen in such a way that every point in the reconstruction order row r can be reached from the root via a subset of its row predecessors. This means that for every point r_i, a path of predecessors of r_i can be constructed that starts at the root r_0 and ends at r_i. We choose such a

path for every point r_i. Let $w(r_i)$, be the chosen path of points that reconstructs r_i. We call $w(r_i)$ the *root path* of r_i,

We now define a vector $\boldsymbol{a}$ as: $[x, y, z, \alpha_1, \ldots, \alpha_{N-1}, \beta_1, \ldots, \beta_{N-1}]$, where (x,y,z) is the root r_0, and α_i and β_i are the angles that are defined for model point r_i. The angles α_i and β_i define the orientation of the segment that connects r_i to its predecessor along the root-path. The vector $\boldsymbol{a}$ consists of the set of position dependent parameters of our model (see Section 9.2).

From now on we assume that the order of the elements of $\boldsymbol{x}$ (introduced in **(EQ 5)**), is such that:

$$\boldsymbol{x} = [r_0.x, r_0.y, r_0.z, \ldots, r_{N-1}.x, r_{N-1}.y, r_{N-1}.z]$$

We will transform the projection equations for $\boldsymbol{x}$ into projection equations for the vectors $\Delta\boldsymbol{a}$, which consist of the differences of the root coordinates and the differences of all corresponding angles between two adjacent frames.

First, we define:

$$u_{r_i}(p) = \begin{cases} 1 \text{ if } p \in w(r_i) \\ 0 \text{ if } p \notin w(r_i) \end{cases} \qquad \textbf{(DEF 2)}$$

The point r_i can then be reconstructed by starting at the root r_0 and adding the segment vectors f for those segments that are part of the root path of r_i:

$$r_i = r_0 + \sum_{k=1}^{i} u_{r_i}(r_k) \cdot f_{l_k}(\alpha_k, \beta_k) \qquad \textbf{(EQ 6)}$$

Since f is a non-linear function, it has to be linearized using first order Taylor expansion yielding partial derivatives, before we can use it in our system of linear equations:

$$\frac{\partial f_{l_i}(\alpha_i, \beta_i)}{\partial \alpha_i} = l_i[-\sin\alpha_i \cos\beta_i, \cos\alpha_i \cos\beta_i, 0] \qquad \textbf{(EQ 7)}$$

$$\frac{\partial f_{l_i}(\alpha_i, \beta_i)}{\partial \beta_i} = l_i[-\cos\alpha_i \sin\beta_i, -\sin\alpha_i \sin\beta_i, \cos\beta_j] \quad \textbf{(EQ 8)}$$

It follows from **(EQ 6)** that:

$$\frac{\partial r_i}{\partial \alpha_k} = u_{r_i}(r_k) \cdot \frac{\partial f_{l_k}(\alpha_k, \beta_k)}{\partial \alpha_k} \quad \textbf{(EQ 9)}$$

Note that $\frac{\partial r_i}{\partial \alpha_k}$ is equal to $\frac{\partial f_{l_i}(\alpha_i, \beta_i)}{\partial \alpha_i}$

if r_k is an element of the rootpath of r_i and 0 otherwise.

A similar equation can be derived for $\frac{\partial r_i}{\partial \beta_k}$.

Recall that:

- $\boldsymbol{x}$ is the vector of model coordinates (three for each point), placed in the reconstruction order r defined by labeling of the model.
- $\boldsymbol{a}$ is the vector consisting of the root coordinates, followed by the α's and β's.

We define the Jacobian J as:

$$J_{ij} = \frac{\partial x_i}{\partial a_j} \quad \textbf{(DEF 3)}$$

We now go back to our initial set of equations $A\boldsymbol{x} = \boldsymbol{b}$, that represent projection information.

We define:

$\tilde{\boldsymbol{a}}$ = the previous frame value of $\boldsymbol{a}$,

$\tilde{\boldsymbol{b}}$ = the previous frame value of $\boldsymbol{b}$,

$\tilde{\boldsymbol{x}}$ = the previous frame value of $\boldsymbol{x}$,

$\Delta\boldsymbol{a} = \boldsymbol{a} - \tilde{\boldsymbol{a}}$, $\Delta\boldsymbol{b} = \boldsymbol{b} - \tilde{\boldsymbol{b}}$ and $\Delta\boldsymbol{x} = \boldsymbol{x} - \tilde{\boldsymbol{x}}$

We can now change from cartesian coordinates $\boldsymbol{x}$ to angle differences $\Delta \boldsymbol{a}$ as follows:

Linearizing g around $\tilde{\boldsymbol{a}}$, we have: $J(\tilde{\boldsymbol{a}}) \cdot \Delta \boldsymbol{a} \cong \Delta \boldsymbol{x}$:

We define $H = AJ(\tilde{\boldsymbol{a}})$. Then we get:

$$H \cdot \Delta \boldsymbol{a} = AJ(\tilde{\boldsymbol{a}}) \cdot \Delta \boldsymbol{a} \cong A \cdot \Delta \boldsymbol{x} = A\boldsymbol{x} - A\tilde{\boldsymbol{x}} = \boldsymbol{b} - \tilde{\boldsymbol{b}} = \Delta \boldsymbol{b} \qquad \textbf{(EQ 10)}$$

Hence,

$$H \cdot \Delta \boldsymbol{a} = \Delta \boldsymbol{b} \qquad \textbf{(EQ 11)}$$

is the set of equations to be solved, and $\Delta \boldsymbol{a}$ is the unknown vector. The difference between this set of equations and the original set $A\boldsymbol{x} = \boldsymbol{b}$, is that the new set of equations contains segment length information and consists of fewer unknown variables than the original set of equations.

10.4.3 Smoothness of motion information

Smoothness of motion can be enforced by adding equations that require each variable to be close to a predicted value. The predicted value can be computed by using information from the previous frames. We choose a simple prediction scheme based on the two previous frames: the predicted value is equal to the difference between those two frames $\Delta \hat{\boldsymbol{a}}$. This leads to a set of equations:

$$I \cdot \Delta \boldsymbol{a} = \Delta \hat{\boldsymbol{a}} \qquad \textbf{(EQ 12)}$$

The equations from **(EQ 11)** and **(EQ 12)** can be combined into a single set of equations, by extending the matrix H and the vector $\boldsymbol{b}$ into $H', \boldsymbol{b}'$:

$$H' \cdot \Delta \boldsymbol{a} = \begin{bmatrix} H \\ \lambda I \end{bmatrix} \cdot \Delta \boldsymbol{a} = \begin{bmatrix} \boldsymbol{b} \\ \lambda \Delta \hat{\boldsymbol{a}} \end{bmatrix} = \boldsymbol{b}' \qquad \textbf{(EQ 13)}$$

The weight factor λ is a measure of the relative importance of the smoothness information compared to the projection and length information. By giving λ a higher value, the bottom half of H'

becomes more important when a least squares solution is computed. This will be explained in Section 10.5.

In the following sections, we will refer to H' as H and to $\boldsymbol{b}'$ as $\boldsymbol{b}$.

10.4.4 Physical constraint information

Physical constraints are constraints that describe limitations of the human body. In particular, angles between segments that can be made in joints are limited to certain ranges. Unfortunately, it is not easy to add the physical constraint information to our set of equations, since it requires inequalities instead of equations. Furthermore, the information that physical constraints give us, does not add much to the cost function. It eliminates 'regions' of the solution space, but usually is does not help us to determine which of two similar solutions is better.

Design Decision: Therefore, we do not use this information in our set of equations. The physical constraint information could be used to check the results of the reconstruction process. However, this approach was not followed in our research.

10.5 Solving the equations using singular value decomposition

10.5.1 Definition of SVD

Singular Value Decomposition (SVD) is a powerful mathematical tool to find a numerical solution of a set of equations [78].

Any matrix H of dimension $M \times N$, $M \geq N$, can be decomposed as follows:

$$H = UDV^T$$

where the matrices U,D, and V have the following properties:

- V is a square matrix of size $N \times N$
- U has size $M \times N$
- U and V have orthonormal columns

- D is a diagonal matrix. Its diagonal entries are non-negative and can be chosen to be increasing, independent of the particular choice of U and V, and are called the *singular values* of H. The singular values are denoted by ω_i:

$$D = \begin{bmatrix} \omega_1 & 0 & \dots & 0 \\ 0 & \omega_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \omega_N \end{bmatrix}$$

If H is singular, there are not enough singular values unequal to zero to fill D. In that case, SVD finds ω's equal to zero for the remaining diagonal elements.

10.5.2 Solving equations with SVD

The SVD of H can be used to analyze a set of equations $H\boldsymbol{x} = \boldsymbol{b}$.

If H has full column rank (i.e. if none of the ω's is equal to zero), a 'least square solution' for the set of equations is given by:

$$\boldsymbol{x} = V \cdot \text{diag}\frac{1}{\omega_j} \cdot (U^T \cdot \boldsymbol{b}) \qquad \textbf{(EQ 14)}$$

A 'least square solution' is a vector $\boldsymbol{x}$, that minimizes the value:

$$\sum_{i=1}^{M} ((H\boldsymbol{x})_i - \boldsymbol{b}_i)^2 \qquad \textbf{(EQ 15)}$$

If the set of equations $H\boldsymbol{x} = \boldsymbol{b}$ has a unique solution, it is also the 'least square solution', so **(EQ 14)** gives an exact solution if possible.

If there are singular values that are equal to zero, a least square solution can be found by using **(EQ 14)**, with $1/\omega_j$ replaced by 0 for every $\omega_j = 0$.

If the least square (or exact) solution is not unique then **(EQ 14)** gives the solution with the smallest norm.

SVD can also be used to evaluate the robustness of a problem. If there is a large difference between the smaller and larger singular values, the system is not robust. This can be determined via the *condition number,* which is defined as the ratio of the highest and lowest singular value of a matrix. The number of small singular values gives the dimension of the subspace of the solution space that is sensitive to errors in the input.

10.5.3 Application of SVD

In the previous sections, we have determined a set of equations that represent the information necessary to reconstruct a 3D model from its 2D projection. This resulted in a set of equations $H \cdot \Delta \boldsymbol{a} = \Delta \boldsymbol{b}$. We perform the SVD on the matrix H. Using the method described in Section 10.5.2, a (least square) solution to $H \cdot \Delta \boldsymbol{a} = \Delta \boldsymbol{b}$ can be found. From the analysis of the singular values (the matrix D), further conclusions about the robustness of the set of equations can be drawn: small condition numbers indicate that the set of equations is robust, while large condition numbers indicate that the set of equations is sensitive to errors.

10.5.4 Scaling and weights

Each of the equations of the set $H \cdot \Delta \boldsymbol{a} = \Delta \boldsymbol{b}$ represents information. If the set $H \cdot \Delta \boldsymbol{a} = \Delta \boldsymbol{b}$ does not have an exact solution, a solution can be given using the least squares criterion. The least squares criterion can be 'manipulated' in two ways: by *scaling* and by giving *weights*.

As was described in Section 10.4.3, the relative importance of the smoothness information can be set with a weight factor λ. When computing a solution using the least squares criterion, it is clear that equations which are given a higher weight, contribute more to the squares sum of **(EQ 15)**, and therefore these equations gain in importance.

Each component of the unknown vector $\boldsymbol{a}$ can be scaled before the SVD-operation and the scaled back after this operation. This way, certain components gain significance. Recall that the first 3 elements of $\boldsymbol{a}$ represent root coordinates, the others represent angles. Using scaling, the relative importance of errors in the root coordinate components with respect to errors in the angle components can

be given. For this we introduce μ, the relative importance of the root coordinates.

10.6 Results of our approach

We have tested our algorithm on simulation data provided by the WALT-system [41]. The WALT system is an animation system developed by the computer graphics group of the Eindhoven University of Technology.

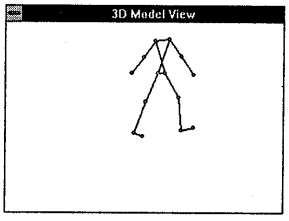


FIGURE 30
Example of a stick figure generated by the WALT system

The test data consists of an animation of a walking person. The WALT system delivers both 3D coordinates and 2D projections. This enables us to test the reconstruction algorithm: the 2D projection are the input data, the output can be checked with the 3D originals. The chosen frame rate is equal to that given in the specification (see Section 4.6). The frame rate is important for the reconstruction since higher frame rates imply smoother motion.

The input provided to the reconstruction algorithm consists of:

- the previous 2 frames in 3D
- the correct segment lengths
- the projection values for the current frame

Like Chen and Lee, we use simulation data to test our algorithm. However, we use not only perfect data, but also data that contains realistic errors. In this section, a number of examples are given. These examples are based on reconstruction of a single frame using the previous frames. This means that propagation errors are not taken into consideration. A single camera view is used, unless otherwise stated. We give average and maximal errors in the angles (in

degrees). Furthermore, the condition number is given, to indicate sensitivity to errors.

We used two image resolutions to test our algorithm: 'infinite' and 751×576. Infinite resolution indicates that all calculations were performed with floating point calculations. In case of a resolution of 751×576, all projected image points are rounded to grid points on a grid of 751×576 pixels. In practical circumstances, an image resolution of 751×576 is realistic. This limitation of the resolution is a potential cause of errors.

10.6.1 Projection information

In an initial test, we only used projection information and left out smoothness information. Analysis of the singular values showed that one singular value was very close to zero, and therefore the condition number of matrix H is very large. This corresponds to the fact that there is one more variable than there are equations: there are $2 \cdot (N-1)+3$ variables and $2 \cdot N$ equations. As was mentioned in Section 10.5.2, in this case SVD gives the solution vector with the smallest norm.

Since in our case, the unknown variable consists of root coordinates and angle *differences*, this means that the solution that is chosen is the one which corresponds to the smallest amount of motion.

Smoothness	**Resolution**	**Length error**	**View**
none	infinite	none	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
2.1	13.7	$6.1*10^{20}$	

Example 16 no smoothness

In our first example, we have assumed an 'infinite' image resolution. In real marker detection systems, there is always a limited number of scan lines and pixels in the images. As a first error-introduction, we add a limited resolution of 751×576. Apparently, the

influence of using a limited resolution of 751 × 576 is relatively small. The average error even dropped in some cases.

Smoothness	Resolution	Length error	View
none	751 × 576	none	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
1.8	23.9	$1.0*10^{20}$	

Example 17 limited image resolution

Next we added random length errors of up to 5%, picked using a homogenous distribution function. The results show large angle errors. The largest error was found in joints with large displacements (elbow and knee).

Smoothness	Resolution	Length error	View
none	751 × 576	5%	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
10.4	137.0	$8.0*10^{20}$	

Example 18 segment length errors

10.6.2 Adding smoothness information

Next we added smoothness information. Smoothness information uses a predicted value, normally based on previous frame values. In our simulation, the 3D data of the frame to be reconstructed was also available. From this data, we computed the 'real' solution and used this to test the effects of adding smoothness information. We call this *perfect prediction* (it is also known as *cheating*). In this

test, we gave smoothness information a relatively large weight compared to the weight of the projection information.

Smoothness	Resolution	Length error	View
perfect prediction	751 × 576	none	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
0.48	2.2	$2.5*10^2$	

Example 19 smoothness with perfect prediction

The singular value decomposition showed no singular values close to zero. This is obvious, since the number of (independent) equations is now much larger than the number of variables. As a result, the condition number is much lower. Because of the perfect prediction, the errors are small. The remaining errors result from limited image resolution and the errors in the linearization of the projection equations.

Next, we tested the algorithm by using information from the two previous frames. The weight of the smoothness information had to be reduced because the smoothness information was no longer perfect. As a result the condition number was larger, but still much lower than in the case were no smoothness was used. The results show only a slight improvement over the case were no smoothness information was used (example 17).

Smoothness	Resolution	Length error	View
normal prediction	751 × 576	none	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
1.4	13.9	$2.3*10^5$	

Example 20 smoothness with normal prediction

Next, we also add segment errors. Again, the results are only slightly better than the case where no smoothness information was added (example 18).

Smoothness	Resolution	Length error	View
normal prediction	751 × 576	5%	sagital
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
9.3	116.3	$1.8*10^5$	

Example 21 smoothness with normal prediction and length errors

10.6.3 Other views

We also tested the effect of changing the camera position, using a front view and a top view.

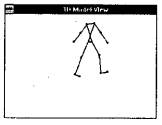
Smoothness	Resolution	Length error	View
normal prediction	751 × 576	5%	front
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
7.9	72.8	$1.3*10^5$	

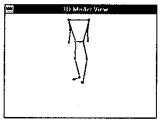
Example 22 front view

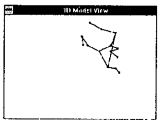
Smoothness	Resolution	Length error	View
normal prediction	751 × 576	5%	top
Average Angle error (degrees)	**Maximal Angle error (degrees)**	**Condition number**	
1.9	13.2	$3.4*10^5$	

Example 23 top view

The best results were found when the camera took a top or bottom view. The reason for this is that in these cases there are only two segments, the shoulder and hip segments, that perform their motion more or less parallel to the viewing plane (see Section 10.3).


Sagital view


Front view


Top view

10.6.4 Iterative approach

One source of errors in our calculations is caused by the linearization of the non-linear equations, as performed in **(EQ 10)**. Linearization is necessary to transform the projection equations based on cartesian coordinates (x) into equations based on angles (a). Linearization takes place around $\tilde{a}$, the previous frame value of a.

An iterative approach was followed in order to minimize these linearization errors. In general, linearization errors are small when the solution for Δa in **(EQ 11)** has a small value. Therefore, we try to minimize Δa, by changing the point to linearize around. This is similar to the Newton-Raphson root finding algorithm. A new point to linearize around is found by adding the solution Δa. This leads to the following algorithm:

$a\_linearize = \tilde{a}$;

$number\_of\_iterations = 0$;

do

begin

compute H, Δb using ***a_linearize;***

solve $H \cdot \Delta a = \Delta b$;

$a\_linearize = a\_linearize + \Delta a$;

$number\_of\_iterations = number\_of\_iterations + 1$;

end while ($\Delta a > threshold$ **and**

$number\_of\_iterations < max\_iterations$)

$a = a\_linearize$;

In the algorithm, linearization occurs around ***a_linearize***, which at the start is equal to the value of the previous frame $\tilde{a}$ and then is changed each iteration using the result Δa of **(EQ 11)**. Note that H and Δb have to be recomputed each time, since they depend on the point that is linearized around. This procedure is continued until Δa is lower than a given threshold or the number of iterations exceeds a given number.

The results from the iterative approach show that when the SVD provided good results in the first place, the results after iteration were even better. In fact, when no errors were introduced and perfect prediction was used, the solution converged towards the real solution.

Smoothness	precision	Length error	View
perfect prediction	float	none	sagital
Average Angle error	**Maximal Angle error**	**Condition number**	
0.0	0.0	$2.4*10^5$	

Example 24 iterative approach with infinite resolution (10 iterations)

However, when errors were introduced, the iteration process did not lead to a better solution. Unfortunately, even adding only imperfect prediction and a limited resolution of 751×576 was enough to disturb the convergence process.

Smoothness	precision	Length error	View
normal prediction	integer	none	sagital
Average Angle error	**Maximal Angle error**	**Condition number**	
1.7	23.6	$1.6*10^5$	

Example 25 iterative approach with a limited resolution of 751×576 (10 iterations)

10.7 Conclusion

In this chapter, the possibility of retrieving kinematics from a single view was investigated. Chen and Lee's algorithm was taken as a starting point. It was concluded that their algorithm is too sensitive to errors to be used for Sybar. An alternative approach was tried by setting up a set of equations, to be solved by singular value decomposition. The algorithm was tested by adding length errors in the range of 5%, which is a realistic value. Unfortunately, the results show that the system of equations is still too sensitive to be useful in the sagital view.

Better results were found using the top view. However, it is not practical to perform the low level marker detection from the top view, since markers are occluded almost all the time. Furthermore, in Sybar, it is the sagital view that physicians want to see and that view therefore is available.

Amaya, Hara and Aoki have published an interesting article after the Sybar project was finished [39]. They follow an approach that is very similar to ours, although there are also some differences. In order to solve the 3D reconstruction problem, they make a few additional assumptions:

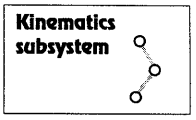
- They assume that the foot does not slip.
- They assume that the body center is located above the stance.

These assumptions may be violated in the gait of patients with motion disorders. Furthermore, results were published for 'Sumo motion', and not for walking gait. We think that this algorithm does not solve the problems we encountered with our own algorithm.

Concluding, our research shows that 3D reconstruction using only one camera leads to a system that is either very sensitive to errors, or uses an unpractical camera view point. We will therefore drop the single camera restriction and investigate how 3D analysis can best be performed using more than one camera. This is the topic of the next chapter.

CHAPTER 11

Kinematics Subsystem



This chapter describes the kinematics subsystem. It uses the results of the research of the previous chapters. First, the system is divided into three subsystems. Next, a description of each the subsystems is given.

11.1 Introduction

In chapter 9, an overview of the currently used kinematics detection methods was given. A marker approach turned out to be the best approach for Sybar. Furthermore, it was concluded that an algorithm that uses a single camera to obtain the 3D kinematics is to be preferred. However, in chapter 10, it was concluded that such an approach is not feasible for Sybar.

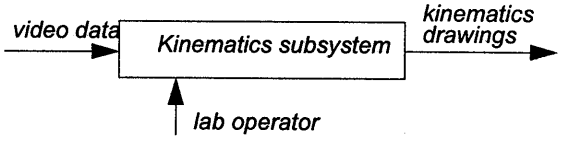
The next best option is an approach that uses more than one camera to obtain 3D kinematics. This chapter describes the design of the kinematics subsystem. It is based on the multi-camera approach, as described in section 9.9.3

11.2 Kinematics subsystem design

The kinematics subsystem acquires a kinematic description of human movement through the analysis of video images. The video images are provided by the recording subsystem. The kinematics

subsystem is able to draw human models as a service to objects of the display subsystem. The kinematics subsystem needs a human operator to perform some tasks. Figure 31 shows the system input and output.

FIGURE 31
subsystem input and output



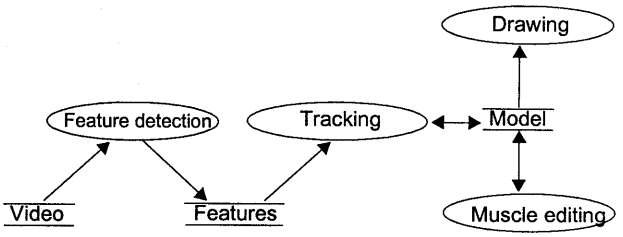
The relevant requirement definition for this subsystem is: *The lab operators should find the system easy to use and should be able to perform data acquisition and data processing quickly* (requirement definition 3, Section 3.3). The video data consists of a set of images. The kinematics output is given in the form of drawings of the position of the stick figure at a certain moment in time.

Design Decision: As mentioned in the specification in Section 4.3.1, the accuracy of the system is not our main concern. To keep things simple, we attach two markers to each segment, and assume that they correspond to joints. A more accurate (possibly future) approach is to use larger number of markers for each segment. The result of the current setup is that the third degree of freedom of the orientation of a segment can not be determined (see Section 9.9.3).

Design Decision: The information available to the kinematics system consists of a large amount of data. For reasons of efficiency, a two stage approach is followed, as described in Figure 23 of chapter 9. First, low-level marker detection is performed for each image. Next, the correspondence and occlusion problems are solved and the resulting 3D kinematics data is stored in a model. A disadvantage of this approach is that the marker detection does not take advantage of model information.

The kinematics system is functional of nature: it basically carries out an algorithm that takes input and produces output. We therefore first give a functional model of the system.

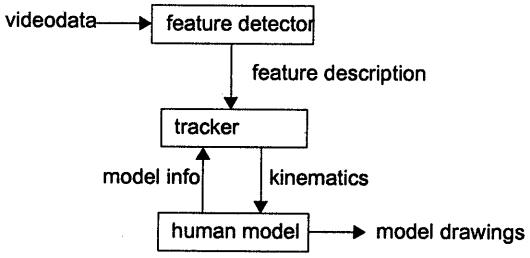
FIGURE 32
Functional Model



The following processes can be distinguished:

1. *Feature detection*: the first step is the low-level detection of markers. Markers can be seen as features of the image. These image features are detected with an algorithm that will described later on. The features are not only detected, but also given a quality, which can be used during the tracking. The quality of a features indicates the likelihood that a feature is indeed the projection of a real marker. Finally, the size of the features is also determined.
2. *Tracking*: the detected features have to be tracked and matched with the correct model points. In case of a 3D kinematic analysis, 3D marker positions have to be calculated from features of different camera views. The 3D reconstruction is considered part of the tracking (although it is strictly speaking a separate process). The result of the tracking process is a description of the position dependent parameters of the model.
3. *Drawing*: the model can be drawn in various ways. Drawing is performed on the request of the display subsystem.
4. *Muscle editing*: the model consists not only of a kinematic description of joints and body segments, but also of muscles. The locations of muscles can not be retrieved from video images directly. Therefore, their position relative to the segment of the matched stick figure has to be set by the lab operator.

We subdivide the kinematics system into subsystems:



The *feature detector* takes care of the feature detection process. The output is a feature description of the video (a list of features for each frame). The *tracker* matches the features to model points. For this, it uses the *human model*. The kinematics are stored in the human model. The human model provides services to draw itself. These services are used by the display subsystem.

The remainder of this chapter describes the three subsystems.

11.3 The feature detector

11.3.1 Marker detection algorithm

The feature detection subsystem detects markers in images. A general method for detecting objects in images is known as *template matching* (see Section 9.9.1). Searching the image space for the best template match is computationally expensive, especially if the shape of the template is not exactly known. In our case, the size of the marker is an additional variable that needs to be taken into consideration. Sometimes, search windows based on previous frame object locations are used to restrict the area of the template search. However, this approach cannot be used in the presence of occlusion.

Most marker detection systems start by first thresholding the image and then search for markers in the resulting binary (black and white) image. If the lighting conditions are good and a high threshold can be set, for example in infrared systems, it may be assumed

that only markers show up in the thresholded image. Markers can then be found by searching the binary image for regions of white points, and marker centroids can be found by taking the average of the coordinates of all pixels in a marker. The marker position can also be used as a starting position for more accurate template matching.

We found that under the conditions that could be set at the motion analysis lab at the VU Hospital, there is rather a lot of noise that shows up in the thresholded image. One of the reasons that the conditions are not optimal is that the images are also to be used for observational analysis. There is a trade-off between conditions that are optimal for marker detection and conditions that are optimal for human observation.

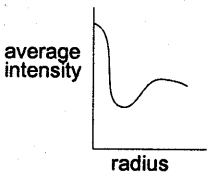
Design Decision: To perform a template match with a variable size on the entire image is computationally too expensive. We therefore first search the thresholded image for connected white pixels, which we call *candidate markers*. Candidate markers are found by using a standard fill algorithm starting at white pixels in the thresholded image. The next step is to determine for each candidate region whether or not it represents a marker. This leads to the following algorithm:

```
threshold the image;
for every pixel p in the thresholded image
if "p is white" and "p is not yet used" then
begin
    candidate_marker= fill_pixel_area (p);
    determine_feature_quality(candidate_marker);
end
```

Design Decision: We have chosen to use white markers shaped as spheres on top of a black base. The base allows the marker to be attached to the skin, and ensures that the marker's surrounding has a high contrast with the marker itself. Because the marker is shaped as a sphere, it has the same appearance, independent of the view point.

To determine the feature quality, we first determine the center of the candidate marker by taking the average of the coordinates of all pixels. Then we go back to the original image and examine the average intensity on circles with a variable radius, with respect to the center of the marker candidate. The graph that shows average intensity against radius typically has the following form:

FIGURE 33
average intensity against radius



Given a candidate marker position, the radius and a quality can be defined by considering the intensity/radius graph. As an estimate for the marker radius in the image, the radius with minimum intensity can be taken. As a measure for the quality, we take the difference between the minimum and the maximum intensity. Using these values, the marker recognition can be performed relatively efficiently, while still using the grey value information of the original image.

The accuracy of the marker detection depends on the lighting conditions, and the distance between the camera and the marker. Under ideal lighting conditions, pixel averaging results in sub-pixel accuracy. However, in our case, an accuracy of one pixel is more realistic. Assuming a sagital view, a resolution of 751×576, and a marker with a diameter of 35 mm, the image diameter of a marker is about 10 pixels. Assuming an accuracy with an error not larger than a single pixel, this results in an accuracy of about 3.5 mm. The inaccuracy that is the result of skin movement is in the range of 10-30 mm [18]. Therefore, the accuracy obtained for the 2D marker detection is reasonable.

The algorithm's accuracy may be improved, at the cost of efficiency, by trying other points than the average marker pixel coordinates as marker center candidates. The algorithm's robustness may be improved by using a more sophisticated quality function.

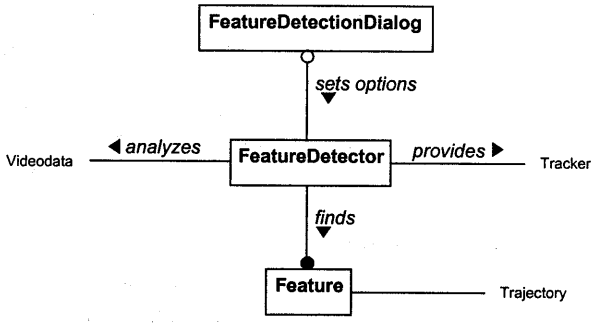
Design Decision: Since the algorithm's performance was reasonably satisfactory, it was decided not too improve it further (considering the limited available resources).

We now describe the classes involved in the marker detection process.

11.3.2 System design

Figure 34 shows the object design for the feature detector subsystem.

FIGURE 34
Feature Detector subsystem



The purpose of the feature detector subsystem is to determine the marker positions for each of the video images. The most important class for this subsystem is the *FeatureDetector*. The *FeatureDetector* analyzes images, provided by the *VideoData* from the recording subsystem. Images are analyzed, using the algorithm described in the Section 11.3.1. This result is a number of *Features*.

The feature detection algorithm requires a number of parameters, to be set by the user. The parameters can be set via the *FeatureDetectionDialog*. We will now describe the individual classes of the feature detector subsystem.

FeatureDetector
parameters image
detect_features set_parameters threshold detect_features fill_pixel_area determine_feature_ quality

11.3.3 FeatureDetector

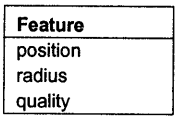
The *FeatureDetector* manages the feature detection process. It has a number of operations that are used to implement the feature detection algorithm.

The *detect_features* operation performs the feature detection, using the algorithm described in Section 11.3.1.

- For each image, a copy is made, by creating an instance of the *image* class. The *image* class is a library class part of the toolkit.
- A *threshold* operation is performed on each image. The threshold operation performs a threshold with a given threshold level, which results in a black and white version of the image.
- The *fill_pixel_area* performs a standard fill operation and results in a candidate marker. The *determine_feature_quality* gives a quality for the similarity of the candidate with an ideal marker. Furthermore, an estimate for the marker radius is given

The results are used by the *Tracker*, from the tracker subsystem. The *set_parameters* operation starts a *FDParametersDialog* allowing the user to set the parameters for the feature detection.

11.3.4 Feature

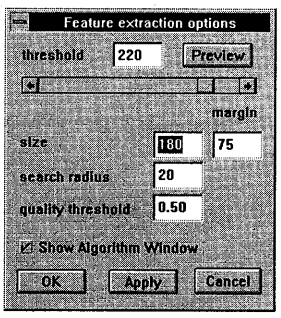


The *Feature* class is a simple class that represents a detected or estimated marker. A feature has a *position* and a *radius*, corresponding to the center and radius of the marker. Furthermore, a *Feature* has a *quality*. A low quality indicates that the feature may be a false feature, meaning that it does not correspond to a real marker. Because markers can be temporarily invisible, feature positions are also sometimes estimated.

11.3.5 FeatureDetectionDialog

The *FeatureDetectionDialog* is a dialog that allows the user to set the parameters for the marker detection:

FeatureDetection Dialog
preview
ok_button
apply_button
cancel_button



The following parameters can be set:

1. *The threshold level*, which can be set with the scroll bar. The *preview* button creates a window with the thresholded image.
2. *The size*: for reasons of efficiency, it is possible to discard features that are not within a given range of size. The size is measured as the number of pixels in an area. The margin indicates the range that the size of the features should be in. The margin is given as a percentage of the size. The margin is usually chosen rather large (about 75%), to make sure that no real features are thrown away.
3. *The search radius*: the maximum radius that is used for the intensity/radius graph of Figure 33.
4. *Quality threshold*: features with a lower quality are discarded. Features with a higher quality are the input for the tracking process.

If the 'Show Algorithm Window' option is chosen, a window that shows an image with the results of the marker detection is generated during the marker detection. The difference between *OK* and *Apply* is that the dialog remains open when *Apply* is pressed, whereas the dialog is closed when *OK* is pressed.

Design Decision: The selection of parameters is a design decision. The trade-off is between ease of use and flexibility: the more parameters, the more flexibility there is to tweak the parameters, and the more difficult it becomes to use the system.

11.4 The human model

11.4.1 System design

Design Decision: The model subsystem consists of two parts: The basic model and the muscle editor. The latter provides the user interface for editing the muscles. The model does not often need to be edited. Therefore, model editing is done via file. Muscle editing needs to be performed by the lab operator for each measurement. Therefore, a graphical user interface is provided.

Figure 35 shows the object design of the model subsystem. The elements from this subsystem are described in the remainder of this chapter.

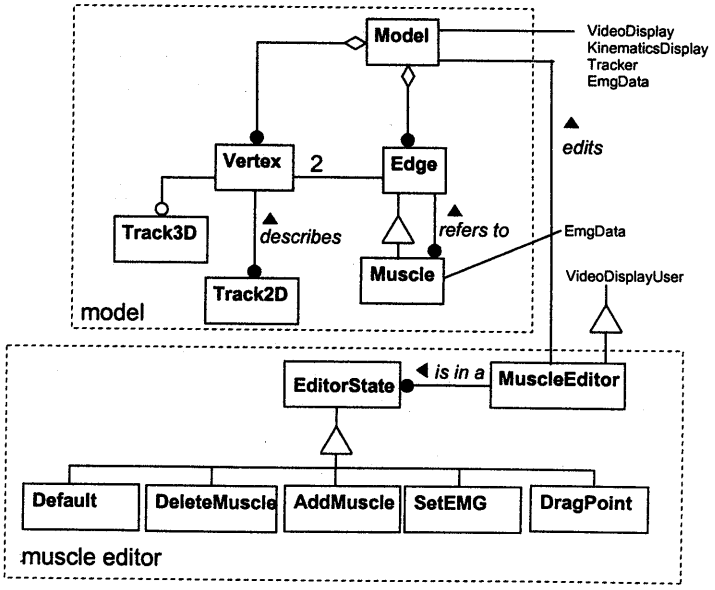


FIGURE 35
Object design human model

Basic model elements

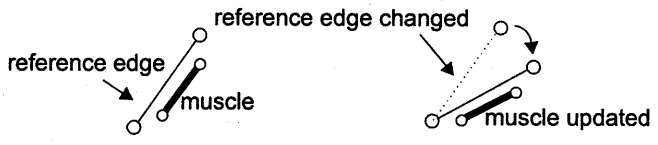
As has been discussed in the previous chapters, kinematic descriptions of human motion are always based on a model. Our model is the stick figure model (see section 9.2). The stick figure model consists of a number of vertices and edges. Vertices correspond to joints and edges to rigid body segments.

A *Model* consists of a number of *Vertices* and *Edges*. *Vertices* are descriptions of joint positions in space though time. *Edges* are objects that have references to two vertices.

During a measurement, joints move through space. In our model, the *Vertices* can describe a path through space. Since both 2D models and 3D models are supported, there are 2D trajectories and 3D trajectories. A *Track2D* consists of an ordered list of *Point2Ds*, a *Track3D* corresponds to an ordered list of *Point3Ds*. *Point2D* and *Point3D* are basic classes (not shown in the object diagram) corresponding to pairs and triples of coordinates respectively. A *Vertex* can be associated with several *Track2Ds*, one for each video channel. If 3D reconstruction has taken place, a *Vertex* is associated with a *Track3D*.

Muscles

One of the purposes of the kinematics subsystem, is to determine the approximate location of muscles in video images, in order to be able to display EMG values. As was mentioned in the specification, EMGs are shown as gauges. Therefore, a muscle can be modeled as an edge with two end points. The location of this edge can be determined from a *reference edge* of the stick figure model:



To incorporate the muscles in our object model, we introduce the *Muscle* class, which inherits from *Edge*. The muscle has an additional association with its reference edge. The end points of *Muscles* are *Vertices*: they can describe both 2D and 3D trajectories.

We will now describe the most interesting classes of the model subsystem: The *Model*, *MuscleEditor* and *EditorState*.

11.4.2 Model

Model
add_vertex
add_segment
add_muscle
delete_muscle
edit_muscle
draw
empty
save
load

The *Model* object is the container for all the model elements. Model elements can be created with the operations *add_vertex*, *add_segment*, and *add_muscle*. Muscles can be deleted with the *delete_muscle* operation.

The *edit_muscle* operation changes the relative position of the muscle with respect to its reference edge. This operation is used by the *MuscleEditor*.

The *Model* can *draw* itself in one of two ways:

1. The 2D model can be drawn directly. Each of the available 2D tracks is shown.
2. The 3D model can be drawn from a given camera position.

The first option is used by the *VideoDisplay* to draw muscle annotation on top of the video image.The second option is used by the *KinematicsDisplay*, which lets the user determine the point of view (see Section 8.2.12).

The *Model* also has the ability to *empty* and *save* itself. Finally, a model can be loaded from disk with the *load* operation.

11.4.3 MuscleEditor

Design Decision:

MuscleEditor
add_muscle
drag_muscle_point
delete_muscle
set_EMG
mouse_click

The *MuscleEditor* allows the user to add, delete, and edit muscles. The *MuscleEditor* is in a certain state, which depends on the user action that is expected. We choose a state machine solution to model the editor: we define a separate class for each state. The states all inherit from *EditorState*. This means that external events can all be delegated to the current state (using polymorphism). The alternative is to check the state for each event. This solution needs less classes, but requires a large number of switch statements, and results in a less clear design.

The *MuscleEditor* responds to user events by sending them to the current state. After dealing with the event, the current state returns the new state of the *MuscleEditor*. In the default state, a mouse click starts the dragging of a muscle point.

Muscles are defined with respect to a reference edge. When the user indicates that a new muscle is required (via the menu), the *add_muscle* operation is called. The user is requested to select an edge in the *VideoDisplay*.

Design Decision: The muscle is then created at a default location with respect to this edge. Creating the muscle at a default location makes defining a new muscle easy for the user.

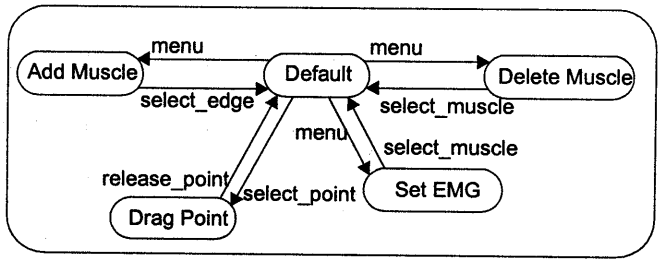
Design Decision: The location can be altered by the user by dragging muscle points, for which the *drag_muscle_point* operation is available. The muscle points are then redefined and automatically altered for each of the frames. This means that the user can alter the position in a single frame, and does not have to change them in each frame. In case of a 3D view, the user can set the muscle positions for both camera views independently of each other. A best matching 3D muscle point is then calculated from the two views, using the same algorithm that determines the 3D position of a marker point from two projections.

The two design decisions mentioned above are easily mistaken for requirements. However, because only the lab operator will use this part of the user interface, these user-interface aspects are considered internal to the system (see system boundaries, Section 4.2).

Muscles can be deleted by the user in a similar way: after the user has indicated that he wants to delete a muscle, the user is requested to select a muscle in the *VideoDisplay*.

The EMG channel that corresponds to a muscle can be set with the *set_EMG* operation. For this purpose, the *MuscleEditor* is associated with an *EMGdata* object. The *MuscleEditor* uses the *VideoDisplay* as a background and therefore is a *VideoDisplayUser*, see Section 8.2.10.

The *MuscleEditor* has the following states:



EditorState
cursor
button_up button_down set_cursor

11.4.4 EditorState

The *EditorState* is a base (parent) for all the classes that correspond to states of the *MuscleEditor*. It has operations to react to *button_up* and *button_down* events. These operations are *not* abstract. In the *EditorState*, they provide the behavior of doing nothing. This means that if a state does not want to react to a mouse button event, it simply does not overrule it. The *EditorState* also has a function *set_cursor* that sets the cursor to the *cursor* attribute. Each state has its own cursor which provides feedback to the user. The children of *EditorState* are not described here, since they have a simple structure.

11.5 Tracker

The tracker subsystem takes care of determining the position dependent model parameters for each image, using the features as detected by the feature detector. We first consider the 2D, single camera, problem of determining the features corresponding to a model point and then extend this to the 3D, multiple camera, reconstruction.

11.5.1 2D correspondence via smoothness of motion

In the 2D case, the problem can be divided into two sub problems:

1. Determine the trajectories of the features in the image sequence
2. Determine which trajectories correspond to which model points

Trajectories can be found by assuming that the motion of the features is smooth. Finding trajectories of feature points can be formulated as an optimization problem: find the set of trajectories that has the highest global smoothness. Given N feature points and F frames, this leads to $(N!)^F$ possible sets of trajectories. The problem is too complex to compute an exact solution for. However, there are a number of heuristic tracking algorithms that efficiently compute an approximation of the solution: see for instance [43, 50,51].

An additional problem is that in general, due to imperfect feature detection, there are both temporarily missing features and 'false features', i.e. points incorrectly identified as features.

One approach that has been used to reconstruct 3D objects is described in [45, 46, 55]. In this approach, large sets of markers are attached to the surface of a deformable object, such as the heart. The 3D trajectories are reconstructed with high accuracy using an 'SVD model'. Unfortunately, this approach can not easily be extended to the entire body.

We found Hwang's tracking algorithm [51], which deals with imperfect feature detection, most suitable for our problem.

Design Decision: We therefore choose Hwang's tracking algorithm as a basis for our tracking algorithm.

11.5.2 Hwang's tracking algorithm

Hwang's tracking algorithm analyzes sets of feature points from an image sequence. The algorithm generates trajectories using a frame by frame approach. The following approach is followed (*n_frames* is the number of frames):

"generate initial trajectories";

for *i*=0 to *n_frames*-2

begin

"generate plausible extended trajectories to frame *i*+1";

"select best n trajectories for each feature in frame *i*+1";

end

"determine best trajectory for each feature point of the final frame";

Plausible trajectories are found by extending the existing trajectories. For each trajectory, a prediction is made where the next trajectory point will be. Feature points in the frame under investigation that are within a given distance from the predicted value lead to new extended plausible trajectories. This means that a feature point can be part of several plausible trajectories. Furthermore, a single trajectory can lead to several extended trajectories, if more than a single feature point is within close range of the prediction. The predicted point is calculated by extending the trajectory using the current speed and acceleration.

To reduce the number of trajectories during the algorithm, only the best trajectories for each of the feature points in the current frame are stored. The quality of the trajectories is determined by examining the smoothness. Furthermore, each feature point gives a vote to the trajectory with the best quality containing that feature point. The number of votes for each trajectory is stored. Trajectories with a large number of votes are selected over those with few votes.

The algorithm deals with occlusion in the following way: if a trajectory can not be extended via a feature point, a new 'phantom' feature point is created. The coordinates of that feature point are those of the predicted value. In the next frame, the trajectory can find a real feature point, or create another one via prediction. The number of times in a row that a real feature point could not be found is called the *age* of the trajectory. Trajectories are not allowed to become too old. This prevents that too many trajectories consisting of phantom points are kept.

Let T be a trajectory, consisting of points $(P_1, P_2,...,P_k)$. Let v_i be the velocity of P_i and θ_i be the direction of the speed of P_i. Smoothness is then defined as [51]:

$$\sum_{i=\max(k-2,2)}^{k} w_1 \cdot (1-\cos(\theta_i-\theta_{i-1})) + w_2 \cdot \left(1-2\frac{\sqrt{v_i v_{i-1}}}{v_i+v_{i-1}}\right) \quad \textbf{(EQ 26)}$$

where w_1 and w_2 are weight factors.

The smoothness, such as it is defined in [51], that corresponds with a totally smooth trajectory is zero. A higher smoothness indicates a

less smooth trajectory. Therefore, the name 'roughness' would have been more appropriate.

11.5.3 Extension of Hwang's algorithm for Sybar

Several extensions were added to Hwang's algorithm to improve the performance. Additional information that is provided by the feature detector is the size and radius of the features. This information is used by adding components to the quality function: the sum of the qualities of the elements and the sum of the size differences are added (with weight factors) to **(EQ 26)**. We define q_i as the quality and s_i as the size of feature i.

A problem we encountered during the implementation is that if features move with low speed, for example with a speed of one or two pixels, the change of the direction of the speed is no longer smooth. Since slow motion is quite common, we therefore had to lower the weight of the direction component. We also added an additional component, the distance component, which favors trajectories with slow moving features. Our final quality function has the following components and experimentally determined weight factors:

TABLE 3 Components of the quality function, with weight factors

Component	Calculated as	weight factor
direction	Sum of direction differences $\sum_{x=\max(k-2,2)}^{k} (1-\cos(\theta_x-\theta_{x-1}))$	-10
velocity	Sum of velocity differences $\sum_{x=\max(k-2,2)}^{k} \left(1-2\frac{\sqrt{v_x v_{x-1}}}{v_x+v_{x-1}}\right)$	-200
distance	Sum of distances: $\sum_{x=\max(k-1,1)}^{k} v_x$	-1

TABLE 3 Components of the quality function, with weight factors

Component	Calculated as	weight factor
marker quality	Sum of marker qualities: $\sum_{x=1}^{k} q_x$	20
marker size	Sum of marker size differences: $\sum_{x=\max(k-1,1)}^{k} (s_x - s_{x-1})$	-6
feature votes	Number of times trajectory was chosen by feature as best trajectory	250
phantom points	Total number of phantom points used in the trajectory	-75

11.5.4 Matching trajectories with model points

Design Decision:

Hwang's tracking algorithm results in a number of trajectories. We are interested in matching these trajectories with model points. For this, it is enough to know the correspondence of trajectory elements to model points for a single frame. This can be solved by letting the user indicate the correspondence for a single frame, a process which we call *labeling*. This leads to the following setup:

1. The user labels a frame which contains all the markers. This gives us a *labeled frame,* for which the correspondence is known.
2. Perform Hwang's tracking algorithm.
3. Translate the result to the position dependent model parameters.

A problem with this approach is that there is no guarantee that the features in the labeled frame are all used in one of the trajectories that are found: Hwang's algorithm does not use the fact that the labeled markers are correct features. We therefore modify Hwang's algorithm in the following way: instead of keeping the best trajectories for each of the features in the current frame, we keep the best trajectories for each of the features in the labeled frame. This

ensures that at least one trajectory is available for each of the labeled features.

11.5.5 Generalization to 3D tracking

As was shown in Section 9.9.3, it is possible to reconstruct a 3D point via two 2D projections. In order to perform this 3D reconstruction, the features corresponding to a marker from each of the views should be matched.

The algorithm described in the previous section can easily be extended to 3D in the following way:

1. The user labels each of the camera views.
2. Perform the tracking algorithm for each of the views.
3. Use the tracks corresponding to the same model points to reconstruct the 3D marker positions.

In this approach, the tracking algorithm is identical to the tracking algorithm for 2D correspondence. Points are reconstructed as shown in Figure 24 of Section 9.9.3.

As described in Section 9.9.3 the algorithm can be improved by using the 3D information for tracking. The shortest distance between the rays through the feature points in the images that correspond to the same feature can be computed. If two image features really correspond to the same marker, this distance is small. Therefore, the distance is a measure for the likelihood that two points from two views correspond to the same model point. This adds another component to the quality function.

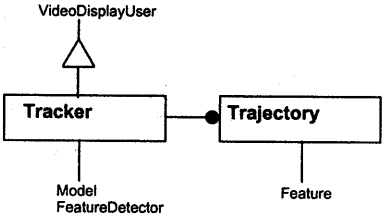
Design Decision: Because the added accuracy of the improvement is limited, compared to the added complexity of the algorithm, we choose to use the simple version of the 3D tracking algorithm.

11.5.6 System design

The object model for the tracking system is simple, since the tracker's main task is to perform the tracking algorithm. There is a *Tracker* class, that uses the features collected by the *FeatureDetector* and a *Model* to determine position dependent model parameters.

As an intermediate result, there are also *Trajectories*. *Trajectories* consist of an ordered list of features.

FIGURE 36
Object model Tracker



Tracker
parameters
start_labeling
mouse_click
clear_labeling
track
generate_plausible _trajectories
select_best
reconstruct3D

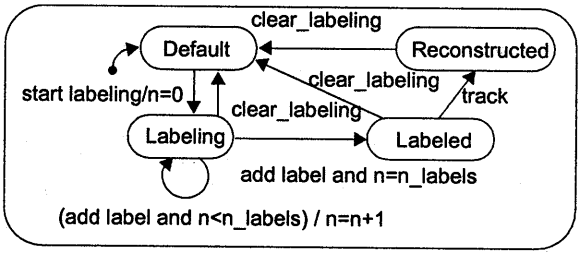
11.5.7 Tracker

The *Tracker* takes care of finding tracks of feature points and matching them with model points. For this purpose, it uses the extension of Hwang's tracking algorithm described earlier in this chapter (Section 11.5.3).

First, the user is required to label a frame in the video. The labeling is started via the *start_labeling* operation. During the labeling, the *Tracker* uses the *VideoDisplay* as a background and therefore is a *VideoDisplayUser*, see Section 8.2.10. Each *mouse_click* leads to an additional point that is labeled. If all the model points are labeled, then there is a labeled frame, and the tracking can take place. The labeling can be canceled via the *clear_labeling* operation.

The *track* operation uses the *generate_plausible_trajectories* and *select_best*, as described in the previous sections. The *reconstruct3D* operation matches the trajectories with model points and reconstructs the 3D model points.

This leads to the following dynamic model:

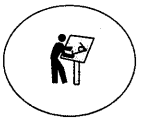


CHAPTER 12

Design Methodology

"A good scientist is a person with good ideas, a good designer is a person who makes a design that works with as few original ideas a possible."

- Freeman Dyson



This chapter discusses the choice of a design method for Sybar. First, we discuss design and design methods in general. Next, we focus on several aspects of software design methods. Finally, the choice for a particular method and design strategy for Sybar is motivated.

12.1 Design & design methods

A characteristic of an engineering discipline is that the design process is structured by means of a design method. In this chapter, we discuss the choice of a design method for Sybar. We start, however, with the meaning of the word *design* itself.

12.1.1 What exactly is (a) design?

The word 'design' is used in a number of ways. According to the Oxford dictionary, a design is a:

1. drawing or outline from something to be made, the art of making such drawings
2. general arrangement or planning
3. pattern; arrangement of lines, shapes, details, as ornament
4. purpose; intention; mental plan

In engineering, there are two rather different uses for the word design. Usually, a design is a plan, drawing or outline that can be used to make a product. However, the word design is also used to indicate the style or appearance of a product (similar to definition 3 from Oxford).

For instance in software, the 'design of an application' can mean the fundamental structure or plan on which the software is built, or the 'look and feel' of the application. In the context of a Ph.D. designers project, the emphasis is on the first meaning.

12.1.2 Phases of the design process

One of the fundamental abilities that is taught in engineering education, is the ability to develop a product in a structured and systematic way. For this, every discipline has its own methodologies and design languages. However, in general, it can be stated that the development of a product involves the following activities:

- analysis of the problem domain
- determination of the requirements
- design
- realization of the design

Most development methods define phases that correspond to these activities. The method then consists of carrying out each of the phases in a specified order. In practice, this is an idealized description of the design process and there are often many iterations necessary between the phases. The amount of iteration depends on the discipline and the innovative character of the design. Versions of the realization of a design that are not final are called *prototypes*.

Design can be seen as a phase in the development of a product. However, because design is considered the most important part of the process, a development method is also often referred to as a design method.

12.2 Software design methods

A problem of software design is, that it is still not clear what level of detail a software design should have. Should a design describe a computer program to the level of programming code? The general consensus in software design methodology seems to be that it is sufficient to design systems up to a level were writing the code becomes a routine job. However, this border between design and implementation is at best arbitrary. In practice, the detail of the design is often limited by what can be expressed in the language of the design method.

According to Sommerville, a software design method consists of [69]:

- A process model, describing the activities in the method
- System modeling notations and rules
- Design guidelines
- Report templates

Programming in a specific programming language can be seen as a (primitive) design method: there are activities to do, there are notations, rules, guidelines and a print-out of the code can be made. The step from global to detail is made by starting with 'abstract' or 'pseudo'-code, and then filling in more and more final code. Programming methods were the first software design methods to be used, and they were (and are) adequate for small systems.

However, when software systems became more and more complex in the early seventies, it was realized that methods were required that took a more abstract high level view of system design. These methods introduced graphical notations to describe designs. The methods were based on the popular programming languages of the particular era (Fortran, Cobol, C). These first-generation design methods are now known as *functional design methods*. Examples are *Structured Design*, *SSADM* and *Jackson Structured Programming*.

In 1967, the concept of object oriented programming was introduced in the language Simula. It was not until the early eighties that object oriented programming language became widely used (in particular C++ and Smalltalk). These languages were the inspiration

for a new generation of design methods: object oriented design methods. As with the functional methods, the first object oriented design methods were very closely language related [62]. These were followed by more general, and more graphical, methods. Examples are *OMT* [66], and *Booch* [67].

There are a number of concepts in object oriented development, that work very well together and explain the success of object orientation. The biggest difference with functional development is that the system under development is not structured into functions, but into classes of objects, containing both data and functions. Well designed object-oriented software is more flexible in the sense that a small change in the required behavior of the system is likely to have only local impact. This leads to faster development, increased quality, easier maintenance, and enhanced modifiability. A short introduction to object oriented development is given in Appendix A.

It is often claimed by design method developers that their method is 'program language independent'. However, we feel that there is a strong relation between design methods and program languages. In particular, the design method and the language have to match in two ways:

1. The design must be relatively straightforward to translate into programming code.
2. It should be easy to express the programming concepts and features that are available in the programming language in the design language.

As a general rule, the number of design decisions that still have to be made during implementation should be minimal.

For example, OMT does not support the concept of parameterized types, a feature that is available in Eiffel and C++. In the follow-up to OMT, it was realized that parameterized types are an important concept, that should be taken into consideration during the design phase. Therefore, in the Unified Modeling Language (UML), support for parameterized types is included.

However, when using a programming language that does not support parameterized types, such as Java, they should not be used in the design. Using parameterized types in a Java design leads to the postponing of design decisions to the implementation phase. Of course, the lack of a concept in a design language is more serious than the availability of a concept that should not be used for the implementation in a particular language. Therefore, design methods often try to generalize the concepts of a number of languages.

12.3 Choosing a toolset

There are many aspects that determine the choice of the design method for a project. A method can be seen as a part of the toolset of the software engineer. The choice of a toolset is an important design decision. It can have a large influence on the design.

The choices of the tools in the toolset often depend on each other. We have already mentioned that there is a relation between programming language and design method. Other parts of the toolset to take into consideration may include:

- the development platform
- compilers
- computer aided software engineering (CASE) tools
- specific libraries

There are also external factors that play a role in choosing a method:

- Available knowledge of project members: learning a new method is often seen as overhead.
- Company standards may require specific choices.

12.4 Toolset for Sybar

The following requirements for a design method were determined for Sybar:

- It should be a method that can deal with the many aspects of Sybar: user interface, hardware connectivity, image processing, computer graphics, etc.

- The method should be able to cope with unclear and changing requirements
- A good CASE tool should be available

Furthermore, requirements for the programming language were:

- The language must be supported by a good development environment
- Graphical user interface libraries should be available
- It should be easy to interface with hardware equipment
- The market position of the language should be good: using a language can be seen as a strategic investment. It is not a good idea to choose a language with a small user-base for a large project.

The Object Modeling Technique (OMT), described in [66], was chosen as the design method. OMT is one of the most frequently used object oriented methods (together with Booch [67]). The choice was based on the fact that the method is accompanied by a well-written book, and experiences of other people were, in general, positive. It is difficult to make an accurate judgement of a method without using it for at least a couple of months. However, OMT seemed a safe choice at the time.

The choice for a programming language was relatively simple: C++. The object oriented language C++ is becoming a *de facto* standard in object-oriented programming (although it now has a competitor in Java). The main disadvantages of C++, its difficult syntax and its unsafeness, were outweighed by its advantages in terms of compiler and tool support, particular on the Windows platform, the platform that was chosen for Sybar. Furthermore, OMT and C++ are often used in combination and they seem to match well. Our evaluation of the design method is given in Section 14.4.

Other choices were based on availability and experience in the hospital:

- Platform: Microsoft Windows 3.11
- Compiler: Borland C++
- Windows Toolkit: Borland Object Windows Library

- CASE tool: unfortunately, a suitable CASE tool could not be found

The use of a Silicon Graphics workstation as a platform, instead of PC/Windows was considered. However, it was not possible to acquire a Silicon Graphics workstation with sufficient hardware capabilities within the budget constraints of the project.

12.5 Design strategy

12.5.1 Phasing

As we described earlier in this chapter, phasing is an important aspect of design methods. There are two ways to approach phasing: one is that the phases should be carried out one after the other. This is known as the waterfall model, since you can not 'go up' in a waterfall. The other approach is to allow iterations between phases. Methods that fall in this category are called incremental or rapid prototyping methods.

In describing a method, it is much easier to assume that a waterfall approach is followed, since the activities of each phase lead to clear results that are the input for the next phase. In practice, the waterfall approach is rarely followed. There are two reasons for this:

1. Errors are found that lead to necessary changes in earlier phases
2. It is often not possible to complete a phase without doing some work that belongs to a next phase.

The OMT does not prescribe a phasing strategy. However, in our case, it was quite clear that we would not be able to follow a waterfall approach. The user requirements were unclear and prototypes were necessary to determine them. It was therefore decided to use an iterative approach.

There are two types of iterative approaches: *incremental development* and *throw-away rapid prototyping*. In throw-away prototyping, prototypes are developed purely for the purpose of gaining knowledge which is used in the 'final version'. In incremental development, some parts of the application are developed, and increments are added based on preliminary results.

Incremental development is to be preferred, because work does not have to be re-done. However, the danger of incremental development is that the system becomes unstructured, because not all changes that have to be made can be foreseen. In practice, this often leads to a well-known trade-off: 'restart and do it properly' or do a 'quick and dirty' fix.

For Sybar, we chose incremental development as a phasing strategy. This choice has implications for the design: incremental development requires a well-structured global design that is flexible enough to include the required increments. Object oriented methods are particularly well suited for incremental development, because of the 'enhanced modifiability' property.

During the Sybar project, there were a large number of small increments that led to the final system. In the chapter structure of this design, we use the phasing terminology, which may give the false impression that the project followed the waterfall strategy. The phasing structure was chosen because it corresponds to OMT and because it leads to a better understanding of the design.

12.5.2 Formal versus informal design process

As we have mentioned, it is difficult to manage a design process, because it is a creative process. However, when there are a large number of developers working on a project in an industrial environment, it is absolutely necessary to somehow formalize procedures. Formalization can be achieved:

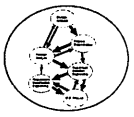
- via tools, for example *version control systems*, *configuration management tools*, and *automatic testing tools*
- via procedures, for example *bug reports* and *change requests procedures*

Tools and procedures are frequently used for coding. However, such tools and procedures are rarely used for high level design. Formalization is usually perceived by designers as annoying and distracting from the creative aspect of the design. When the scale of the projects permits, it is advisable to use informal procedures whenever possible.

The Sybar project is a small scale project in terms of number of developers. Therefore, an informal approach was followed for the entire development process. For example, no special tools were used and testing was performed during code development. Some simple procedures were introduced to deal with multi-developer aspects.

CHAPTER 13

Design Process



This chapter describes the design process of the Sybar project. First, the process is divided into subprocesses. Next, these processes are described, including some of the problems that were encountered. Finally, the project team and the contributions of the members of the team ar described.

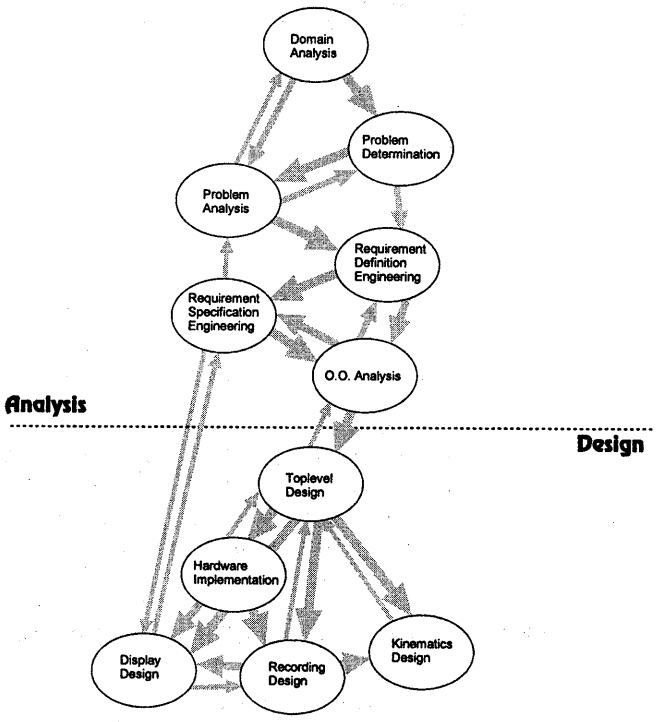
13.1 Subprocesses of the design process

The design process is a creative process that is difficult to describe. A description of a design process is almost inevitably a simplification. Nevertheless, this chapter tries to give an impression of the design process that lead to Sybar.

In Figure 37, development process is divided into a number of subprocesses, similar to the structure of the thesis. An arrow indicates the flow of information from one process to another. Only the most important dependencies are shown.

In the remainder of this section, each process is described in more detail. A description of the process is given, including the field that most design aspects were related to. The implementation and testing of subsystems are not distinguished as separate processes. The design, implementation, and testing of a subsystem were very much related. They are described as a single process.

FIGURE 37
Process dependencies



13.1.1 Analysis processes (total time 25%)

Process	Design area	Result described in	Time spent
Domain analysis		Chapter 2	2%

Domain analysis took place by studying medical literature, and by studying current practice in the hospital. A lot of knowledge was available via Jaap Harlaar, a biomedical engineer who has a background in electrical engineering. He has been working at the rehabilitation department for over ten years. These ten years were obviously not counted within the 2%.

Process	Design area	Result described in	Time spent
Problem determination		Chapter 2	1%

The basic problem is well known in biomedical literature: why are biomedical measurement techniques not successful in clinical practice? The problem determination can be seen as the first step from a general knowledge building process (domain analysis) to a project specific analysis. It answers the question: what is the purpose of the project?

Process	Design area	Result described in	Time spent
Problem analysis	Biomechanics, Rehabilitation medicine, Computer Science	Chapter 2	3%

The problem analysis led to the initial approach. Most of the ideas in the initial approach came from Harlaar, who wrote the first project proposal. The initial approach did not change a lot during the project. It is the first process that involves design aspects. In particular, the whole idea of solving the problem by means of a multi-media system is a major design decision.

Process	Design area	Result described in	Time spent
Requirement definition engineering	rehabilitation medicine, computer science	Chapter 3	3%

The initial approach was translated into a set of general requirements. These requirements were used as guidelines in the design process. The guidelines were developed and improved during the

project, as more and more it became clear what the major issues in the project were.

Process	Design area	Result described in	Time spent
Requirement specification engineering	rehabilitation medicine, user interface	Chapter 4	10%

The requirement specification is a more detailed description that has many user-interface design decisions. This specification was continuously improved via user feedback of the prototypes. Determining the hardware configuration is considered to be part of the design phase. Therefore, it is not part of the requirements.

Process	Design area	Result described in	Time spent
O.O. Analysis		Chapter 5	6%

An important aspect of object oriented software development is modeling the system as it is seen from the outside. We found that the object oriented analysis (OOA) had little impact on the requirements (see Section 14.4.1). The analysis model was useful as a starting point for design. We used OOA to describe the complete system, not just the software system.

13.1.2 Design processes: total time (75%)

Process	Design area	Result described in	Time spent
Toplevel design	software engineering, electronics	Chapter 6	3%

The first 'pure' design activity was the design of a basic hardware configuration that could be used as a basis for the software design and for developing the prototypes. The hardware design was changed a number of times because of problems in the implementa-

tion. The toplevel software design was reasonably stable during the project.

Process	Design area	Result described in	Time spent
Recording design	software engineering, electronics	Chapter 7	24%

Most of the problems encountered while designing the recording subsystem were in the area of controlling the hardware devices. Controlling the video card was much more difficult than anticipated. The problems were caused by unclear specifications and 'strange behavior' of the hardware.

Process	Design area	Result described in	Time spent
Display design	software engineering	Chapter 8	17%

The display subsystem had the most iterations resulting from updated requirements. These requirements were mostly in the area of data presentation and user interaction. Implementing the new requirements was relatively straightforward.

Process	Design area	Result described in	Time spent
Kinematics design	biomechanics, computer vision, software engineering	Chapters 9,10,11	24%

In order to retrieve kinematics from the video, a marker detection algorithm had to be developed. This was more difficult than anticipated. The initial algorithm was very sensitive to changes in the lighting conditions, and was only suitable for the analysis of a single leg in the sagital view. It was therefore decided to develop a more advanced algorithm This is the algorithm described in Section 11.3.1. Furthermore, the possibility was added to manually correct the marker detection algorithm by pointing at missed markers with the mouse.

Camera calibration consists of determining the parameters of the camera view (position, orientation, focal distance) from one or

more video images, see Section 9.6.2. The camera calibration for Sybar was first developed and tested for the purpose of displaying the force vector at the correct position in the video display. However, camera calibration was also necessary for determining 3D kinematics. During the development of the 3D kinematics part, it was concluded that although the accuracy of the calibration was suitable for force vector display, it was not sufficient for 3D kinematics detection. In order to improve the accuracy, a more stable calibration algorithm was implemented. Furthermore, a calibration structure consisting of 15 'known' points was developed by the mechanical engineering group.

For reasons described in Section 9.11, the first intention was to develop an algorithm that could perform kinematics using a single camera view. As described in Chapter 10, it was decided that this was not feasible. This was a major setback in the project. Next, a multi-camera approach, as described in Chapter 11, was followed.

Process	Design area	Result described in	Time spent
Hardware implementation	electronics	Appendix B	7%

Hardware implementation and maintenance took place during the entire project. It took a long time to realize a properly functioning hardware implementation, because components had to be selected and bought. Furthermore, it turned out that in the initial implementation, synchronization of data could not be achieved because the video digitizer could not interpret the time stamps. This lead to the solution described in Appendix B. The synchronization problems led to a delay in the introduction of the first version into the clinical practice. Another problem caused by hardware malfunction resulted in buying a new data-acquisition board.

13.2 People

In Table 4, an overview is given of the activities that were performed by various team members. Most team members were employees of the VU Hospital and worked in department of Clinical Physics and Engineering. Two students from the Hogeschool Amsterdam also worked on the project as part of their studies.

TABLE 4 Development and support activities of team members

Team Member	Function	Topic of process	Period
Peter Tump	Software Engineer	Data acquisition part of recording subsystem, development of 3D kinematics display	Last quarter 1994, First quarter 1995, Fourth quarter 1996
Rob Peters	Software Engineer	Data acquisition part of the recording subsystem	Full project
Ronald Bonneveld	Software Engineer (student)	Force data visualization	Second quarter 1995
Andre Visser	Software Engineer (student)	A more advanced marker detection algorithm	Second quarter 1996
Cor Klok, Danny Koops	Mechanists	Markers and calibration frame	Fourth quarter 1996
Rob de Bree, Jan Rutger Kuiper	Computer Technicians	Hardware Support	Full project
Edwin Hautus	Software Engineer/ Project Manager	Worked on everything	Full project

CHAPTER 14

Evaluation

This chapter evaluates the Sybar project. Both the developed product and the design process are evaluated. Finally, some lessons learned during the project are described.

14.1 Evaluation of Sybar

The evaluation of Sybar is divided into the following three parts:

- Evaluation of the main goal: did we succeed in building a system that is useful for the clinical practice?
- Evaluation of the system with respect to the requirements definition. To what extent did we meet these requirements?
- Evaluation of the design method. What have we learned with respect to the software design process itself?

14.2 Evaluation of the main goal: clinical usefulness

The main goal of Sybar was to build a system that would find acceptance within the clinical practice. A proof of the fact that we succeeded in achieving this is that a preliminary version of the system was used during the last year of the project. Furthermore, the responses of the physicians were very positive ("I could not do without it anymore!"). The system was used intensively by two physicians of the rehabilitation department. Furthermore, demon-

strations were given to many other physicians in and outside the hospital. The reactions were, in general, very positive.

Since the end of the Ph.D. project, plans have been developed to introduce a second system in another rehabilitation centre. Furthermore, a special teleconferencing version of Sybar will be developed. This is further evidence that the system is valued within the medical community. There are also plans to develop a commercial version together with an external company.

The main advantages of using Sybar that are mentioned by the physicians:

- It supports the clinical decision process by providing objective information
- It is very useful for communication purposes: it used to be difficult to discuss a patient's condition without the patient actually being there. Now, it is much easier to discuss the condition of patients which are difficult to diagnose, together with colleagues in front of the computer.
- It is very useful as an education tool, to show certain cases to medical students, and sometimes to show patients what is causing their problems.

Of course, the comments of the physicians do not prove that using the system results in a better diagnosis of the patients. In order to really prove the clinical usefulness, a large test would have to be performed resulting in statistical evidence that a better diagnosis can be made by using the system. However, such a test was obviously not within the scope of the project.

The physicians have very few comments on the usability. Once they got used to the user interface, it was very easy for them to use it. It must be said however, that the physicians that have used the system up to now, have a relatively positive attitude towards new technology.

There are also a number of 'wishes' for extensions of Sybar: the most important ones are a printing facility and a measurement database.

14.3 Evaluation of the system with respect to the requirement definition

Measurement information is to be presented in a way that is useful for physicians

The physicians are satisfied with the way the information is presented in the final version. Particular useful is the EMG graph in combination with the video. Experienced physicians learn to interpret the graphs. Their interpretations are tested by watching the movement in the video together with the graph.

Navigation through measurement data should be simple and intuitive for novice computer users

The physicians are very pleased with the way they can navigate through the data. The control panel is used for viewing the video at normal speed, usually at the beginning of a session. The control panel is also used to navigate frame by frame. The graphs are used to jump to interesting points. The EMG gauges are used a lot for showing and hiding EMG channels in the displays.

The lab operators should find the system easy to use and should be able to perform data acquisition and data processing quickly

The usability of the system from the point of the lab operator still needs improvement. One of the problems is the complexity of the hardware configuration. A new hardware setup is currently under investigation. Another problem is the amount of work necessary to make a 3D measurement.

The measurement system has to be patient friendly and safe

The system complies to the hospitals safety rules. There is still room for improvement in the area of patient friendliness: the recording of measurements still takes about half an hour and involves a lot of intimidating technology. Most of the technology is part of the measurement system which was not the focus of the project.

The system should be reliable

The physicians were satisfied with the reliability of the system. The system does occasionally crash. The crashes are attributed to hard-

ware failures of the video digitizer. A new 'next generation' digital video card is planned.

The system should be maintainable.
Maintainability during the project was not a big problem. The biggest problem with regard to maintenance was keeping the documentation up to date. This was caused by the lack of an integrated design and implementation environment. There is a lot of overhead in keeping the requirements, design, and implementation up to date. In the next section, we have more comments on the design methods. Currently, the maintainability of the system is tested by the request to adapt it to new hardware and a multi-user option.

14.4 Evaluation of the design methods

14.4.1 The design method OMT

As mentioned before, we used OMT as our general design method. In general, we are positive about the method. In particular, the use of diagrams helps to keep a good overview of the design. The diagrams definitely give a better insight in the system than programming code. Furthermore, the diagrams are an aid in the communication between developers. We have some criticism on the OMT/C++ combination. Some of the points that caused this criticism are solved in the follow-up to OMT, the new modeling method UML (Unified Modeling Language), which is currently under development [73].

We found that the distance between design and implementation is too large. The object diagram and the programming code have a clear relation. However, the object diagram describes only the static behavior of the system. We found the state diagrams insufficient to show the global dynamic behavior resulting from object interactions. UML provides sequence diagrams and collaboration diagrams for this purpose.

Furthermore, the functional model was of little use to us. Apparently, we are not the first to notice this, since the functional model has been dropped in UML, the follow-up method.

In OMT, associations (use-relations) have no direction. The direction of an association is considered to be an implementation issue. We feel that the direction of associations is a design issue, especially when designing for re-use. It is important to decide which objects know of other objects. In UML, an association can be given a direction.

Another problem encountered was the lack of a subsystem concept in OMT. At a certain level of detail, it becomes impossible to show all the classes in a single diagram. However, OMT has no way of structuring systems of classes. Again, this problem is fixed in UML with the package concept.

One of the most serious problems we had with the OMT/C++ combination was that there is such a large step between design and implementation. It is a lot of work to keep the design and the implementation up to date during the many iterations. This is also caused by the lack of an integrated environment that deals with both the design and the implementation aspects of the development. Most existing environment have either a strong focus on design or a strong focus on implementation.

Finally, we found the object oriented analysis of little use in determining requirements in our case. Requirements were almost exclusively determined via informal conversations and via the use of prototypes. We think that OMT analysis might be more useful for determining requirements in (pure) information systems. May be, the UseCase models introduced in UML are of more use for determining requirements.

14.4.2 Evaluation of the design strategy

In Section 12.5.1, we motivated our choice for incremental development as the phasing strategy. Basically, a waterfall approach was not possible, and incremental development was considered more desirable than throw-away prototyping. We found that we managed reasonably well in following the incremental approach. The global design of the toplevel, the recording subsystem and the display subsystem was defined within the first year of the project, and it did not have to be changed very much.

However, for the kinematics subsystem, we decided to use a throw-away prototyping approach. The main reason was the uncertainty

whether the algorithms would actually work correctly. We developed a simulation environment to test algorithms for the single camera approach. This eventually lead to the 'throw-away' of the prototype, because it was concluded that this approach was not feasible. We managed to obtain this result with a subsystem that was not 'production-quality'. If we had built a production quality subsystem, it would have taken much more effort to figure out that the approach was not feasible. Therefore, we think that the phasing strategy was the best choice.

In section 12.5.2, we motivated our choice for using an informal design approach. In practice, the informal approach worked fine. At certain times, we did have to be careful with sharing code between developers. A source code control system might have been handy, although there were no major problems in this area during the three years of development.

The lack of an integrated design and implementation environment was felt as a major problem. It was a lot of work to keep the analysis, design documentation, and implementation up to date. For example, it was quite common that a new name for a class was chosen, because the name was thought to be a better description of what the class represented. A perfect designer in an ideal world may be able to determine exactly the correct concepts and represent them in class names that do not need to be changed. However, in our experience, new roles for objects lead to new insights and name changes. Even simple name changes are a lot of work if there is no case tool support.

14.4.3 Lessons learned

Note: since the following comments are personal, a switch is made from 'we' (the development team) to 'I' (Edwin Hautus).

Having studied the theory of object oriented analysis, design and programming, my idea was that these three activities are clearly defined and can easily be distinguished during development. However, this view clearly changed during the project.

A lot of design takes place during analysis

Software development is often separated in analysis, design, and implementation. During the project, it became more and more clear to me, that there are many design aspects involved in the analysis. For example, determining the high level requirement definitions, and the design of the user interface, were some of the most interesting design activities in the project. Both activities clearly have the purpose to determine *what* the system should do, and therefore are analysis activities.

The gap between design methods and programming languages is too large

The use of a specific design method, next to a programming language, is still not yet completely accepted in the object oriented world. Interestingly enough, the programming language designers, such as Meyer[71] and Stroustrup[72], are those who question its use. They claim that the design language distracts from the programming language itself, which has enough possibilities to describe a high level design.

At the start of the project, I was clearly in the camp of the design methodologists. However, during the project, my view changed. I still think object oriented design methods are very useful for design. However, I also think that the distance between design methods and programming languages is currently too large. Too much work has to be done during coding. Either the programming languages need to incorporate design method concepts, or the design methods need to be able to describe the design to the level of detail of the code. In particular, the amount of (design) work to implement an OMT design into C++ is too large.

This problem is related to the lack of good CASE (computer aided software engineering) tools. However, as long there is a 'conceptual gap' between the design method and the implementation language, CASE tools can not really solve the problem. There are simply some important decisions that have to be made when implementing an OMT design that can not be easily automated.

The possibilities of structuring the design process is limited by organizational aspects

The intention was to involve physicians in the design process from the beginning, since they are the end-users. It would seem that the

location where the project took place, a hospital, would be ideal to achieve this. However, because of organizational aspects this was not feasible. Physicians have no time and no interest in participating in the design process. This seems to be a re-occurring problem in hospitals. Furthermore, when a number of people are involved in a project, it is not possible to force them all into a different (more structured) way of working. The changes have to be introduced gradually.

The ability to make the right abstractions is the most important skill for an object oriented developer

Abstractions have always been a key aspect of computer science. Abstractions have become even more important with the rise of object oriented development. Object orientation allows software to be structured in a much nicer way, *if and only if* the right abstractions are made. If the right abstractions are not made, object orientation becomes a burden in stead of a help.

APPENDIX A

Object Oriented Methods & OMT

This appendix gives a short introduction to object oriented methods and the Object Modeling Technique (OMT). Furthermore, it introduces two small extensions of the notation that are used in the thesis.

1.1 A very short introduction to object oriented methods

In traditional programming, datastructures and functions/procedures are seen as the basic building blocks: functions act upon datastructures. In object oriented programming, the *object* is the basic building block. Objects consist of both data and functions, and are organized in *classes*. The objects in a class are identical in structure, but each object in a class, called *an instance* of that class, has its own data.

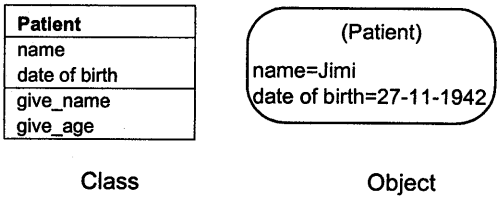


FIGURE 38
Graphical representation of a class and an object of that class

One advantage of Object Orientation (O.O.) is that real world objects can be modeled more easily. We will show this using a (simplified) *Patient* class as an example. The data elements, or *attributes*, of a patient are *name* and *date of birth*. Furthermore, the *Patient* class has two functions, or *operations*, to inspect these attributes. The operation *give_age* returns the age of the patient and the operation *give_name* returns the name of the patient.

Another key-concept of O.O. is *encapsulation.* The idea of encapsulation is to make a clear distinction between operations and attributes of an object that can be used by other objects and those which are private to the object. In our example, the age of the patient, which can be requested with the *give_age* operation, is not stored in a patient object. The age is calculated using the private *date of birth* attribute. However, the way that the age is calculated internally is irrelevant to other classes and may be changed without effecting those other classes.

Another key issue is *inheritance*: subclasses can inherit from superclasses. Classes that inherit from other classes have all the attributes and operations from the superclasses, but can add their own. An example for our patient class is a special class for patients with a certain disease. Additional information common to these patients can then be stored and operated on. Subclasses can also redefine functions of superclasses, allowing for more efficient implementations of certain specialized cases.

Sometimes, classes are introduced that can not be instantiated. These *abstract* classes contain operations or attributes that are common to a number of subclasses. For example, in Sybar we have the class *Display*. A *Display* shows part of the measured data in a certain way. Two subclasses of *Display* are *EmgGraph* and *VideoDisplay*. A *Display* cannot be instantiated: it is an abstract class. It contains information common to *EmgGraph* and *VideoDisplay*, for instance each display can be either on or off.

A final characteristic we mention is *polymorphism*: operations and attributes can be given the same name, but a different behavior, depending on the class that is involved. For instance, both the patient class and the recording class can have a *load* function, to load a patient or a recording from disk.

1.2 The object modeling technique

The object modeling technique (OMT) described in [62], is a methodology for the object-oriented development of software systems. The methodology consists of building a model of an application domain during analysis and then adding implementation details to it during the design. OMT consists of the following phases:

- *Analysis*: In this stage, the problem statement is determined. Furthermore an analysis model is developed that describes a specification of the system.
- *System Design*: In this stage, the global architecture of the system is determined.
- *Object Design*: In this stage, the analysis model is developed in greater detail, with objects added for implementation.
- *Implementation*: In this stage, the design model is translated into computer code, usually an object oriented language.

Ideally, each phase is completely finished before the next one is started. In practice, an evolutionary approach is usually followed. There can be overlap between stages: objects can be in different stages of development.

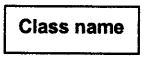
1.2.1 OMT diagrams

The OMT methodology uses three models to describe a system: the object model, the dynamic model and the functional model. The *object model* describes the classes and objects in the system and their (static) relationships. The *dynamic model* describes the states an object can be in and the interactions between objects. The *functional model* describes data value transformations in the system. The object model is considered to be the most important in most applications. We will now describe the concepts and notations of the three models.

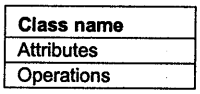
Object model notation

The object model describes the static structure of a system. It describes classes, relationships between classes, and attributes and operations of the classes.

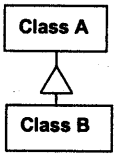
Classes are drawn as rectangles:



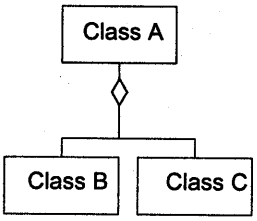
Attributes and operations can optionally be displayed in classes as follows:



Inheritance (an object of class A *is* a special type of an object of class B) is shown by means of a triangle:



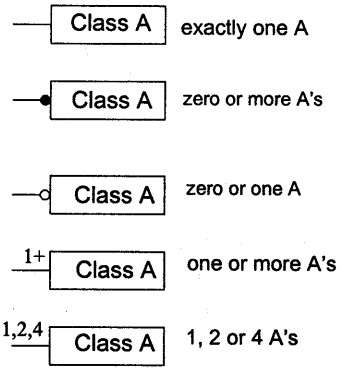
Aggregation (an object of class A *consists of* parts B and C) is shown as:



External classes are shown as rectangles with dotted lines.

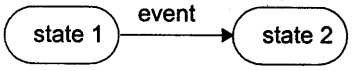
Associations between classes indicate there is a relationship between two classes. Names of associations can optionally be shown in the diagram.

Dots indicate the multiplicity of the association:

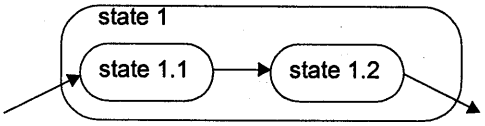


Dynamic model notation

The dynamic model describes the states that an object can be in. Transitions between states are triggered by events, that are produced elsewhere in the system or that are received from the environment of the system. States are indicated by rounded rectangles, events by arrows:



States can have sub-states and superstates, forming a hierarchical structure.



The arrows contain an inscription of the following form: *event A[guard B]/event C.* This transition occurs when event A occurs and guard B holds. As a result of the transition, event C is generated.

Initial and final states of objects are shown as:

A default state is entered when a arrow ends at the boundary of the superstate.

Functional model notation

The functional model describes the functional behaviour of the system in terms of input and output values and data-dependencies. Processes are used to indicate computations:

Data flows, necessary for computations, are shown by means of arrows:

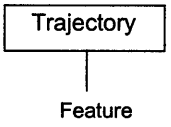
A data store is a passive object that stores information for later use. They are shown as:

Actor objects are objects that drive the dataflow by producing or consuming data values. They are drawn as rectangles:

1.3 Two small extensions of OMT

In order to improve readability, two extensions of OMT are used in the object diagrams of this thesis:

- The direction of association names is indicated with a black triangle. In standard OMT, the direction has to be determined by the reader, which can sometimes lead to confusion. This is solution is also used in the Unified Modeling Language (UM L).
- Associations to classes that are not in the diagram, are shown by smaller text without a class box:



Trajectory has an association with the *Feature* class, which is described in another diagram. This extension is a 'patch' for the lack of a subsystem concept in OMT.

APPENDIX B

Hardware Implementation

Most of the hardware that was to be used for the Sybar project was bought specifically for the project. This appendix describes the hardware components and the way they are connected.

2.1 Measurement devices

The Sybar system uses the following measurement devices:

- An EMG measurement system, developed by K_Lab.
- An AMTI Biomechanics Platform, manufactured by Advanced Mechanical Technology, Inc.
- Two camera's for recording the video.

Multiple camera support is realized via a video mixing device as described in Section 6.3.1.

2.2 Synchronization

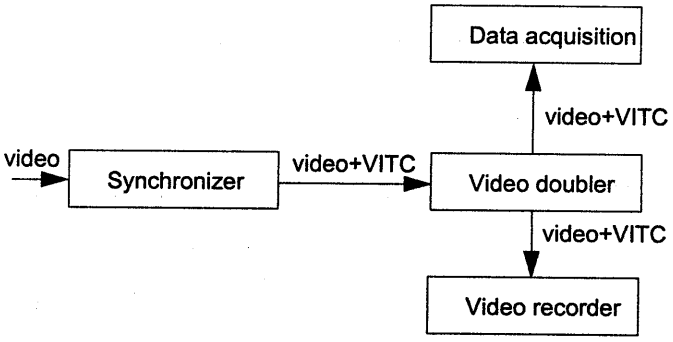
The synchronization module, see Section 6.3.2, is realized with the Vertical Time Code (VITC) standard. The VITC is a signal that can be added to analog video.

The video is first send to the synchronizer, which adds the time code to the video. Next, the video data is doubled by a video dou-

bler device. One copy of the video stream is recorded by the video recorder. The other copy of the video stream is presented to the data acquisition system. The data acquisition unit records the time code simultaneously with the EMG and the force plate. This leads to two datasets with a common time scale.

The hardware setup for the synchronization is shown in Figure 39.

FIGURE 39
Synchronization setup



2.3 Video recorder

The video recorder is controlled by the computer via a video recorder controller device. The video recorder controller provides a link between a computer and a video recorder. The computer can communicate via a serial port, while the video recorder is controlled by sending infrared signals, which are also used by the remote control. There are also video recorders that can be controlled via a computer directly. However, by using the video recorder controller, it possible to use an ordinary video recorder.

2.4 Video digitizing and hardware compression

Digitizing video involves large amounts of data. For example, one second of PAL video (25 frames per second) stored with 24 bit color depth costs about 32 Mega bytes. In order to deal with such large amounts of data, it is necessary to compress the video with a dedicated hardware compression device.

After extensive market research, it was concluded that the Videologic Mediaspace video digitizer provided the best possible video quality that could be afforded. The Videologic Mediaspace is a hardware extension for IBM compatible PCs, that allows analog video to be viewed on the PC. Furthermore, video can be digitized, stored on hard disk and played back.

For the compression, the Videologic Mediaspace uses a MJPEG compression scheme. MJPEG compression is a video compression scheme based on the single image compression standard JPEG (Joint Photographic Experts Group). The MJPEG standard applies JPEG compression on each of the frames of the video. Unlike MPEG compression, MJPEG does not use similarity between frames to further compress the data. MJPEG is a lossy compression scheme. This means that information is lost during compression. The restored video is therefore not exactly the same as the original video. The effect of JPEG compression is that the whole picture loses some contrast, although the effect is not noticeable with low compression rates. The effect for the marker detection is that in some images, certain markers can no longer be detected. However, this only takes place in cases that are difficult in the first place. It can be compensated by better lighting conditions.

In order to synchronize the video data with the other measurement data, the time code signal has to be retrieved during the digitizing of the video. Unfortunately, the Videologic digitizer card does not support time codes. Furthermore, it is not possible to have any control over the video digitizer during the digitizing.

The first solution to this problem that was tried was to read the time code with a special device, a timecode reader. By storing the time code just before the digitizing starts, the time code of the first frame is known. Since the video is recorded with a fixed sample rate, the time codes of the other frames can then be calculated. However, this approach is not fail proof, since the video digitizer has a delay of a variable length before it actually starts recording the first frame.

We therefore had to add another device, a time code displayer, that writes the time code into the video image. Using this time code, the lab operator has to check and, if necessary, correct the synchroniza-

tion. If a new video digitizer is found which includes support for VITC, the synchronization can be performed fully automatically.

2.5 Data acquisition

For the data acquisition of the measurement data, an A/D converter device is used. The A/D converter digitizes EMG and force plate data. To retrieve the time code signals from the video stream, a timecode retriever is used.

2.6 List of devices and overview

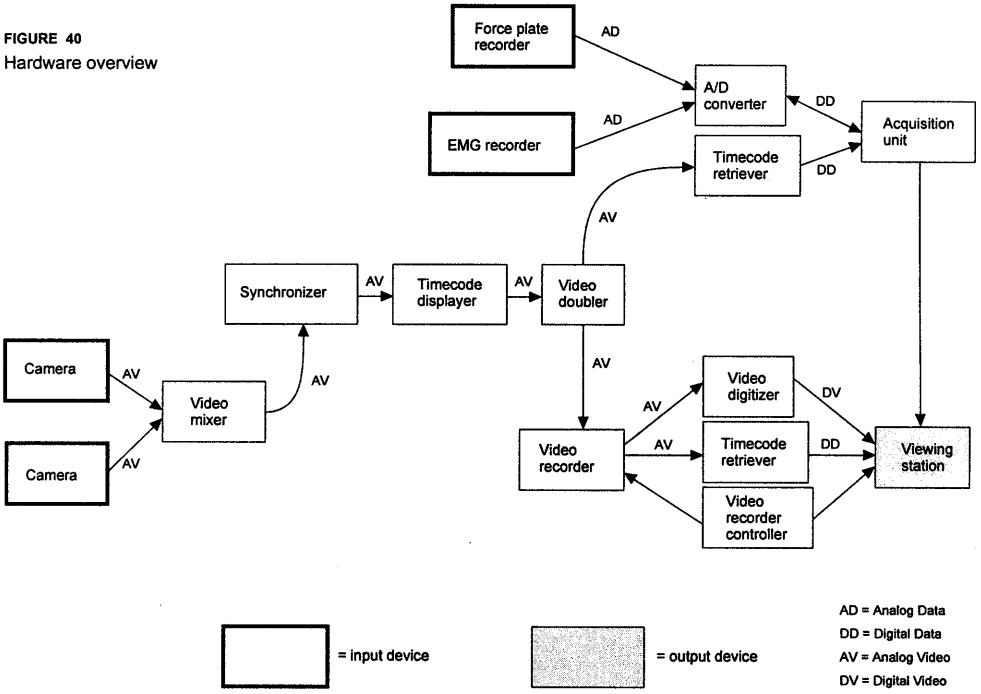
In Table 5, a list of the hardware devices is given.

TABLE 5 List of devices

Function	Device	Manufacturer
EMG recorder	K_Lab M108	K_Lab Amsterdam
Force plate recorder	AMTI force plate	AMTI
A/D converter	PCL 818	Advantech
Synchronization module	TC 30 G generator	Alpermann+Velte
Timecode displayer	TC 30 reader/ inserter	Alpermann+Velte
Video doubler	DV-101 Video Distributor	JVC
Timecode retriever	PCL-10	Alpermann+Velte
Camera 1	CL352 Camera	Panasonic
Camera 2	CR-6200S Camera	Blaupunkt
Video recorder	RTV-925 Hifi Video recorder	Blaupunkt
Video recorder controller	PE 800 Motion Link	Alpermann+Velte
Video mixer	WJ-5500 Special Effects Generator	Panasonic
Video Digitizer	Mediaspace/ DVA 4000	Videologic

Figure 40 gives an overview of all the devices.

FIGURE 40
Hardware overview



Bibliography

TABLE 6

Abreviations	
AI	International Journal on Artificial Intelligence
CVPR	Proceedings Computer Vision Pattern Recognition
CVGIP	Computer Vision Graphics & Image Processing
CVIU	Computer Vision, Image Understanding
CGIP	Computer Graphics & Image Processing
ECAI	European Conference on Artificial Intelligence
ICCV	International Conference on Computer Vision
ICPR	International Conference on Pattern Recognition
IVC	Image and Vision Computing
JOOP	Journal of Object Oriented Programming
JVCIR	Journal of Visual Communication and Image Representation
PAMI	IEEE Transactions on Pattern Analysis and Machine Intelligence
PR	Pattern Recognition
PRL	Pattern Recognition Letters
SIGGR	ACM Computer Graphics

TABLE 6

Abreviations

SMC	IEEE Transactions on Systems, Machines and Cybernetics
MBEC	Medical & Biological Engineering & Computing
JB	Journal Biomechanics
APMR	Arch Phys Medical Rehabilitation
BIO	Biolocomotion: a century of research using moving pictures
BRMI	Behavior Research Methods & Instrumentation
JMET	Journal of Medical Engineering & Technology
PT	Physical Therapy

Human Motion Analysis - Biomechanics

[1] Tait, J.H., Rose, G.K., *The real time video vector display of ground reaction forces during ambulation,* JMET 3 no.5, 1979, p.252-255

[2] Boccardi, S., Pedotti, A., Rodano, R., Santambrogio, G.,C., *Evaluation of Muscular Moments at the lower limb joints by an on-line processing of kinematic data and ground reaction,* JB 14 1981, p.35-45

[3] Brand, R.A., Crowninshield, R.D., *Comment on criteria for patient evaluation tools,* JB 14 1981, p.655

[4] Taylor, Mottier, Simmons, Cohen, Pavlak, Cornell and Hankins, *An automated motion measurement system for clinical gait analysis,* JB 1982

[5] Cappozzo, A., *Considerations on clinical gait evaluation,* JB 16 1983 , p.302

[6] Winter, D.A. *Pathologic gait diagnosis with computer-averaged electromygraphic profiles,* APMR vol 65, 1984, p.393-398

[7] Quanbury, O.A., *The clinical gait lab: form and function,* in Winter, D. A., Norman, R. W., Wells, R. P., Hayes, K. C., Patla, A. E., *Biomechanics IX-A,* Human Kinetics Press, Champaign, IL., 1985, p.509-512

[8] Krebs, D.E., Edelstein J.E., Fishman S., *Reliability of Observational Kinematic Gait Analysis,* FT 65, no.7, 1985, p.1027-1033

[9] Woltring, H.J., *Data Acquisition and Processing Systems in Functional Movement Analysis*

[10] An K.N, Jacobsen, M.C., Berglund, L.J. and Chao, E.Y.S., *Application of a magnetic tracking device to kinesiologic studies,* JB 21 1988, p.613-620

[11] Olsson, E., *Methods of Studying Gait,* in: G. Smidt (ed.), *Gait in Rehabilitation,* Churchill Livingstone, New York, 1990, p.21-41

[12] Eastlack, M.E., Arvidson J., Snyder-Mackler, L., Danoff, J.V., McGarvey, C.L., *Interrater reliability of videotaped observatonal gait-analysis assessements,* PT 71 no.6, 1991, p.465-472

[13] Whittle, M., *Gait Analysis: An Introduction*, Butterworth-Heinemann, Oxfort and Toronto 1991

[14] Harris, G.F., Wertsch, J.J., *Procedures for Gait Analysis*, APMR 75 1994, p.216-225

[15] Johanson, M.E., *Gait Laboratory: structure and data gathering*, in: Rose, J., Gamble, G., *Human Walking,* Williams & Wilkins, 2nd edition 1994, p.201-223

[16] Malouin F., *Observational gait analysis*, in : Craik, R.L., Oatis, C.A. (ed.), *Gait Analysis: Theory and Application*, Mosby, 1995, p.112-124

[17] Koff, D., *Joint kinematics: camera based systems,* in: Craik, R.L., Oatis, C.A. (ed.), *Gait Analysis: Theory and Application*, Mosby, 1995, p.183-204

[18] Cappello, A.,Cappozzo A., Leo, T., Paul, J.P., *3D Reconstruction of Human Motion, Theoretical and practical aspects,* Notes of a tutorial held at the XV Congress of the international society of Biomechanics, 1995

[19] Harlaar, J., Kleissen, R.F.M., Lankhorst, G.J., *Clinical assessment of movement disorders with kinesiological EMG visualized by multi-media-technology,* submitted for publication to Gait & Posture, 1997

Human Motion Analysis - Computer Vision - General

[20] Cédras, C., Shah, M., *A Survey of Motion Analysis from Moving Light Displays*, CVPR 1994, 214-221

[21] Young, *Handbook of Pattern Recognition and Image Processing: Computer Vision,* Chapter 12, Academic Press, 1994

Human Motion Analysis - Computer Vision - 2D

[22] Leung, M.K., Yang, Y.H., *A Region Based Approach for Human Body Motion Analysis,* PR 1987

[23] Leung, M.K., Yang, Y.H., *Human Body Segmentation in a Complex Scene*, PR 1987

[24] Ferrigno, G., Gussoni, M., *Procedure to automatically classify markers in biomechanical analysis of whole-body movement in different sport activities,* MBEC 1988

[25] O'Malley, Lynn, de Paor, *Kinematic analysis of human walking gait using digital image processing*, MBEC 1993

[26] Guo, Y., Xu, G., Tsuji, S., *Understanding Human Motion Patterns,* ICPR941994

[27] Guo, Y., Xu, G., Tsuji, S., *Tracking Human Body Motion Based on a Stick Figure Model,* JVCIR 1994

[28] Niyogi, S.A., Adelson, E.H., *Analyzing and Recognizing Walking Figures in XYT,* CVPR1994

[29] Leung, M.K., Yang, Y.H., *First Sight: A Human Body Outline Labeling System,* PAMI 1995

Human Motion Analysis - Computer Vision - 3D

[30] O'Rourke, J., Badler, N.I., *Model-based analysis of human motion using constraint propagation*, PAMI 1980

[31] Hogg, D., *Model based vison: A program to see a walking person*, IVC 1983

[32] Akita, K., *Image sequence analysis of real world human motion,* PR 17, 1984, p.73-83

[33] Chen, Z., Lee, H.J., *Determination of 3D body postures from a single view,* CVGIP 1985

[34] Chen, Z., Lee, H.J., *Knowledge-Guided Visual Perception of 3-D Human Gait from a Single Image Sequence,* SMC 1992

[35] Rohr, K., *Towards Model-Based Recognition of Human Movements in Image Sequences,* CVGIP IU 1994

[36] Gavrila, D.M., Davis, L.S., *3-D model-based tracking of human upper body movement: a multi-view approach*, Internet document, University of Maryland (http://www.umd.edu/)

[37] Rehg, J.M., Kanade, T., *DigitEyes: Vision-Based Human Hand Tracking*, Technical Report, Canegie Mellon University, CMU-CS-93220

[38] Kakadiaris, I.A., Metaxas, D., *3D Human Body Model Acquisition from Multiple Views,* Internet document

[39] Amaya, K., Hara, Y., Aoki, S., *Reconstruction of 3D movement using inverse analysis*, Proceedings of Graphics Interface/Vision Interface, 19-23 May 1997, Kelowna, British Columbia, Canada, 1997

Human Movement Synthesis

[40] Cutting, J.E., *A program to generate synthetic walkers as dynamic point-light displays,* BRMI 1978

[41] van Nieuwenhoven, M.S.E., *WALT, an interactive computer animation system,* Master's Thesis Eindhoven University of Technology, 1992

Feature tracking

[42] Rashid, R.F., *Towards a system for the interpretation of moving light displays*, PAMI 1980

[43] Sethi, I.K., Jain, R., *Finding Trajectories of Feature Points in a Monocular Image Sequence,* PAMI 1987

[44] Mori, S., and Doh, M., *A Sequential Tracking Extraction of Shape Features and its Constructive Description*, CGIP vol. 19, 1982, p.349-366.

[45] Muijtjens,, A.M.M., Roos, J.M.A., Prinzen, T.T., Hasman, A., Reneman, R.S. and Arts, T., *Noise reduction in estimating cardiac deformation from marker tracks*, Am J Physiol 1990, p.599-605.

[46] Muijtjens, A.M.M., Roos, J.M.A., Arts, T., Hasman, A., Reneman, R.S., *Extrapolation of incomplete marker tracks by lower rank approximation,* Int J Biomed Comput., no. 33, 1993, p.219-239.

[47] Lai, J.Z.C, *Tracking multiple features using relaxation*, PR, vol. 26, no. 12, 1993, p.1827-1837

[48] Sethi, I.K., *Tracking multiple features using relaxation - Comments*, PR vol. 27, no. 6, 1994, p.865.

[49] Krishnan, S., Raviv, D., *2D feature tracking algorithm for motion analysis*, PR vol. 28, no. 8, 1995, p.1103-1126

[50] Salari, V., Sethi, I.K., *Feature point correspondence in the presence of occlusion*, PAMI vol. 12, no. 1, 1990, p.87-91.

[51] Hwang, V.S.S., *Tracking feature points in time-varying images using an opportunistic selection approach*, PR vol. 22, no. 3, 1989, p.247-256.

[52] Zhang, Z.Y., *Token tracking in a cluttered scene, Image and Vision Computing*, vol. 12, no. 2, 1994, p.110-120.

[53] Yao, Y.S., Chellappa, R., *Tracking a dynamic set of feature points*, IEEE Transactions on Image Processing, vol. 4, no. 10, 1995, p.1382-1395.

[54] Muijtjens, A.M.M., Roos, J.M.A., Arts, T., Hasman, A., Reneman. R.,S., *Tracking markers with missing data by lower rank approximation*, JB 30, 1997, p.95-98.

Determining kinematics from marker positions

[55] Spoor, C., Veldpaus, F., *Rigid body motion calculated from spacial coordinates of markers,* JB 13 1980

[56] Miller, N.R., Shapiro, R., McLaughlin, T.M., *A technique for obtaining spatial kinematic parameters of segments of biomechanical systems from cinematographic data,* JB 13 1980, p.535-547

[57] Veldpaus, F.E., Woltring, H.J., Dortmans, M.G., *A least-squares algorithm for the equiform transformation from spatial marker co-ordinates,* JB 21 1988, p.45-54

[58] Söderkvist, I., Wedin, P.A., *Determining the movements of the skeleton using well configured markers,* JB 26 1993, p.1473-1477

[59] Chèze, L., Fregly, B. J., Dimnet, J., *A solidification procedure to facilitate kinematic analysis based on video system data,* JB 28 1995, p.879-884

Camera Calibration

[60] Wood, G.A., Marshall, R.N., *The accuracy of DLT extrapolation in three-dimensional film analysis.* JB 19 1986, p.781-785

[61] Tsai, R.Y. *An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision,* CVPR 1986, p.364-374

Software Engineering and Object Oriented Methods

[62] Meyer, B. *Object-oriented software construction*, Prentice Hall, 1988

[63] Rumbaugh, J., Blaha M., Premerlani W., Eddy F., Lorensen W., *Object-Oriented Modeling and Design,* Prentice-Hall Inc., 1991

[64] Jacobson, I., Christeron, M., Jonsson, P., Övergaard, G., *Object oriented software engineering - a use case driven approach.* Addison-Wesley, 1992

[65] Harmon, P. , *Objects In action: commercial applications of object-oriented technologies,* Addison-Wesley Publishers, 1993

[66] Rumbaugh, J., *Getting started, using use cases to capture requirements,* JOOP, sept. 1994, p. 8-23.

[67] Booch, G., *Object-Oriented Analysis And Design With Applications*, Benjamin Cummings, 1994

[68] Eliëns, A., *Principles of object-oriented software development,* Addison-Wesley Publishers, 1995

[69] Sommerville, I., *Software Engineering,* Addison-Wesley Publishers, 1996

[70] Rumbaugh, J., *To form a more perfect union, unifying the OMT and Booch methods,* JOOP, jan. 1996, p. 14-18.

[71] *Object-Oriented Languages in the Industry: A Comparison*, internet document from the Eiffel Web site, see http://www.eiffel.com

[72] *Object orientation FAQ, What Is OOA/OOD?*, FAQ of the comp.object newsgroup.

[73] *UML version 1.0*, internet document, see http://www.rational.com

Computer Vision, Image Processing and Computer Graphics

[74] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., *Computer graphics, principles and practice,* Addison-Wesley Publishing, 1990

[75] Jain, R., Katuri, R., Schunck, B.G., *Machine Vision*, McGraw-Hill, Inc., 1995

Scientific Visualization

[76] Yu, Chong Ho, *Visualization Techniques of Different Dimensions,* internet document. Arizona State University, (http://www.asu.edu)

[77] Bertin, J., *Graphics and graphic information-processing*, De Gruyter, 1981.

Mathematics

[78] Press, Teukolsky, Vettering, Flannery, *Numerical Recipes in C,* Cambridge University Press, 1994

Acknowledgments

I thank Jaap Harlaar for identifying the problem that kept me busy for more than three years, and for the pleasant cooperation during my time at the hospital. I also thank Kees van Overveld for his support and his many creative ideas that kept the project going.

I would like to thank all my colleagues at the department of Clinical Physics and Engineering, who provided a pleasant working atmosphere and who actively contributed to the project. In particular, Peter 'Sybar assistant' Tump and Rob Peters contributed to the software development. Hardware support was provided by the computer boys: Rob de Bree and Jan Rutger Kuiper. Ronald van Schijndel contributed a lot during brainstorm sessions. Even the mechanists Cork Klok, and Danny Koops contributed to the project by developing a calibration frame with great expertise.

I also thank Ronald Bonneveld and André Visser, two students from the Hogeschool Amsterdam, who participated via school projects. I also thank Isabelle Reymen, who gave some valuable comments on the manuscript in the area of design methodology.

I would also like to thank the management of the department, Albert-Jan Spruyt, Jan Arie Groot, and Michiel Sprenger for believing in the project and providing adequate resources. Furthermore, I would like to thank Onno van Roosmalen (TUE) and Rob Roelofs (VU Hospital) for making the project possible in the first place. In addition, I would like to thank the promotors Dieter Hammer and Klaas Kopinga and the other members of the committee.

Finally, I would like to thank my parents for their support during my entire education. My dad never pressured me into a certain direction or expected too much. However, he was always there whenever there were problems, in particular when they were of a mathematical nature. My mother made sure that I kept on going whenever I felt like quitting.

Samenvatting

Dit proefschrift beschrijft het Sybar-project. Sybar is een **Sy**steem voor **b**ewegings**a**nalyse bij **r**evalidatie. Het is ontwikkeld op het VU Ziekenhuis in Amsterdam, in het kader van een promotie op proefontwerp aan de Technische Universiteit Einhoven.

Menselijke bewegingen kunnen geanalyseerd worden met biomechanische meetsystemen. Hiervoor zijn in de loop der tijd een groot aantal verschillende methoden en technieken ontwikkeld. De meetsystemen kunnen belangrijke informatie geven over patiënten met bewegingsstoornissen. In de klinische praktijk wordt echter nog weinig gebruik gemaakt van de meetsystemen, omdat ze niet goed aansluiten op de werkwijze van de artsen.

Het doel van het Sybar-project is de ontwikkeling van een bewegingsanalysesysteem, specifiek voor de klinische praktijk. Net zoals in de klinische praktijk, wordt de menselijke observatie als uitgangspunt genomen. De bewegingen van de patiënt worden op video opgenomen. De gegevens van de meetsystemen worden niet apart gepresenteerd, maar geïntegreerd met de videobeelden. Deze combinatie van meetgegevens en videobeelden maakt het de artsen veel gemakkelijker de gegevens direct te relateren aan de bewegingen van de patiënt.

Sybar is ontwikkeld met object-georiënteerde methoden. De ontwerpmethode OMT (Object Modeling Technique) is als basis gebruikt. Interessante aspecten vanuit het oogpunt van softwareontwikkeling zijn:

- De specificatie van Sybar kan alleen bepaald worden met behulp van *rapid prototyping*.
- De omgeving waarin Sybar moet functioneren, is complex en dynamisch.
- De systeemeisen met betrekking to digitale video en beeldbewerking zijn op de grens van wat technisch mogelijk is.

Curriculum Vitae

Edwin Hubertus Hautus

ehautus@xs4all.nl

15 november 1969	Geboren te Eindhoven
1982 - 1988	Atheneum B, Strabrecht Colllege te Geldrop
1988 - 1992	Ingenieursopleiding, Technische Informatica, Technische Universiteit Eindhoven
1992 - 1994	Ontwerpersopleiding, Technische Informatica, Technische Universiteit Eindhoven
1995 - 1997	Promotie op proefontwerp, Technische Universiteit Eindhoven, uitgevoerd in het VU Ziekenhuis te Amsterdam
1997 -	Werkzaam bij Compuware's Uniface Lab te Amsterdam

Stellingen

Edwin Hautus

1. Bij het weergeven van biomechanische meetgegevens voor revalidatie-artsen is het essentieel dat deze gegevens geïntegreerd zijn met videobeelden.

   (dit proefschrift, paragraaf 2.4)

2. Over de betekenis van 'ontwerpen' zal binnen de informatica voorlopig nog geen overeenstemming worden bereikt.

3. De vele CAD (Computer Aided Design)-gereedschappen, die voor de verschillende disciplines beschikbaar zijn, moeten het in het begin van het ontwerpproces nog steeds afleggen tegen pen en papier.

4. Hoe meer informatie er tijdens het ontwerpen beschikbaar komt, hoe belangrijker het is eigenwijs te blijven.

5. Het kenmerk van een instabiel software-product is dat het zich op allerlei ongewenste momenten bijzonder stabiel gaat gedragen.

6. Sommige ontwerpers hebben de neiging om alles wat technisch mogelijk is ook toe te passen. Bij het ontwerp van webpagina's heeft dit ertoe geleid dat het web met elke nieuwe versie van de taal HTML minder gebruikersvriendelijk is geworden.

   (stellingen 2-6 zijn geïnspireerd door het werk aan dit proefontwerp)

7. Murphy's law lijdt aan het demo-effect: als men wil demonstreren dat iets vanwege Murphy's law fout gaat, zal het goed gaan.

8. De bewegingen die gepaard gaan met het salsa-dansen, kunnen zowel therapeutische waarde hebben, als tot verslaving leiden.

9. De snelste manier om op vrijwillige basis de rijkdom te herverdelen in een land waar iedereen even rijk is, is door middel van een piramidespel.

10. Mensen die zeggen 'laten we eerlijk zijn', zijn dat kennelijk gewoonlijk niet.

11. Het gebruik van ondertitels bij Nederlandstalige programma's is niet alleen gunstig voor doven en slechthorenden, maar ook voor hen die onder een aanvliegroute op Schiphol wonen.

12. De tijd die vroeger besteed werd aan de jacht naar voedsel, wordt tegenwoordig door veel mensen besteed aan het uitzoeken van wat de voordeligste spaarzegelactie is.