TECHNODOLLY CGI data export

The TECHNODOLLY crane is ideally suited to provide a precise and easy to use data export for virtual sets. No calibration is required because all axes are equiped with absolute encoders instead of incremental encoders. Furthermore, the TECHNODOLLY performs all calculation steps to get real cartesian coordinates from the axis positions. Its not necessary for the user to care about the geometry of the crane or about the excentricity of the head. The positional values sent are the true cartesian coordinates of a reference point on the optical axis of the camera[1].
If a fixed focal length lens (prime) is used, this reference point can be choosen on the center of perspective of the camera and, consequently, the values are equivalent to the ones of the virtual camera. In case that zoom lenses are used (where typically the shift of the center of perspective with changing focal length must be considered), this effect has to be taken into account in the CGI software.
This document describes the details of the implementation of the TECHNODOLLY data export to enable people to read the camera positions from a CGI software.

Definitions of terms

BP Boom Pan. Horizontal swing axis of crane.

BT Boom Tilt. Vertical swing axis of crane.

LH Levelling head.

The graphical user interface (GUI)

A graphical user interface (GUI) allows the user to choose the data export format and to enter some mechanical specs which vary with different head configurations. Consequently, the user interface looks slightly different when using a two-axes or a three-axes head (see Fig. 1). See Fig. 5 for an explanation of the mechanical specs.
The user can choose between *binary*, *ASCII* (text) and *kuper compatible* (similiar to ASCII but with an uncommon end-of-line delimiter) formats. For each format, you can choose between polar and cartesian coordinates.

Definition of coordinates

Fig. 2 and 3 and Fig. 4 show the TECHNODOLLY coordinate system. The x-axis is parallel to the tracks, y is the horizontal coordinate perpendicular to the tracks and z is the height above floor.

[1] pure axis data output ("polar") is also avilable

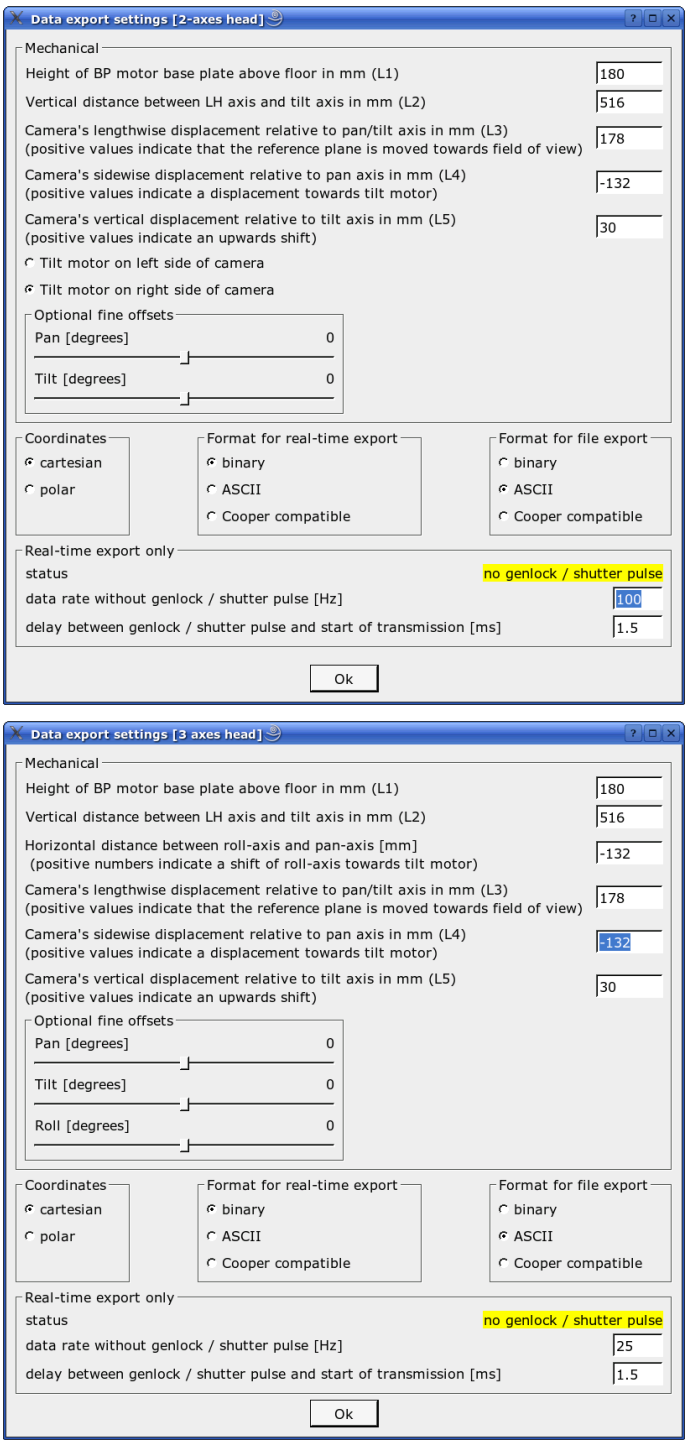

Figure 1: The user interface for a two-axes head (top) and for a three-axes head (bottom). For the meanings of the numbers in the "mechanical" section see Fig. 5. The settings in the "real-time data export only" section have no effect when writing CGI data to a file. See section genlock / shutter pulse for details.

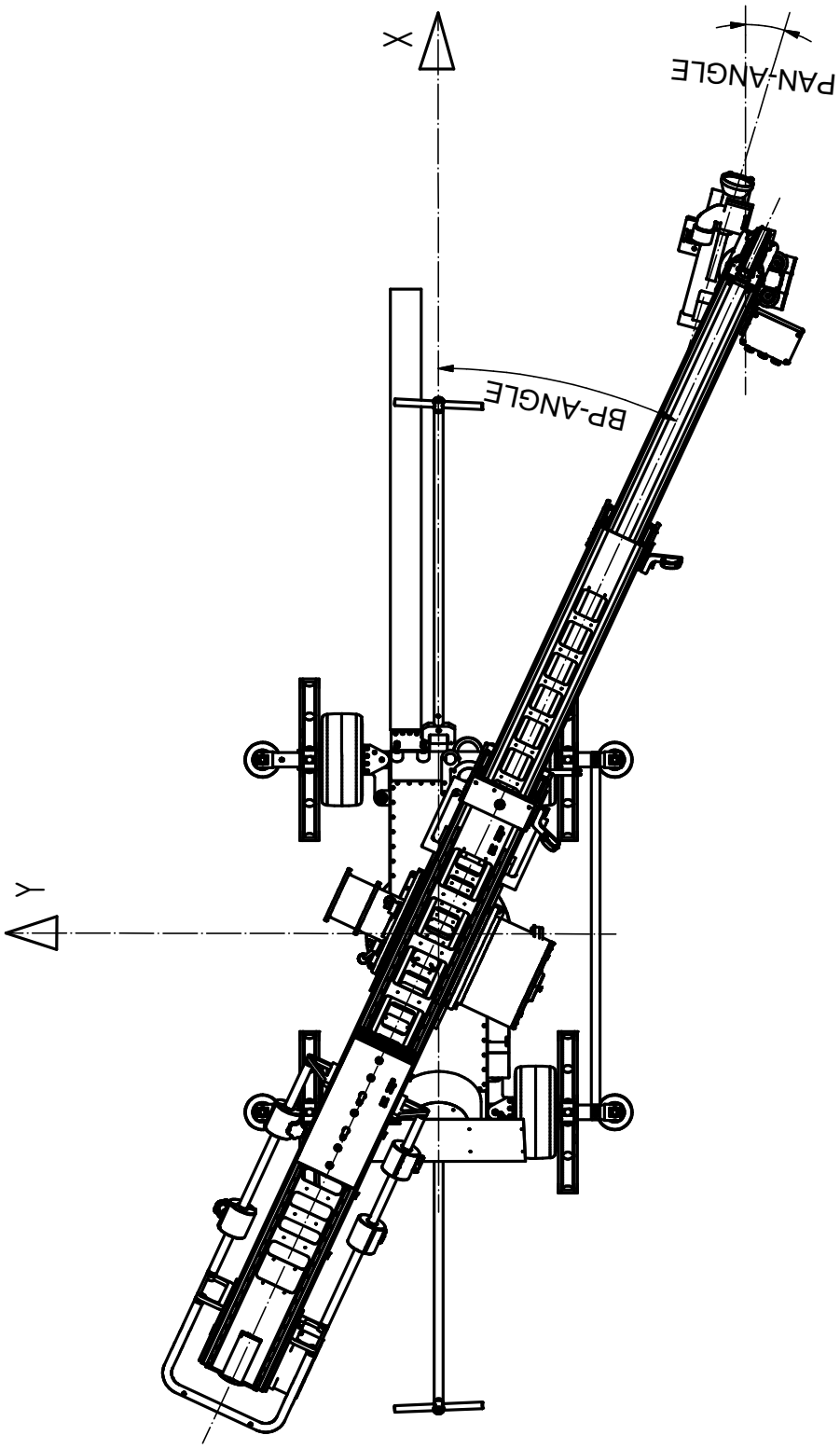

Figure 2: The TECHNODOLLY's coordinate system (top view).

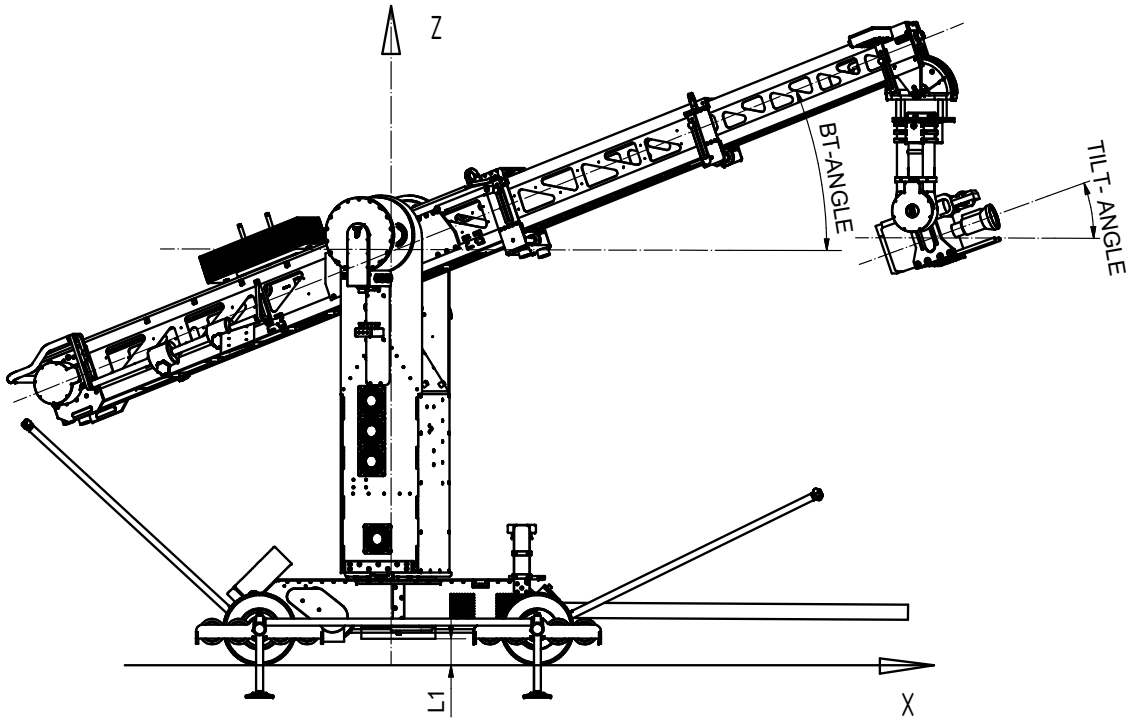

Figure 3: The TECHNODOLLY's coordinate system (side view).

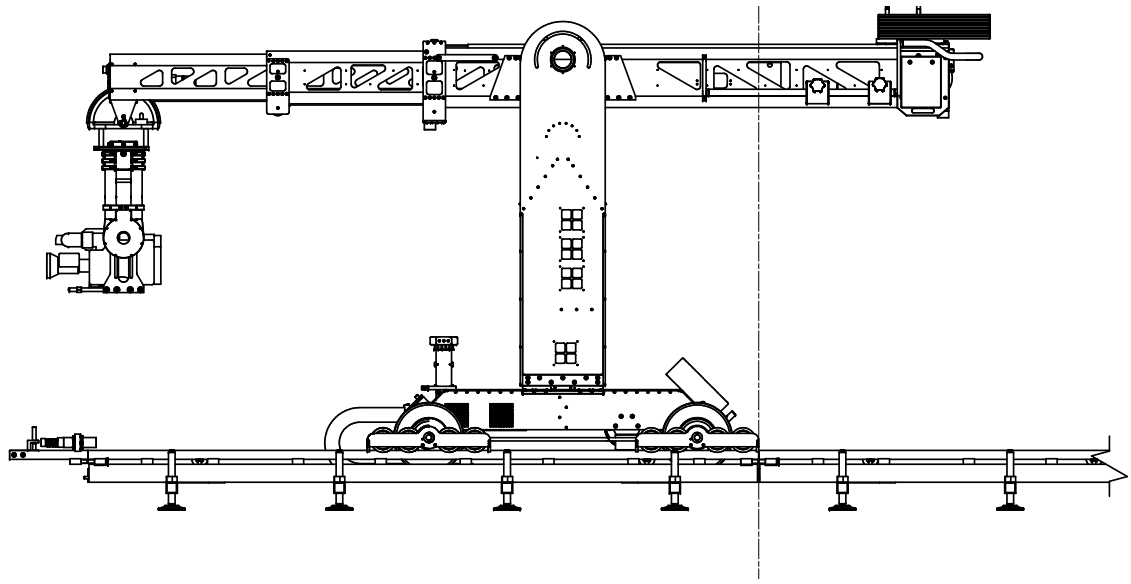

Figure 4: Zero position of track axis. When track is at the position shown, the origin of TECHNODOLLY's coordinate system is in the center of boom pan rotation and a floor height.

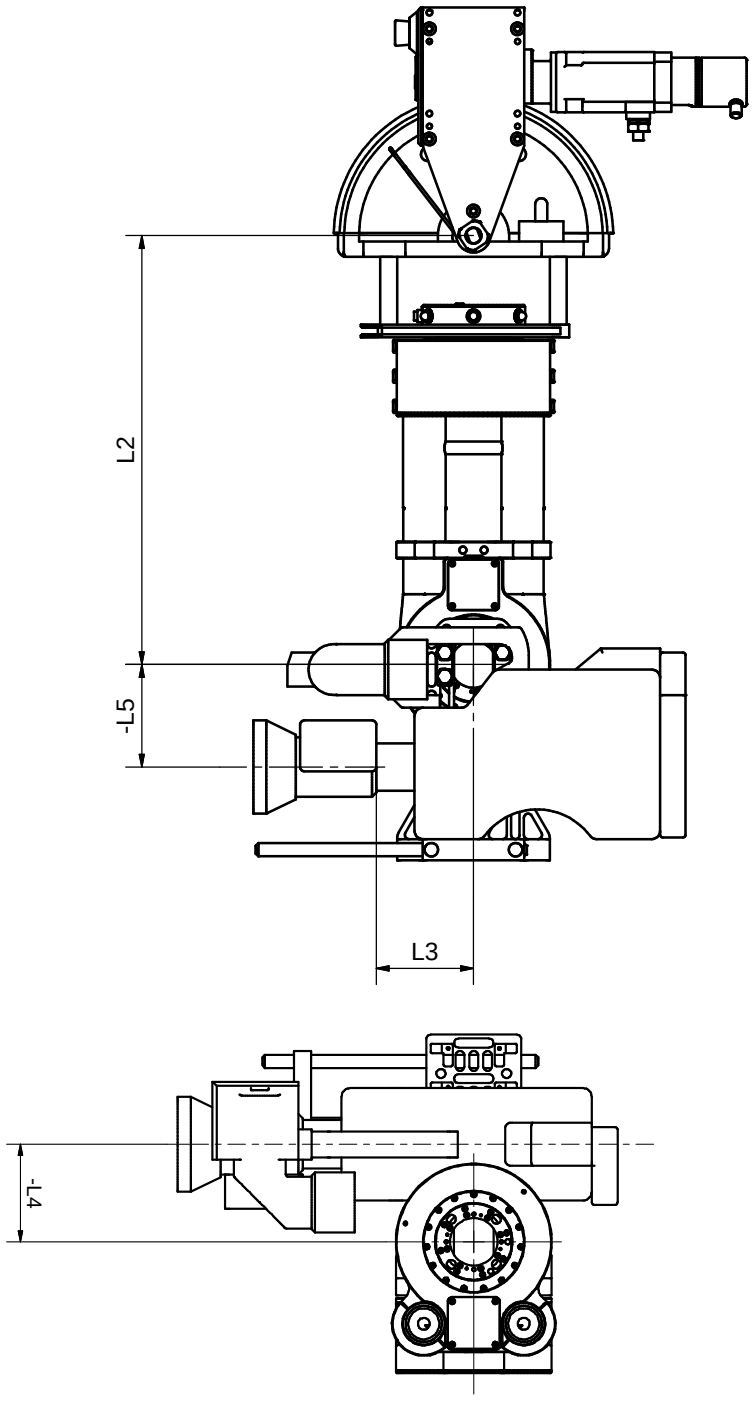

Figure 5: The definition of the four lengths L2 ... L5 which describe the displacements of the camera relative to centers of rotation. The top shows a side view, the bottom shows a top view. For definition of L1, see Fig. 3.

The origin of the cartesian space is at the intersection of boom pan axis and floor when track is at zero. Fig. 4 shows how track zero is defined for wire-drive track.[2]
For definition of pan, tilt and roll see Fig. 2 and 3. A positive pan angle means a clockwise rotation (top view). A positive tilt angle means swinging up the camera towards sky. Roll is the rotation around optical axis. A positive roll angle means turning the camera clockwise (viewing towards field of view). As can be seen, tilt and roll directly correspond to the respective motor angle while pan is also affected by BP angle.

Remarks for Maya users

For maya in y-up configuration we get the following rules:
Technodolly-x corresponds to Maya-x, Technodolly-y corresponds to negative Maya-z, Technodolly-z corresponds to Maya-y. Consequently, pan is a rotation around Maya-y, tilt is a rotation around Maya-z and roll is a rotation around Maya-x. The order of rotations is yzx. Remind that all the transformations above will be automatically done if you are using our Maya-plugin to read the TECHNODOLLY's CGI data.

Units

All lengths are given in meters. All angles are given in degrees. For lens values, the meaning of the ouput depends: If a lens calibration table is not given[3], the output numbers are simple %-values between the mechanical stops. If a lens calibration table is given the values represent real physical values. To distinguish between the two options, calibrated lens values always get a negative sign.

zoom: The value is the negative of the focal length. Units are mm.

focus: The value is the negative of the reciprocal focal distance. Units are 1/m.

iris: The value is the negative of the f-stop values.

[2] **A remark for cranes with cograil type track (as opposed to wire-drive type):** Obviously, the x-coordinate is affected by the position on the track. Because the track motor can be locked to the cograil at any position, you can have quite large offsets for the x-coordinate. To get a well-defined origin along x-axis run the move called "zero.move" before locking the track motor to the cograil.

[3] You can always use uncalibrated (raw) value by un-clicking "always use raw (uncalibrated) zoom / focus / irsi values" in data export settings window.

Type of link

At the moment, the data output uses a serial port. RS232 outputs are provided (for shorter cable lengths) as well as RS422 outputs (for longer cables).The bit rate is 115200 bits per second (BAUD), one start bit and one stop bit, no parity. The endianess is little endian, that means for a multi-byte quantity, the lowest significant byte is transferred first.
Later on , we will also support ethernet transmision (UDP protocol)

Genlock / shutter pulse

Let's first summarize the folowing section: You need genlock or shutter pulse

- if you repeat the same move and you are planing to combine diffrent shoots in postprocessing or
- if you are using real-time data export.

Now, the details:

Genlock /shutter pulse and repeatability

In many situations, the TECHNODOLLY is used to repeat the same move several times. Later on in post-processing, the recorded shoots (video or film) are overlayed. In this situation the TECHNODOLLY's bloop-plate is used. The bloop plate produces a short flash when starting the move and allows to identify the first frame.
Iamgine that we shoot PAL video (25 frames per second) and that the crane is moving with a moderate speed of 1 m/s. If we do not synchronize the start of the TECHNODOLLY move with the image formation on the CCD chip somehow, we have to expect a time lapse up to 1/25 seconds between corresponding frames. This leads to an positional error up to 4 cm, which can be well visible. Remind that this error is arbitrary, two shoots may be in perfect coincidence by good luck or may show a large error.
The solution to this problem is to provide a genlock or shutter pulse signal. With a genlock or shutter pulse signal present, the TECHNODOLLY will not instantly start the move when the red button at the bloop plate is pressed. Instead, it waits for the next genlock or shutter pulse before actually starting the move [4]. This little delay (for example, 0.04 seconds maximum for PAL genlock) will eliminate the described positional error.

[4]With genlock, the crane acually waits for the next pulse which belongs to an *odd* field (remind that the supported video formats are intelaced).

Genlock /shutter pulse and data export

Even if not overlaying different shoots there is another reason to use genlock or shutter pulse: Not only the move is synchronized to the genlock or shutter pulse but also the real-time data export.
To understand the problem imagine the following situation: You shoot PAL video in a greenscreen studio at nominally 25 fps. TECHNODOLLY's real-time data export is connected to a rendering computer which runs e.g. VizRT software. For each new frame to render, this system uses the most recent camera positional data available.
You set the TECHNODOLLY data rate to 25 fps but due to variations in either the TECHNODOLLY's timebase or the camera's timebase the data rate is slightly lower then the frame rate. As a consequence, at certain frames it happens that *two* frames rendered by the VizRT system use the same positional data from TECHNODOLLY: For the previous frame, a new data packet had arrived just in time. For the actual frame, the next data packet was not yet available because the data rate is slightly slower then the frame rate. The result are visible jerks in the rendered images which are not acceptable for broadcasting.
Again, the solution is to provide a genlock or shutter pulse signal to the TECHNDOLLY.

Delay between genlock / shutter pulse and start of data

The previous thoughts also explain the meaning of "delay between genlock / shutter pulse and start of data" in the data export window (see Fig. 1). With bad luck, it may happen –even with genlocking– that data packets are transmitted at unappropriate instants of time. That means that the completion of the packets is very close to the moment when the rendering software scans for new data. Then, the system may use the just arrived data or the previous one, depending on unavoidable variations in data processing time.
The solution to this problem is to slight shift data transmission relative to genlock. As a rule of thumb, if you observe arbitrary jerks in the mixed image even with genlock, you should add one quarter frame periode (10 ms for PAL) to the value. Then the jerks will disappear.

Data rate

The data rate is either determined by the Gen-Lock signal or —if no Gen-Lock is present— set by the user in the GUI. The maximum data rate (binary & cartesian) is 180Hz.

The ASCII & cartesian data stream

This is the most recommended format to import a TECHNODOLLY CGI data file into external programs. It is a simple, comma-separated format. Such a file would look like

```
R0.00,-1.2666,0.9436,0.8534,-35.011,-179.043,0.000,100.000,100.000,100.000,123
R1.00,-1.2666,0.9436,0.8534,-35.011,-179.043,0.000,100.000,100.000,100.000,124
R2.00,-1.2666,0.9436,0.8534,-35.011,-179.043,0.000,100.000,100.000,100.000,125
.
.
.
```

Each packet starts with a capital letter which is 'R' if the crane is running a pre-programmed move and which is 'S' otherwise. Each packet ends with a line feed (0xa, '\n') [5]. In between, there are a couple of comma-separated numbers. These numbers are frame number, X, Y, Z, pan, tilt, roll, zoom, focus, iris and finally absolute frame number [6]. "frame number" is reset to 0 for each start of a move while "absolute frame number" counts the absolute number of frames since the technodolly was started up. See also sections "Definitions of coordinates" and "Units" for details.
For programmers: The data can be generated with a c-statement similiar to

```
printf("%c%d.00,%.4f,%.4f,%.4f,%.3f,%.4f,%.4f,%.4f,%.4f,%d%c",
       startLetter, frameNumber,
       X, Y, Z,
       pan, tilt, roll,
       zoom, focus, iris, absoluteFrameNumber, '\n');
```

The binary & cartesian data stream

Each packet has a length of 64 bytes. It starts wit a constant start indicator (0x7f7a5aa5) which is choosen to never represent a real floating point number. The floats are IEEE 32-bit floating point numbers. u8 means an 8-bit unsigned integer (typically eqivalent to unsigned char on most systems), u32 is a unsigned 32-bit integer (typically eqivalent to unsigned int on most systems) et cetera. See the tabel below and the c-code samples.
Introduced end 2012, Technodolly time code. In that case, the meaning of the two fields "frameNumber" and "time" changes to contain timecode information instead (called version 2). To find out which version is actually used, a reader can check the MSB of "frameNumber". If this bit is set, it means that timcode (version 2) is used.
The table below shows the contents of a data packet. The endianess is little endian, that means for a multi-byte quantity, the lowest significant byte is

[5] In kuper compatible mode, each packet is ended by the combination of a carriage return (0xd, , '\r') and a zero-character (0x0, '\0')

[6] The last field "absolute frame number" was introduced in version 610 mainly to help using the permanent CGI data streams

transferred first. For example, the sync patter will be transferred in the order `0xa5`, `0x5a`, `0x7a` and `0x7f`.

meaning	data type	data length (bytes)	units
sync pattern (0x7f7a5aa5)	u32	4	-
packet number	u32	4	-
frame number or timecode	u32	4	-
time or timecode user bits	float	4	s
x	float	4	m
y	float	4	m
z	float	4	m
pan	float	4	degrees
tilt	float	4	degrees
roll	float	4	degrees
zoom	float	4	% or mm
focus	float	4	% or 1/m
iris	float	4	% or f-stop
spare area	struct spareArea	4	-
track	float	4	m
checksum	u32	4	-

The following c-declarations shows the definition of c-structures which represent a binary packet:

```
/*
 * This is the definition of the 64-bit bitfield used in
 * CGIDataCartesianVersionX. At the moment, it holds just two bits:
 * "running" indicates that TECHNODOLLY is running a preprogrammed move.
 * "cameraOn" indicates that camera is on.
 */
struct spareArea{
  unsigned int running  :  1;
  unsigned int cameraOn :  1;
  unsigned int spare    : 30;
  //30 bits for future use
};

/*
 * Make sure that your compiler packs the structure below (check that CGI_DATA_LENGTH equals 64).
 */
struct CGIDataCartesianVersion1{
                            //Version without time code, the only one until November 2012
  u32 syncVal;              //always 0x7f7a5aa5 to detect beginning of packet
```

```
  u32 packetNumber;
  u32 frameNumber;          //MSB is always 0. If MSB is set, it means that we actually
                            //have a packet of version 2 containing timecode.
  float time;               //[seconds]
  float x,y,z;              //[m]
  float pan, tilt, roll;    //[degrees]
  float zoom, focus, iris;  //A negative value indicates that calibrated lens values are used.
                            //Uncalibrated: % from one stop to the other
                            //Calibrated: zoom  = - focal length [mm]
                            //            focus = - 1 / focal distance [1/m]
                            //            iris  = - f-stop

  struct spareArea spare;   //32 bits, most are free for future use.
  float track;
  u32 checkSum;
};



struct CGIDataCartesianVersion2{
                            //Version with time code.
  u32 syncVal;              //always 0x7f7a5aa5 to detect beginning of packet
  u32 packetNumber;
  struct timeCodeStruct timeCode;
                            //MSB is always 1. If MSB is 0, it means that we actually
                            //have a packet of version 1 containing frame-numbers and time.

  u32 timeCodeUserBits;
  float x,y,z;              //[m]
  float pan, tilt, roll;    //[degrees]
  float zoom, focus, iris;  //A negative value indicates that calibrated lens values are used.
                            //Uncalibrated: % from one stop to the other
                            //Calibrated: zoom  = - focal length [mm]
                            //            focus = - 1 / focal distance [1/m]
                            //            iris  = - f-stop
  struct spareArea spare;   //32 bits, most are free for future use.
  float track;
  u32 checkSum;
};
```

The structures above are also used within or testprograms (*datagenerator* and *datadisplay*). Remind, however, that the implementation of bitfileds is not guaranteed to be portable across machine boundaries. You may have to change the bit positions within `struct spareArea` and you may have to enforce that the `struct CGIDataCartesian` is packed.

Notes for developers of plugins to read the TECHNODOLLY's real time data

ASCII format

Each lines starts with an english letter (right now, just the two letters 'R' and 'S' are used). After that, we have just digits, commas (','), periods ('.') and the newline character ('\n'). Even if we use different letters in the future to display additinal information, we will always keep this basic structure. That means that your reader routine should find the start of a new packet by scanning for a character which is a letter [A-Z, a-z].

Furthermore, we may append additional comma separated fields in the future. Consequently, you should not rely on the fact that we have exactly 10 numerical fields (frame number, X, Y, Z, pan, tilt, roll, zoom, focus, iris) after the starting character. But, the first 10 fields will always contain the values above in exactly that order.

Binary format

Always keep in mind that the serial interface is inherently a character oriented interface. It does not support a concept like a packet. For a serial interface, everything is just a stream of 8-bit quantities [7]. To scan for the sync val, you have to search for the occurence of the four characters 0xa5, 0x5a, 0x7a and 0x7f (in that order!). Take a look at the code example in cgidata.c and cgidata.h. There are complete and tested functions to synchronize to the serial data stream and to test for checksum errors (written in c language, but should be easy to port to other languages as well).

The testprograms *datagenerator* and *datadisplay*

We provide two testprograms to help in developing interfaces to other programs and to simplify the task of inspecting and converting the data export of TECHNODOLLY cranes. The programs are open source and are covered by the GNU public license. Some basic procedures (contained in files `cgidata.h` and `cgidata.c` are covered by an even more permissive license to allow to be incorporated into closed-source software.)
Both programs run natively under the LINUX operating system. If you don't have LINUX comupter available, you can easily use a live distribution like Knoppix (www.knoppix.org). This is a full-featured LINUX system which boots from DVD without making any changes to your local harddisk.

Installing the testprograms

The package comes as a simple tar-archive (td_cgi.tgz). Create a directory whereever you want, unpack the archive and run make. This could, for example, be done by entering (finish every line by pressing enter):

```
cd
mkdir td
cd td
tar -xzvf td_cgi.tgz
make
```

[7] Seriel interfaces support 5, 6 and 7-bit quantities as well. The character size of the TECHNODOLLY data export is always 8-bit.

This will create the two executables datagenerator and datadisplay. That's all.

General notes

The operate correctly, both programs need to know the name of the serial port to use (device name). For computers with built-in serial interfaces (9-pin D-SUB male connector) this is typically `/dev/ttyS0` (which is the program's default if no `--device` or `-d` option is given). On some comupters it may also be `/dev/ttyS1` or `/dev/ttyS2`. On computers without built-in serial ports, you can either use PCI or PCIe serial cards (in that case, the name will be `/dev/ttySX`, X = 0,1,2... as well) or you use a USB-to-serial adapter. Most of these work well with linux, some may not. They appear typically at `/dev/ttyUSB0`.
For testing, we recommend to build your own "loopback" adapter: Get a 9-pin female D-SUB connector and connect pin 2 (receive input) with pin 3 (transmit output). Plug it in to the serial port you want to use. Then simply try out the different possible device names by starting `datagenerator` and `datadisplay` with the same `--device` argument and see for which device name it works.
Whether you can run the programs as a normal user or not depends on the permissions of your respective device nodes. If you have read- and write-access you can simply start the programs by typing `./datagenerator` or `./datagenerator` from the installation directory. If only root has the read and write access rights, you need to type `sudo ./datagenerator` or `sudo ./datagenerator` and will be asked for root's password.

datagenerator

This program simulates the data output of a TECHNODOLLY crane. It continously writes data to serial port (or to a file) which has exactly the same format as produced by TECHNODOLLY cranes. The program always use cartesian coordinates. You can choose between binary data (the default) and ASCII (by using the respective option). To invoke the program from the instalation directory type

```
sudo ./datagenerator [OPTIONS]
```

To stop the program, just type CTRL-c. The following command line options are recognized:

1. General options
continued on next page

continued from previous page

`-h` `--help`	Show a detailed help screen.
`-v` `--verbose`	Increase verbosity.
`-t` `--ascii`	Generate ASCII output instead of binary.
`-d DEV` `--device DEV`	Write to character device or file DEV. Defaults to /dev/ttyS0.
`-f FLOAT` `--fps FLOAT`	Generate FLOAT packets per second. Defaults to 25.
`--timecode`	Generate timecode. In ASCII mode, the timecode is appended in the usual format hh:mm:ss:ff at the end of each data line. In binary mode, the format defined in 'struct CGIDataCartesianVersion2' is used. At the moment, only PAL timecode (25 fps) is generated, consequently, it doesn't make much sense to combine the timecode option with anything other then 25 fps.
`-e NUM` `--errors NUM`	Generate a checksum error for each NUMth packet.

2. Options affecting the camera moves

These options can be specified several times and affect only camera moves which are specified afterwards. They are overridden by any new occurence of the respective command line switch. See also examples.

`-c FLOAT` `--cycle FLOAT`	Set the periode for subsequent moves to FLOAT seconds. The default depends on the move type.
`-a FLOAT` `--amplitude FLOAT`	Set the amplitude for subsequent moves to FLOAT. The default depends on the move type.
`-o FLOAT` `--offset FLOAT`	Set the offset for subsequent moves to FLOAT. The default depends on the move type.
`-p FLOAT` `--phase FLOAT`	Set the phase for subsequent moves to FLOAT degrees. The default is zero.
`-s` `--no-smooth`	Use trapezoidal oscillations instead of sinusoidal ones.
`--run-start FLOAT` `--run-end FLOAT`	These options do not affect the positional data output itself. They just set the start and end time (in seconds) for indicator bit "running pre-programmed move". The default is 0 and infinity, repsectively (i.e. bit is set all the time).

continued on next page

continued from previous page

`--cam-start FLOAT` `--cam-end FLOAT`	These options do not affect the positional data output itself. They just set the start and end time (in seconds) for indicator bit "camera on". The default is 0 and infinity, repsectively (i.e. bit is set all the time).

3. Options selecting the camera moves

These options can be combined as long as they affect different axes. The `--random` and `--random-full` options affect all axes and, consequently, cannot be combined with any other camera move selection.

`--x`
`--y`
`--z`
`--pan`
`--tilt`
`--roll`
`--zoom`
`--focus`
`--iris`
	Generate simple oscillating moves for the respective axes. The respective axis will move according to
	$$\text{value} = \text{offset} + \text{amplitude} \times \sin\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right),$$
	where time is the time since starting the program in seconds. If option `--no-smooth` is given, a trapezoidal function is used instead of a sine. The defaults for (`cycle, amplitude, offset, phase`) are:
	`--x, --y, --z`: (5, 1, 0, 0)
	`--pan, --tilt, --roll`: (5, 45, 0, 0)
	`--zoom, --focus, --iris`: (5, 0.5, 0.5, 0)
	These options can be arbitrarily combined to get a camera move whith more then one axis moving (see also examples).

`--panTilt`	Pan and tilt draw a circle (smooth) or square (non-smooth) in object plane. Mathematically: $$\text{pan} = \text{amplitude} \times \sin\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right)$$ $$\text{tilt} = \text{amplitude} - \text{amplitude} \times \cos\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right)$$ Again, if option `--no-smooth` is given, a trapezoidal function is used instead of a sine resulting in a square instead of a circle. The defaults for (`cycle, amplitude, phase`) are (10, 45, 0).

continued on next page

continued from previous page	
`--xyPan`	A circle (smooth) or square (non-smooth) in xy-plane, camera watching the origin. Mathematically: $$\begin{aligned} \text{x} &= \text{amplitude} \times \\ &\quad \sin\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right) \\ \text{y} &= \text{amplitude} \times \\ &\quad \cos\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right) \\ \text{pan} &= -\frac{360}{2\pi}\ \text{atan2}(-y, -x) \end{aligned}$$ Again, if option `--no-smooth` is given, a trapezoidal function is used instead of a sine resulting in a square instead of a circle. The defaults for (`cycle, amplitude, phase`) are (10, 3, 0).
`--loop`	An aircraft inside loop. Mathematically: $$\begin{aligned} \text{x} &= \text{amplitude} \times \\ &\quad \sin\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right) \\ \text{x} &= -\text{amplitude} \times \\ &\quad \cos\left(2\pi\left(\frac{\text{time}}{\text{cycle}} + \frac{\text{phase}}{360}\right)\right) \\ \text{pan} &= -\frac{360}{2\pi}\ \text{atan2}(-y, -x) \end{aligned}$$ The defaults for (`cycle, amplitude, phase`) are (10, 3, 0).
`--random`	Put binary random data in all fields except sync pattern, packetNumber, frameNumber, time and check sum.
`--random-full`	Put binary random data in all fields except sync pattern and check sum.

Examples

To generate test data where pan oscillates between -90 and +90 degrees with a periode of 8 seconds and to send it to serial port `/dev/ttyS1` at a rate of 24 dataframes per second (all other axes will be constantly 0) use:

```
sudo ./datagenerator -d/dev/ttyS1 -a90 -c8 --pan --fps 24
```

The same as above but additionally, the “running pre-programmed move”

indicator bit is set between 8th and 16th second. Hence, the second cycle of pan oscillation is marked by that indicator bit. The backslash can be used to split the command to several lines.

```
sudo ./datagenerator -d/dev/ttyS1 -a90 -c8 --pan --fps 24 \
--run-start 8 --run-end 16
```

Now the same as above but usind ASCII output. Between 8th and 16th second, each line is started with the letter 'R' indicating that we are "running pre-programmed move". Outside this period, each line is started with the letter 'S'.

```
sudo ./datagenerator -d/dev/ttyS1 -a90 -c8 --pan --fps 24 --ascii \
--run-start 8 --run-end 16
```

Remind that you can abbreviate each long option as long as it remains unambigous. Consequently, the following command is equivalent to the previous one:

```
sudo ./datagenerator -d/dev/ttyS1 -a90 -c8 --pan --fp 24 --as \
--run-s 8 --run-e 16
```

To make the pan oscillate between -30 and +30 degrees with a period of 4.4 seconds and to put the crane to constant position $(x, y, z) = (1, 2, 3)$ use:

```
sudo ./datagenerator  -a30 -c4.4 --pan -a0 -o1 --x -o2 --y -o3 --z
```

To generate binary arbitrary data with at a rate of 150 dataframes per second and to generate a check sum error every 30 seconds (i.e. every 7500th dataframe) use:

```
sudo ./datagenerator  --random -f150 -e7500
```

datadisplay

This program displays the data export of TECHNODOLLY camera cranes in a human-readable fashion. The output is written to standard output. The program can be seen as a binary to various-test format converter which also keeps track of communication errors. To invoke the program from the installation directory type

```
sudo ./datadisplay [OPTIONS]
```

To stop the program, just type CTRL-c. The following command line options are recognized:

`-h` `--help`	Show a detailed help screen.
`-v` `--verbose`	Increase verbosity.
`-d DEV` `--device DEV`	Read from character device or file DEV. Defaults to /dev/ttyS0.
`-s` `--no-statistics`	Don't display some statistical information for each packet. Check sum errors, however, are always written to standard error stream.
`-p` `--no-packets`	Don't display packet data.
`-u` `--no-unsynced`	Don't display characters when the receiver is out of sync.
`-x` `--hex`	Show floats as hex values. Good for debugging.
`-c` `--columns`	Ouput raw column based text. Each packet generates a line and each field corresponds to a column. Good for interfacing to external programs. The order of columns corresponds to the order of fields in `CGIDataCartesian`.
`-r` `--realtime`	Ouput raw column based text equivalent to the ASCII real time data string (comma separated data). Good for converting from binary to ASCII.

Examples

To receive data from an attached USB-to-serial converter and to print it to screen use:

```
sudo ./datadisplay -d/dev/ttyUSB0
```

To receive data from the standard serial port and to write the output to a file output.dat use (checks sum errors will still appear in the terminal window):

```
sudo ./datadisplay > output.dat
```

To convert a binary file `file.bin` to a simple space-separated ASCII file `file.asc` use:

```
datadisplay -d file.bin -cs > file.asc
```

Here, we don't use the `sudo` command because we simply operate on a regular file instead of the serial interface.