LCD-Terminal

Manual

LCD-Terminal: Manual

This document was generated on 2017-09-22T11:34:43+02:00.

Table of Contents

List of Figures

List of Tables

List of Examples

1. Overview

The LCD-Term is a ready to use black-white display including VT100 support, three built-in fonts, matrix keyboard support, simple IO functionality and graphic routines such as drawing primitives, GIF images as well as blitting from and to multiple back-buffers. It is available in four physical display sizes from 1.2" to 3.5".

Any headless hardware that shall be extended by some output capabilities or full user interaction can be connected to LCD-Term. Requirements are kept low due to the utilisation of standard peripherals like USB or a serial port which should be readily available on many platforms.

The design is made to be used in an easy manner: Four wires are sufficient to provide full display functionality, all further IO pins can be configured to be used as matrix keyboard or IO pin. Configuration is done via a built-in setup shell accessible through a simple terminal program. There is no need for any configuration software running on a host PC. Thus testing, configuration and development is independent from any specific OS platform.

Adding LCD-Term support to existing software projects is simple, too. There is no library that has to be used to interact with the firmware; all commands are text messages to be sent directly via the serial connection. Developers are not limited to any specific programming language - simple string manipulation and access to the serial port or USB is all you need. Therefore, most of the samples are either messages typed into a serial terminal or short LUA code[1].

[1] LUA is a small and simple scripting language. See: http://www.lua.org

2. Commissioning

Power is either be supplied via a 5V USB unit which is connected to micro USB port or via the side connectors using the 3.3V or 5V pin if available.

Figure 2.1. Pin-out LCDTerm12

Figure 2.2. Pin-out LCDTerm15

Figure 2.3. Pin-out LCDTerm25

Figure 2.4. Pin-out LCDTerm35

2.1. USB

When plugging a USB cable using the micro USB port X3, the device is enumerated as USB CDC device. Modern operating systems such as Windows 10, MAC OS X or Linux come with appropriate drivers pre-installed.

After initialisation, a new communication device is added to the system. On Windows this is called `//./COMx` where *x* is a decimal number. Linux and UNIX-like systems in general use either `/dev/ttyACMx` or `/dev/ttyUSBx` as naming scheme.

Simple text output can be done from almost every shell in a similar way.

```
echo "Hello world !" > //./COM5
```

Example 2.1. Hello world on a Windows shell

2.2. UART

The UART interface is designed to be used in embedded circuits keeping costs as low as possible. According to this, no RS232-LVTTL level shifters are used. Most microprocessors/microcontrollers used in such an environment can be connected directly to the LCD-Term.

Caution

Using the UART as a connection to a common PC requires RS232-LVTTL level shifters. As an alternative, LVTTL to USB converters such as an FTDI cable or chip can be used. This solution is already covered by LCD-Term's USB connector. However, USB driver support can be easier on older OS when using FTDI chips/cables.

2.3. Configuration

LCD-Term offers a built-in setup dialog for configuration. It can be entered by either short circuiting GND and PA00 during power-up or by an escape sequence typed into a serial terminal program. The sequence is `ESC[Q`. See Table 3.4, “Additional functions” for details.

The setup dialog can be attended via a terminal program. You are free to use USB or the UART connection. Follow the menue to configure backlight, display contrast and IO pins. Don't forget to save before exiting the setup to make your changes persistent. By delivery, LCD-Term is configured with reasonable values.

LCD-Term is capable of displaying GIF images from its internal Flash memory. At the moment this documentation is written, uploading images requires a third party tool from the manufacturer of the microcontroller that is used. Since we are going to simplify this procedure in the future, it is not covered here. Please see: *http://www.taskit.de/LCD-Term_en.html*

3. Instruction set

All instructions follow a basic scheme compatible with ANSI escape sequences. A sequence starts with an introductory character (hexcode 0x1B), in the following written as ESC, and is completed by either a upper- or lowercase basic latin letter (A-Z, a-z) representing a function code. Arguments to these functions are written as decimal integer numbers, separated by semi-colon, between ESC and the function code.

All printable ASCII codes that are not part of an active escape sequence are shown on the display. The text cursor advances automatically.

3.1. Text functions

Text functions are used to control the output position of characters to be printed, erase the display or parts of it, scroll the display's content and toggle different modes of operation such as inverse presentation. Here is a list of all textual functions.

Command	Description	Example	LUA string
Text cursor functions			
A	Move text cursor *n* lines up. If *n* is omitted, one line is moved up.	`ESC[A ESC[2A`	`"\27[2A"`
B	Move text cursor *n* lines down. If *n* is omitted, one line is moved down.	`ESC[B ESC[2B`	`"\27[2B"`
C	Move text cursor *n* columns right. If *n* is omitted, one column is moved right.	`ESC[C ESC[12C`	`"\27[12C"`
D	Move text cursor *n* columns left. If *n* is omitted, one column is moved left.	`ESC[D ESC[16D`	`"\27[16D"`
a	Set text cursor to line *n*. If *n* is omitted, cursor is set to the first line.	`ESC[a ESC[2a`	`"\27[2a"`
c	Set text cursor to column *n*. If *n* is omitted, cursor is set to the first column.	`ESC[c ESC[5c`	`"\27[5c"`
H, f	Set cursor to position [*y*; *x*]. If parameters are omitted, cursor is set to [1; 1].	`ESC[H ESC[2;12f`	`"\27[2;12H"`
s	Save current cursor position.	`ESC[s`	`"\27[s"`
u	Restore previously saved cursor position	`ESC[u`	`"\27[u"`
Mode settings			
z	Set fontsize *n*. Available sizes are: 0(8x6; default), 1(8x8), 2(16x8). If *n* is omitted, font is set to 8x6. Any changes of fontsize set curser to position [1; 1]	`ESC[z ESC[2z`	`"\27[2z"`
l	Disable mode *n*. Supported modes are: 1, 2, 3, 8, 25, 75. See Table 3.2, “Options” for details.	`ESC[?2l ESC[?25l`	`"\27[?2l"`
h	Enable mode *n*. Supported modes are: 1, 2, 3, 8, 25, 75. See Table 3.2, “Options” for details.	`ESC[?2h ESC[?25h`	`"\27[?2h"`

Command	Description	Example	LUA string
Display clear functions			
J	Erase display content according to option *n*. Options are: 0 = from cursor to end, 1 = from start to cursor, 2 = full display. Omitting *n* erases the whole display.	`ESC[2J ESC[J`	`"\27[2J"`
K	Erase line content according to option *n*. Options are: 0 = from cursor to line end, 1 = from line start to cursor, 2 = full line. Omitting *n* erases the current line from cursor to end.	`ESC[2K ESC[K`	`"\27[2K"`
P	Delete *n* characters right to the cursor. The character at the cursor position is also cleared. If *n* is omitted, one character is erased.	`ESC[P ESC[4P`	`"\27[4P"`
M	Delete *n* characters left to the cursor. The character at the cursor position is also cleared. If *n* is omitted, one character is erased.	`ESC[M ESC[4M`	`"\27[4M"`
Scroll functions			
S	Scroll display content *n* text lines up. If *n* is omitted, one line is scrolled.	`ESC[S ESC[1S`	`"\27[2S"`
T	Scroll display content *n* text lines down. If *n* is omitted, one line is scrolled.	`ESC[T ESC[1T`	`"\27[2T"`

Table 3.1. Text functions

Commands *l* and *h* are used to dis/enable several options that influence the textual representation. The description of these options are collected in Table 3.2, “Options”.

Option	***h*: enable**	***l*: disable**
1	If cursor presentation is also enabled, the cursor is drawn as a large block.	If cursor presentation is also enabled, the cursor is drawn as a thin line.
2	Cursor presentation is turned on.	Cursor presentation is turned off.
25	Cursor blink is switched on.	Cursor blink is switched off.
3	Text is drawn in inverse mode. Background is dark.	Text is drawn in normal mode. Background is clear.
8	Automatic wrap on end of line is on. Display will scroll when last line is wrapped.	Automatic wrap at end of line is off.

Table 3.2. Options

This LUA sample code displays some text and the current time and date on top of the screen in two different modes.

```
-- All commands will be sent using standard file IO
-- on the USB CDC port my LCD-Term is connected to.
-- On Linux it will be either "/dev/ttyACMx" or "/dev/ttyUSBx";
-- on Windows the device is "//./COMx".
f = io.open("/dev/ttyACM0", "wb")

function wait(seconds)
 local start = os.clock()
 repeat until os.clock() > start + seconds
end

function clear_display(f)
 f:write("\27[2J")
 f:flush()
 wait(0.01)
end

function clear_line(f)
 f:write("\27[2K")
 f:flush()
 wait(0.01)
end

function gotoYX(f, y, x)
 -- insert y and x at their appropriate positions
 -- format: ESC[y;xH
 f:write(string.format("\27[%d;%dH", y, x))
 f:flush()
 wait(0.01)
end

-- disable inverse mode
f:write("\27[?3l")
-- enable automatic line wrap
f:write("\27[?8h")

clear_display(f)

gotoYX(f, 4, 1)
f:write("\nSome text here...\nand here...\n\nand there.")

-- enable inverse mode
f:write("\27[?3h")
-- disable automatic line wrap
f:write("\27[?8l")

-- clear first three lines
for y = 1, 3, 1 do
 gotoYX(f, y, 1)
 clear_line(f)
end

date_format = "%H:%M:%S %Y/%m/%d"

while true do
 -- output current time
 gotoYX(f, 2, 2)
 f:write(os.date(date_format))
 f:flush()
end

-- never reach this because of loop for infinity
f:close()
```

Example 3.1. example1.lua

3.2. Graphic functions

Besides textual functionality, LCD-Term offers a set of sequences that can be used for drawing. Simple forms such as lines, rectangles and circles can be filled with different patterns, text can be positioned on a per pixel basis instead of a per character one, a screen can be prepared in a back buffer page and being blit to the front buffer. Icons and special symbols are supported through the GIF feature.

Most of the functions are not shown on screen directly but rendered into one of the two back buffers. This allows to set a screen up without bothering the user with multiple updates of the same region (e.g. when drawing something on top of an image). There is a function controlling and forcing the screen update which can be called after rendering the back buffer is finished.

Command	Description	Example	LUA string
Graphic primitives			
x	Sets or clears a pixel at position [*y*; *x*]. The third parameter *c* decides wheter the pixel is set (*c* > 0) or cleared (*c* = 0).	ESC[*y*;*x*;*c*x ESC[10;40;1x	"\27[10;20;1x"
y	Draw a line from [*y0*; *x0*] to [*y1*; *x1*]. The last parameter *p* specifies a repeating pattern where each bit corresponds with a pixel. A set bit results in a set pixel.	ESC[*y0*;*x0*;*y1*;*x1*;*p*y ESC[0;0;63;127;255y	"\27[63;0;0;127;85y"
v	Draw a rectangle from [*y0*; *x0*] to [*y1*; *x1*]. The last parameter *p* specifies a repeating pattern to be used for the rectangle's outlines.	ESC[*y0*;*x0*;*y1*;*x1*;*p*v ESC[0;0;63;127;15v	"\27[63;0;0; 127;111v"
w	Fill a rectangle from [*y0*; *x0*] to [*y1*; *x1*]. The last parameter *p* (0-15) is the number of one of sixteen fill pattern.	ESC[*y0*;*x0*;*y1*;*x1*;*p*w ESC[0;0;63;127;7w	"\27[62;1;1;126;3w"
k	Draw/fill a circle with center [*y*; *x*] and radius *r*. Parameter *fill* selects wheter *p* is a fill (*fill* = 1) or an outline pattern.	ESC[*y*;*x*;*r*;*fill*;*p*k ESC[32;64;20;1;7k	"\27[32;64;30;0; 111k"
Graphic cursor			
G	Set graphic cursor to pixel coordinates [*y*; *x*]. Text functions as well as printing strings work relative to this position.	ESC[*y*;*x*G ESC[10;30G	"\27[57;100G"
GIF support			
Y	Display GIF *num* at position [*x*; *y*]. Parameter *sub-image* selects the sub-image to be shown in multi image GIFs. Only *num* is mandatory; omitted parameters default to 0.	ESC[*num*;*x*;*y*;*sub-image*Y ESC[1;32;16Y	"\27[3;16;16;4Y"
Buffers and blitting			
V	Switch to buffer *num*. All further drawing is done in the selected buffer.	ESC[*num*V ESC[0V	"\27[1V"

Command	Description	Example	LUA string
U	Update screen: Depending on *mode* the function automatic update is disabled (*mode* = 0) or enabled (*mode* = 1). When *mode* is equal to 2 or omitted, the currently selected buffer is copied onto screen.	ESC[*mode*U ESC[0U ESC[1U ESC[2U	"\27[U"
W	Blit source rectangle at [*x0*; *y0*] with size [*dx*; *dy*] to [*x1*; *y1*]. Source buffer is *src*, target buffer is *dst*.	ESC[*src*;*x0*;*y0*;*dx*;*dy*;*dst*;*x1*;*y1*W ESC[1;0;0;32;32;0;10;15W	"\27[1;10;20;16;16;0;0;0W"

Table 3.3. Graphic functions

Here is a more complex example using graphic functions to draw an animated random graph.

```
-- All commands will be sent using standard file IO
-- on the USB CDC port my LCD-Term is connected to.
-- On Linux it will be either "/dev/ttyACMx" or "/dev/ttyUSBx";
-- on Windows the device is "//./COMx".
f = io.open("/dev/ttyACM0", "wb")

function wait(seconds)
 local start = os.clock()
 repeat until os.clock() > start + seconds
end

math.randomseed(os.time())

graph_x = 10
graph_y = 10
graph_dx = 128 - 2 * graph_x
graph_dy = 64 - 2 * graph_y
variance = 5
steps = 4
pattern = 0xFF
values = {}
values[0] = math.random() * graph_dy

function calc_value(old_y)
 y = old_y + (math.random() - 0.5) * variance, graph_dy
 -- shrink value to graph size
 y = math.max(math.min(y, graph_dy - 1), 0)

 return y
end

function calc_graph(start_x, end_x)
 for x = start_x + 1, end_x, 1 do
  -- round value to the nearest integer
  y = math.floor(calc_value(values[x - 1]) + 0.5)
  values[x] = y
  old_y = y
 end
end

function draw_graph(f, start_x, end_x)
 for x = start_x, end_x - 1, 1 do
  -- draw a line between the two points
  f:write(string.format("\27[%d;%d;%d;%d;%dy",
                        graph_y + values[x], graph_x + x,
                        graph_y + values[x + 1], graph_x + x + 1,
                        pattern))
  f:flush()
  -- here we don't use XON/XOFF, so make a little pause after
  -- having drawn some lines
```

```
  if (x % 8) == 0 then
   wait(0.01)
  end
 end
end

-- draw frame
f:write(string.format("\27[%d;%d;%d;%d;%dw", 0, 0, 63, 127, 4))
f:flush()
if (graph_y > 4) and (graph_x > 4) then
 f:write(string.format("\27[%d;%d;%d;%d;%dw",
                       graph_y - 4, graph_x - 4,
                       graph_y + graph_dy + 3,
                       graph_x + graph_dx + 3, 0))
 f:flush()
 f:write(string.format("\27[%d;%d;%d;%d;%dv",
                       graph_y - 2, graph_x - 2,
                       graph_y + graph_dy + 1,
                       graph_x + graph_dx + 1, 0x33))
 f:flush()
end
wait(0.01)

calc_graph(0, graph_dx - 1)

while true do
 -- clear graph using filled rectange
 f:write(string.format("\27[%d;%d;%d;%d;%dw",
                       graph_y, graph_x,
                       graph_y + graph_dy - 1,
                       graph_x + graph_dx - 1, 0))
 f:flush()
 wait(0.05)

 -- shift graph left, add new values
 for x = 0, graph_dx - 1 - steps, 1 do
  values[x] = values[x + steps]
 end
 calc_graph(graph_dx - 1 - steps, graph_dx - 1)

 -- draw graph
 draw_graph(f, 0, graph_dx - 1)
 f:flush()
 wait(0.05)

 -- update screen
 f:write("\27[U");
 f:flush()

 wait(0.05)
end

-- never reach this because of loop for infinity
f:close()
```

Example 3.2. example2.lua

3.3. Additional functions

All other functionality such as controlling IOs, backlight or the display's contrast settings are gathered in table Table 3.4, “Additional functions”.

Command	Description	Example	LUA string
Controlling display settings			
i, I	Set display background light to *num*.	ESC[0i ESC[255i	"\27[128i"
j	Set display's contrast to *num*.	ESC[0j ESC[63j	"\27[35j"

Command	Description	Example	LUA string
IO functionality			
N	Get pin level of requested IO pin *n*[a].	ESC[2N	"\27[4N"
O	Set pin level of selected IO pin *n*[a] to *level*.	ESC[2;0O ESC[2;1O	"\27[1;1O"
Configuration			
Q	Enter LCD-Term's setup dialog.	ESC[Q	"\27[Q"
e	Jump into the device's bootloader.	ESC[e	"\27[e"

[a]See Section 4.2, “General pin-out” for details about *n*.

Table 3.4. Additional functions

When uploading GIF images, it is required to exit the firmware and enter the bootloader first. This example shows how to achieve this.

```
echo -e "\033[e" > /dev/ttyACM0
```

Example 3.3. Activate bootloader from a Linux shell

4. Technical data

Specification	Value				Unit
	LCD-Term12	LCD-Term15	LCD-Term25	LCD-Term35	
Power supply	3.3 .. 5				V
Power consumption (min.)	38				mW
Power consumption (max.)	160				mW
Operating temperature	-30 .. +85				°C
Resolution	128 x 64				Pixel
Width	25	29.4	43.2	53	mm
Length	49.1	59.3	74.4	94.8	mm
Height	5.5	6.0	6.8	7.5	mm
Pixel array	26.8 x 12.0	35.8 x 19.8	47.3 x 26.2	66.5 x 33.2	mm^2
IO pins	5	8	12	14	

Table 4.1. Characteristics

A more detailed listing of the energy requirements depending on the backlight volume is presented in Table 4.2, “Power consumption”.

Backlight	off	25%	50%	100%	Unit
Mode of operation					
idle[a]	7.6	13.4	18.0	26.6	mA
full load[a]	13.1	18.5	23.1	31.5	mA

[a] LCD-Term35: V=4.9V; baud=115200; contrast setting=35

Table 4.2. Power consumption

4.1. Power supply

A few things should be kept in mind regarding the power supply. When connecting the USB device port, LCD-Term uses the +5V line from the connected host. In this case, it also generates the +3.3V from that source.

Caution

None of the pins described as 5V shall be connected to a second power source when USB is used!

If USB is not used - neither to supply power nor as communication port - one is free to use any 5V pin as power input.

Alternatively, power can also be supplied via one of the 3.3V pins. Using a 3.3V pin as power source does not collide with any of the prior options. However, combining the 5V and 3.3V supply is not recommended.

4.2. General pin-out

All pins labeled PA*xx* / IO*yy* can be used as IO pins. They are referenced in LCD-Term's setup and IO functions as IO*yy*. The pin-out tables contain the mapping between a connector's pin number and the IO*yy* pins for the different LCD-Term boards.

```
echo -e -n "\033[3;0O" > /dev/ttyACM0
```

Example 4.1. Setting IO03[1] to low level.

4.2.1. Left connector

This connector basically contains the serial interface and power supply.

Pin	Description			
	LCD-Term12 (X4)	LCD-Term15 (X1)	LCD-Term25 (X1)	LCD-Term35 (X2)
1	3.3V[a]	3.3V[a]	3.3V[a]	3.3V[a]
2	GND	GND	GND	GND
3	TXD	TXD	TXD	TXD
4	RXD	RXD	RXD	RXD
5	PA11 / IO04	PA11 / IO07	PA11 / IO10	PA11 / IO13
6		5V[b]	5V[b]	5V[b]
7		PA24 / USB_D-	PA24 / USB_D-	PA24 / USB_D-
8		PA25 / USB_D+	PA25 / USB_D+	PA25 / USB_D+
9		GND	GND	GND
10			SCL[c]	SCL[c]
11			SDA[c]	SDA[c]
12			PA31 / IO11	PA31 / IO12
13				PA30 / IO11
14				PA15 / IO10
15				PA14 / IO09
16				PA10 / IO08

[a] Can be used as power supply; otherwise generated from +5V to supply external hardware
[b] From USB +5V; can be used as power supply when USB is not connected
[c] Currently unused; do not connect

Table 4.3. Left connector

4.2.2. Right connector

The right connector mainly comprises of IO pins.

Pin	Description			
	LCD-Term12 (X2)	LCD-Term15 (X2)	LCD-Term25 (X2)	LCD-Term35 (X4)
1	GND	3.3V[a]	3.3V[a]	5V[b]

[1] This pin occupies different pin numbers on the right connector. See: Table 4.4, “Right connector”

Pin	Description			
	LCD-Term12 (X2)	LCD-Term15 (X2)	LCD-Term25 (X2)	LCD-Term35 (X4)
2	PA00 / IO00	GND	GND	3.3V[a]
3	PA01 / IO01	PA00 / IO00	PA00 / IO00	GND
4	PA02 / IO02	PA01 / IO01	PA01 / IO01	PA00 / IO00
5	PA03 / IO03	PA02 / IO02	PA02 / IO02	PA01 / IO01
6		PA03 / IO03	PA03 / IO03	PA02 / IO02
7		PA04 / IO04	PA04 / IO04	PA03 / IO03
8		PA05 / IO05	PA05 / IO05	PA04 / IO04
9		PA06 / IO06	PA06 / IO06	PA05 / IO05
10			PA07 / IO07	PA06 / IO06
11			PA10 / IO08	PA07 / IO07
12			PA14 / IO09	PA10 / IO08
13				PA14 / IO09
14				PA15 / IO10
15				PA30 / IO11
16				PA31 / IO12

[a]Can be used as power supply; otherwise generated from +5V to supply external hardware
[b]From USB +5V; can be used as power supply when USB is not connected

Table 4.4. Right connector

4.3. Dimensions

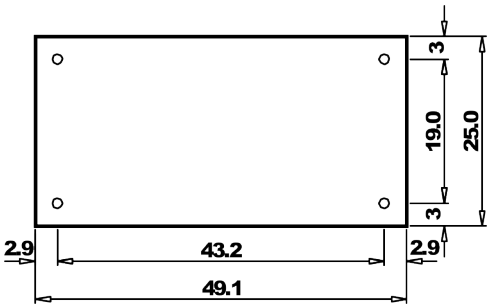

Figure 4.1. LCDTerm12 dimensions[2]

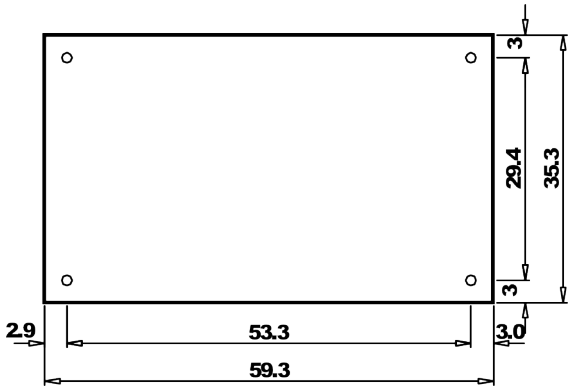

Figure 4.2. LCDTerm15 dimensions[2]

[2]All sizes are presented in mm.

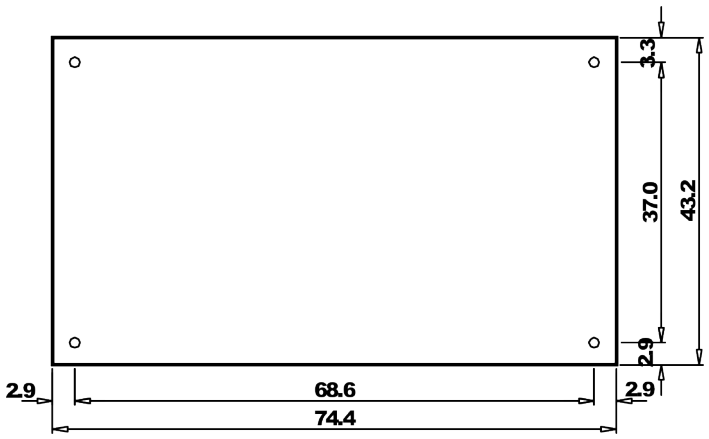

Figure 4.3. LCDTerm25 dimensions[2]

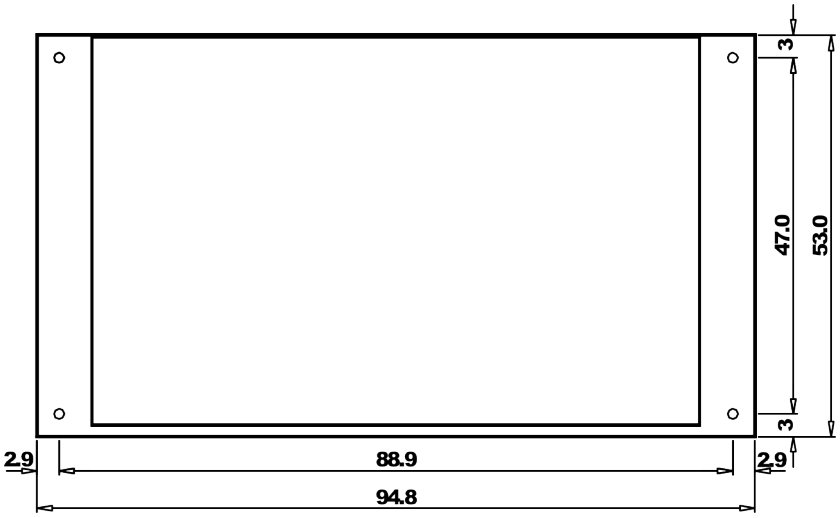

Figure 4.4. LCDTerm35 dimensions[2]