UNIVERSITY OF AALBORG

April 1998

Controlling a Movable Laser from a PC

Technical Report number R98-1002
by:

Thomas B. Moeslund
Mads Blidegn
Lau Bakman

Center for PersonKommunikation (CPK)
Institute of Electronic Systems
Fredrik Bajers Vej 7A – DK 9220 Aalborg Ø – Denmark
Tel.: +45 96 35 86 40 – Telefax +45 98 15 15 83

Preface

This tech-report is the documentation of a movable laser and the software which controls it. The laser is part of the CHAMELEON platform (see [1]) located at Center for PersonKommunikation (CPK), Aalborg University, Denmark.

The work presented here was done in the fall of 1997 as part of the multidisciplinary effort under the Multimedia and Multi-modal User Interfaces (MMUI) initiative, IntelliMedia 2000+.

Under this initiative the laser is used as an alternative output modality.

Currently it is used in two demonstrators developed by the work-team of MMUI; 'Campus Information System' and 'Virtual Airhockey Game', and a student project within the MMUI initiative.

Both the documentation and code can freely be used if a reference to this tech-report is clearly stated. The code and documentation files can be found in /usr/local/Laser/ on the PC, **R2D2**, located at CPK.

The physical setup of the laser and the low level control software is developed by Henrik Lausen from Laser Interface. Documentation about this stuff can be found in [2].

The authors would like to thank Bo Nygaard Bai for his help with specification of the laser.

Thomas B. Moeslund (tbm@vision.auc.dk)
Mads Blidegn (blidegn@kom.auc.dk)
Lau Bakman (bakman@kom.auc.dk)

ISSN 0908-1224

Contents

Chapter 1

Introduction

In every interface between a human and a machine, it should be possible for the machine to give feedback to the user. In normal interfaces images/graphics and sounds are used by the computer. When you want to design a new kind of interface, which is the case for the MMUI initiative, where the dependency between the human and the old modalities (keyboard, mouse and screen) should be broken, one needs to find a new way for the machine to give feedback to the user. Therefore it was chosen to use a laser.

A contact was made to the Danish company, Laser Interface. An agreement was set up where the company delivered a laser system (laser, lenses, mirrors, controllers etc.) and the low level software to control this laser system.

The work group within the MMUI initiative should then develop a high level driver, which made it possible to control the laser from another computer.

The laser is together with a speech synthesizer, used by the projects within the MMUI initiative to give responses to the user. The laser can generate a dot which can be used to point with, it can draw shapes e.g. circles to indicate an area, it can draw a path from one location to another e.g. in a map application etc.

It is all together a very nice output modality when designing multi modal systems.

In the rest of this report the laser system designed by Laser Interface and the high level software driver to control it is presented.

Chapter 2

Controlling the Laser

The laser is being controlled from a standard 486 DOS PC, which only task is to control the laser. The DOS PC is therefore controlled by another computer, where the actually program, which needs the laser, is running. The laser and the controlling PC are not designed to give feedback, making the interaction a one way communication.

This master/slave structure can be seen in figure 2.1 where the arrows indicate one way communication. For further details see [1].

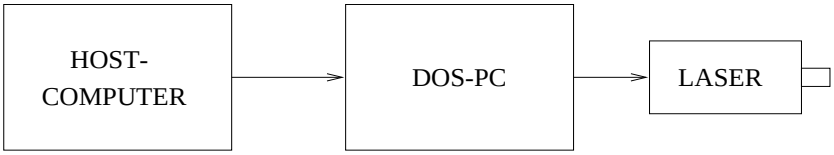

Figure 2.1: The master/slave control of the laser.

Since the laser is used to draw paths and dots at different locations, it must be movable. This is obtained by moving two small mirrors which reflect the laser beam. In figure 2.2 a schematic representation of the laser is shown.

The laser beam is generated in the laser diode and the more current this diode receives the more power the laser beam will have. This is known as the modulation of the laser. The diode is a red (640nm) laser capable of generating 15mW at 100% modulation. It is small (1cm) and cheap (500-1000$) but not very focused. Therefore the lenses are added to compensate for this. After the beam has been focused it hits the two small mirrors (5x10mm) which position determines where the laser beam ends up.

The control of the modulation and the control of the mirrors are extremely

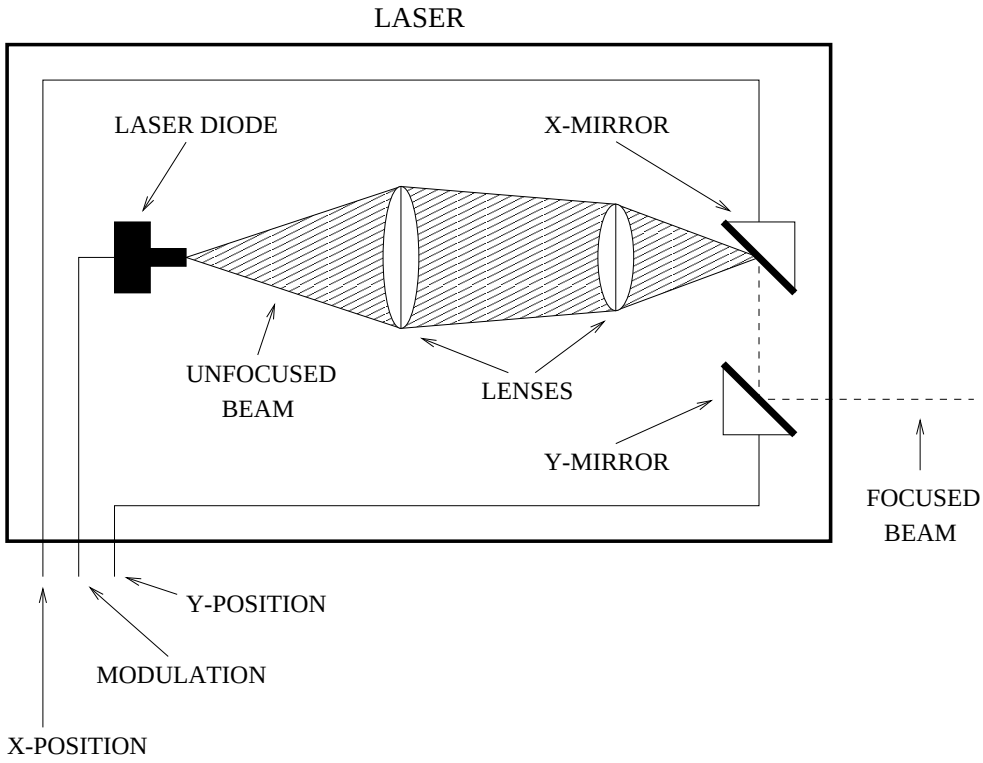

Figure 2.2: The internal structure of the laser.

fast. You can scan 600 points at 50 Hz, which corresponds to drawing a path with 100 corners and updating it without the human eye noticing it.

However, the more points the lesser light per point, meaning that not too many points should be present at the same time since this will make it harder to see the laser beam. To solve this problem you can not just buy a more powerful laser, because the power spectrum of the laser moves into the inferred area as the power increases. This can be solved by buying a green laser, but then the price increases with a factor of 10.

Another way of increasing the visibility of the laser is to lower the intensity of the surrounding light. If you do so, you should be aware of the effect this can have on the image quality, if a camera is involved.

Chapter 3

Calibrating the Laser

Before the laser can be used by any system, it must be calibrated.

The direction of the beam, which is send from the laser, corresponds to the angles of the two mirrors. There is a linear dependency between the different angles, meaning that the output will correspond to Cartesian coordinates if the laser is projected onto the inside of a sphere. This is usually not the case since the laser normally is projected on to a flat table. Therefore the coordinates used by the laser will be distorted (depending on the distance to the focus point on the table) with respect to a standard coordinate system and some kind of correction algorithm must be made.

How this correction should look like depends on the application. When the laser is projected onto a flat table (perpendicular to the laser) the correction can be done using inverse circle arcs.

The correction algorithm transform the laser's coordinate system, into a standard Cartesian system which can be calibrated by linear transformation.

Chapter 4

The Serial Interface

The software, which is used to control the laser from the DOS PC is, developed by Laser Interface, [1]. The software to control the DOS PC from another computer is called the serial interface and will be described in the following.

4.1 Introduction

The serial interface is used to remotely control the laser from a host computer. The functionality of the serial interface is to send commands and coordinates for controlling the laser. The commands which can be send to the laser controller are equivalent to the commands, which can be issued directly from the user interface on the DOS PC [1]. The communication between the serial interface and the laser controller is only one way so it is only possible to send information to the laser controller. A consequence of this is, that the serial interface has to keep track of what is sent to the laser controller.

4.2 Layers

The serial interface consists of three layers of functionality; a C-API providing functions for controlling the laser, an error check and mirror layer which makes sure that coordinates sent to the laser are within the allowed range, and finally a communication layer which takes care of actually sending information to the laser controller. The different layers of the serial interface

are shown on figure 4.1.

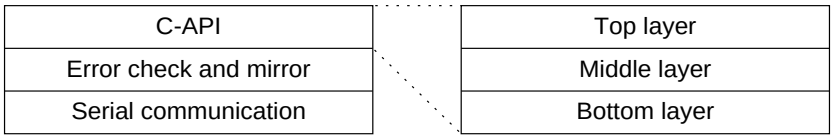

Figure 4.1: The Layers of Serial Interface and the C-API.

4.3 Functionality

The serial interface is implemented as a shared library programmed in C. The library is divided into three levels of abstraction as seen on figure 4.1:

Top level: The functions at the top level have the highest level of abstraction. When using these functions the caller does not need to produce frames beforehand. The functions in this level uses a font library to send text and symbols (markings, pointers, arrows, etc.) to the laser. Currently the only function in this level is "Laser_send_text".

Middle level: These functions are on a lower abstraction level as the caller should create frames before calling the functions. Currently two functions are present in this level: "Laser_send_frame" and "Laser_render_te xt". The "Laser_send_frame" function transmits the frame making sure that the right commands are sent before and after transmitting the coordinates. The "Laser_render_text" function adds text from the font library to a frame but does not send it.

Bottom level: These functions map almost directly to the protocol commands, refer to the laser manual. This is the lowest level of abstraction as the caller takes care of everything from sending the points to saving and showing the frame. This level also contains functions for making flow sequences (animation) and moving existing frames.

Chapter 5

The Protocol

The protocol used between the host computer and the DOS PC contains the commands shown in figure 5.1. In the following the individual commands is described.

NEW DRAW The laser is set in draw mode and is ready to receive points to be drawn. The drawing is not shown while the points are transmitted.

NEW DRAW SHOW The laser is set in draw mode and is ready to receive points to be drawn. The drawing is shown while the points are transmitted.

Pxxxx,yyyy A new point. X=xxxx, Y=yyyy. This point is shown. X=[0;4095], Y=[0;4095].

Bxxxx,yyyy A new point. X=xxxx, Y=yyyy. This point is not shown. X=[0;4095], Y=[0;4095].

Qxxyy A new Point. X=xx, Y=yy. This point is shown. This command uses base 64. X=[00;oo], Y=[00;oo].

Cxxyy A new Point. X=xx, Y=yy. This point is not shown. This command uses base 64. X=[00;oo], Y=[00;oo].

SAVE FRAME ff Saves the current frame as frame number ff. The current drawing is finished. ff=[0;100].

SHOW FRAME ff Shows frame number ff. ff=[0;100].

OFFSET xxxx,yyyy Moves the current frame to position X=xxxx, Y=yyyy. X=[0;4095], Y=[0;4095].

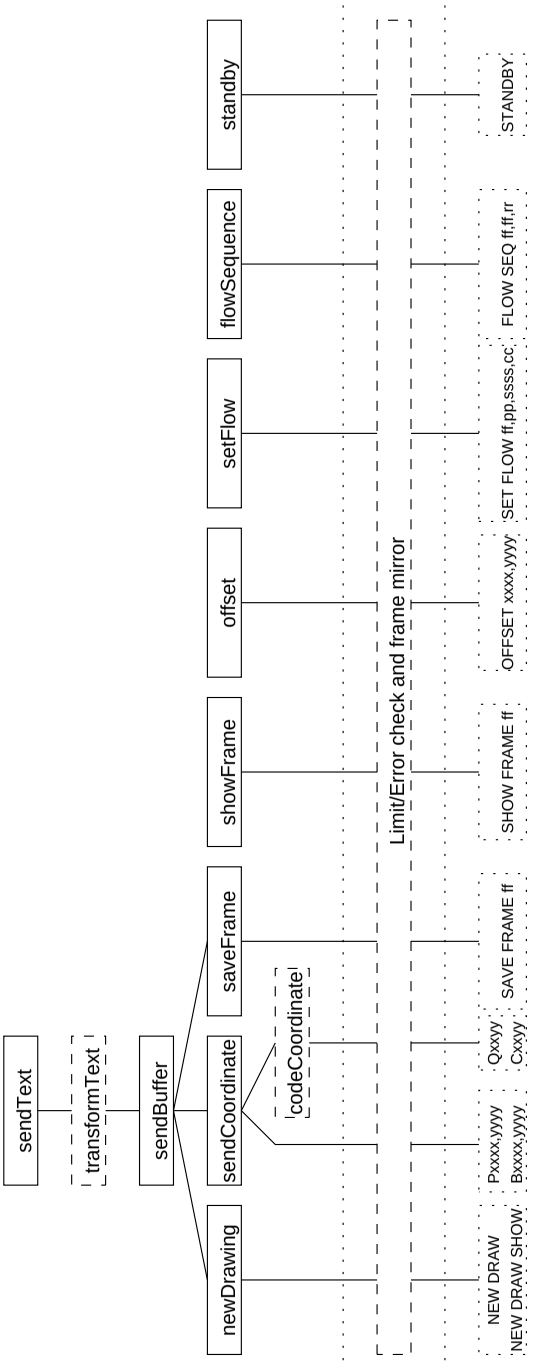

Figure 5.1: Serial protocol.

WRITE FILE ff Saves the frames in the buffers to a file. This does not delete the frames from the buffer. ff=[0;10].

READ FILE ff Reads a file of frames and puts them into the buffer on the DOS PC. ff=[0;10].

SET FLOW ff,pp,sssss,cc Sets a flow process for a frame. ff: frame number [0;100], pp: process number [1;16], sssss: flow steps or speed [1;65535] and cc: process curve [1;6].

FLOW SEQ f1,f2,rr Starts a flow sequence where all frames from f1 to f2 are shown in a sequence. rr states the number of times the sequence should be repeated. f1=[0;100], f2=[0;100], rr=[0;65535]. rr can be excluded and the sequence is only shown ones.

STANDBY Sets the laser in standby mode. This means that the scanner is set in center position and the modulation of the beam is set to 0.

Chapter 6

Technical Documentation

In this chapter the available functions and the installation of these are described.

6.1 Installation

Before the functions can be used they must be installed. This is done by uncompress the file Laser.tgz (tar xvzf Laser.tgz) yielding the following files:

liblaser.so - The C-library containing the methods for communicating with the Laser.

laser.h - Header file describing available functions, must be included when using the library.

font.h - Header file included by laser.h

default.font - Font definition file describing the ordinary English alphabet.

symbol.font - Font definition file describing special symbols like a cross or the CPK-logo etc.

main.c - Example source code for demonstrating the functionality of the Laser.

run - Script used to set the appropriate environment variables and execute the Laser software.

Makefile - Makefile for compiling the example source code.

Edit main.c and redefine the COM_PORT variable to the used com port.

The software is now ready for use. Compile the example software using make and execute the run script to try out the program.

6.2 Requirements

In order to use the Laser library the following is required:

- Linux PC, with serial port
- DOS PC connected to the Laser
- The GCC-compiler and the Make-utility
- Serial cable connecting the Linux PC and the DOS PC
- The LaserXi software running on the DOS PC

6.3 Specification of Available Functions

All functions return 0 for success and -1 for failure unless otherwise specified. In case of failures two global variables are set: an integer giving an error code and a string giving a description of the error.

6.3.1 Point and Frame Definition

When sending coordinate information to the laser it can be done either as single points or as frames. A point is a struct consisting of the following three integers:

```
- struct point {
-      int x; /* The x coordinate of the point */
-      int y; /* The y coordinate of the point */
-      int show; /* States whether the line ending in this point is shown or
not */
- };
```

A frame consist of a number of points describing an image to be drawn using the laser. More specifically a frame is an array of points.

Laser Laser_init(const char *dev, int base)

Must be called before any communication with the laser is possible. The function takes care of establishing the connection with the DOS PC and perform some initialization. Returns a struct which is used as an argument for all Laser functions.

Arguments:

***dev** - The serial port to open.
base - The base of the coordinates, either 10 or 64.

int Laser_close(Laser l)

Should be called before terminating the program as it takes care of gracefully closing the serial connection.

Arguments:

l - The struct belonging to the laser class.

Font Font_init(char *path)

Reads a font definition file and returns a Font struct representing the font. The function can be used if symbols different from the default font is wanted.

Arguments:

***path** - The path and filename of the font file.

int Laser_send_text(Laser l, char *text, Font font, point tOffset, point lOffset, float xscale, float yscale, int *points, int frameNr, int show)

Converts a text string into a frame using the specified font. If NULL is supplied instead of a Font struct the default font is used. After converting the text, the frame is sent ti the laser.

Arguments:

l - The Laser struct returned by Laser_init.

***text** - The text which is sent to the laser.

font - A Font struct representing the font which should be used to convert the text into a frame. If NULL is supplied instead of a Font struct the default font is used.

tOffset - TextOffset: An offset coordinate which is added to all points in the frame sent to the laser.

lOffset - LetterOffset: Determines the distance between the letters and the direction of the letters. See section 6.3.2.

xscale - Scale of the letter in the x direction.

yscale - Scale in the letter in the y direction.

***points** - This variable must be set to zero before calling this function. The value of the variable after the call to the function is the number points sent to the laser.

frameNr - The frame number which the frame should be saved as [0;99].

show - An integer stating whether the frame should be shown while being drawn (L_FALSE=not shown, L_TRUE=shown).

int Laser_send_frame(Laser l, point *frame, point offset, int frameNr, int *points, int show)

Sends a frame of points to the laser. This function should be used whenever sending points to the laser. Before calling the function an array of points should be constructed describing the desired drawing.

Arguments:

l - The Laser struct returned by Laser_init.

***frame** - An array of points to be sent to the laser.

offset - An offset coordinate which is added to all points in the frame before sending the frame.

frameNr - The frame number which the frame should be saved as [0;99].

***points** - The number of points in the frame to be sent. The value of the variable after the call to the function is the number points sent to the laser.

show - An integer stating whether the frame should be shown while being drawn (L_FALSE=not shown, L_TRUE=shown).

point *Laser_render_text(Laser l, char *text, Font font, point *frame, point tOffset, point lOffset, float xscale, float yscale, int *points)

Adds text to the given frame at a given offset.

Arguments:

l - The Laser struct returned by Laser_init.

***text** - The text string to be drawn by the laser.

font - A Font struct representing the font which should be used to convert the text into a frame. If NULL is supplied instead of a Font struct the default font is used.

***frame** - The frame to which the text should be added. The frame does not necessarily have to contain anything. But be sure that the points parameter is zero if the frame is empty.

tOffset - TextOffset: An offset coordinate which is added to all points in the frame sent to the laser.

lOffset - LetterOffset: Determines the distance between the letters and the direction of the letters. See section 6.3.2.

xscale - Scale of the letter in the x direction.

yscale - Scale in the letter in the y direction.

***points** - The number of points in the frame to be sent. This value should be zero if the frame is empty. The value of the variable after the call to the function is the number points sent to the laser.

frameNr - The frame number which the frame should be saved as [0;99].

show - An integer stating whether the frame should be shown while being drawn (L_FALSE=not shown, L_TRUE=shown).

int Laser_new_drawing(Laser l, int show)

Initiates a new drawing. The drawing can either be shown while being created or not. The function is only needed when single points are to be send.

Arguments:

l - The Laser struct returned by Laser_init.

show - An integer stating whether the frame should be shown while being drawn (L_FALSE=not shown, L_TRUE=shown).

int Laser_send_point(Laser l, point p)

Sends a single point to the laser. Be aware that the Laser_new_drawing function must be called before sending the first point.

Arguments:

l - The Laser struct returned by Laser_init.
point - The point to be send to the laser.

int Laser_save_frame(Laser l, int frameNr)

Saves a frame as the specified frame number.

Arguments:

l - The Laser struct returned by Laser_init.
frameNr - The frame number which the frame should be saved as [0;99].

int Laser_show_frame(Laser l, int frameNr)

Instructs the laser to show the specified frame.

Arguments:

l - The Laser struct returned by Laser_init.
frameNr - The frame number to be shown by the laser [0;99].

int Laser_offset(Laser l, point offset)

Moves all frames on the DOS PC according to the given offset. This operation does not change the frames stored on the DOS PC but adds the offset to the frame right before it is drawn. As this operation affects all frames it is important to reset the offset again before showing new frames.

Arguments:

l - The Laser struct returned by Laser_init.
offset - The offset which the frame should be moved by i.e. the offset is relative. The x and y coordinates of the offset are positive values [0;4095].

int Laser_set_flow(Laser l, int frameNr, int process, int flowSteps, int processCurve)

Defines a flow for the specified frame. Please refer to the laser manual for an explanation of the different flows.

Arguments:

l - The Laser struct returned by Laser_init.
frameNr - The frame number which the flow should affect [0;99].
process - Process type e.g. zoom in, rotate [1;16].
flowSteps - Speed of the flow process [1;65535].
processCurve - The curve which the process is to follow e.g. linear, exponential etc. [1;6].

int Laser_flow_seq(Laser l, int startFrame, int endFrame, int repeat)

Defines and executes a flow sequence containing one or more frames. The flow used for each frame is defined by the Laser_set_flow function.

Arguments:

l - The Laser struct returned by Laser_init.
startFrame - The first frame in the flow sequence [0;100].
endFrame - The last frame in the flow sequence [0;100].
repeat - The number of times to repeat the flow sequence, 0 = infinite [0;65535].

int Laser_standby(Laser l)

Centers the scanner and "shuts of" of the laser i.e. the modulation of the laser is set to zero.

Arguments:

l - The Laser struct returned by Laser_init.

int Laser_save_file(Laser l, int filename)

Saves the array of frames which is stored on the DOS PC to a file.

Arguments:
l - The Laser struct returned by Laser_init.
filename - The name of the file to be saved [0;99].

int Laser_read_file(Laser l, int filename)

Reads a file containing frames and places them in an array on the DOS PC.

Arguments:
l - The Laser struct returned by Laser_init.
filename - The name of the file to be read [0;99].

int Laser_sync(Laser l)

Clears the array of frames on the DOS PC and sends over the mirror of the frames stored on the Linux PC.

Arguments:
l - The Laser struct returned by Laser_init.

int Laser_clear(Laser l)

Clears the array of frames on the DOS PC.

Arguments:
l - The Laser struct returned by Laser_init.

6.3.2 Letter Offset Description

The letter offset is defined as the point struct described in section 6.3.1, where only the x and y values are used.

The different values of x and y:

- When both x and y is 0: The letters of the text is written in a 45 degree down angle.

- When both x and y is grater than 0: The letters of the text is written in a down angle determined by the values of x and y.
- When x is 0 and y is greater that 0: Vertical oriented text where the distance between the letters is determined by y.
- When x is greater that 0 and y is 0: Horizontal oriented text where the distance between the letters is determined by x.

Chapter 7

Conclusion

In this tech report the controllable laser at CPK, Aalborg University, Denmark is together with the high level driver to control it is, presented.

The laser is controlled from a DOS PC which is again controlled from a host computer. The communication between the DOS-PS and the laser is developed by Henrik Lausen, Laser Interface [2] while the communication between the host computer and the DOS PC is described in this report. The communication is done through a three layer structure which consist of; a C-API, an error check and mirror layer, and finally the serial communication.

The C-API again consist of three layers each offering C-functions to the user at different abstraction levels.

All the software needed to control the laser is located in /usr/local/Laser/ on a Linux PC at CPK named **R2D2**.

References

[1]: "A platform for developing Intelligent MultiMedia applications". T. Brøndsted, P. Dalsgaard, L.B. Larsen, M.J. Manthey, P. Mc Kevitt, T.B. Moeslund and K.G. Olesen. Technical Report. CPK, Aalborg University, Denmark. 1998.

[2]: "LaserXI". Henrik Lausen, Laser Interface, Center for Advanced Technology, Denmark. 1997.