Electrical Lens Driver 4

Lens Driver Manual

The Electrical Lens Driver offers a simple yet precise way to control Optotune's electrical lenses. There are two types of lens drivers available. The Electrical Lens Driver 4 is used to drive the EL-6-18 and EL-10-30-C lens series and comes in a plastic housing. The second Lens Driver 4i is suited to drive the EL-10-30-Ci industrial version and is contained in a steel housing. Both drivers can be used as a standalone solution or integrated into OEM designs. The main features are:

- Current control range up to -290 to +290 mA with 12 bit precision
- Drive frequencies from 0.2 to 2000 Hz (rectangular, triangular or sinusoidal)
- I2C sensor read-out e.g. for temperature compensation
- USB powered
- Driver software in Windows 7 and Windows 8
- Available with or without housing

Mechanical specifications	**Lens Driver 4**	**Lens Driver 4i**	
Dimensions (L x W x H)	77 x 19 x 13	99.05 x 19 x 13.5	mm
Weight	11	41	g
Interface to lens	0.5mm pitch 6 way FPC connector	6-pin Hirose connector	
Interface to PC	USB Type A		

Electrical specifications	**Lens Driver 4**	**Lens Driver 4i**	
Maximum output current	Up to 290mA, depending on resistance (see Figure 8)		
Drive current range for EL-6-18	-290 to +290		mA
Drive current range for EL-10-30-C, -Ci	-200 to +200		mA
Maximum output update frequency	>100		kHz
USB input voltage	5		V
Power consumption	50-1100		mW
Digital to analog converter	12 bit Analog Devices ADN8810		
Microcontroller	8-bit, 16 MHz with 32 KB Flash (Atmel ATmega32U4)		
Connector	6-way FPC (Molex 503480-0600)	6-way Hirose HR 10 G	

Thermal specifications	**Lens Driver 4**	**Lens Driver 4i**	
Operating temperature	-20 to +65		°C
Storage temperature	-40 to +85		°C

Integration into OEM systems

Both Lens Drivers are easily integrated into OEM systems. The microcontroller that is used offers serial interfaces over USB, RS-232 or SPI. Also, analog input from 0-5V is available. Schematics and part list of the Lens Drivers are available on request. Documentation of the firmware is presented at the end of this document. For more information please contact sales@optotune.com.

System requirements

- Windows 7 or Windows 8.1, 32/64 bit
- Lens driver: USB 2.0 port
- uEye camera: USB 2.0 port (preferably 3.0)

Hardware Operation

Connecting the EL-10-30-C

The Molex flex cable of the EL-10-30-C lens can be plugged directly into the connector of the lens driver. The copper side of the cable has to be upwards and the black clamp has to be closed.

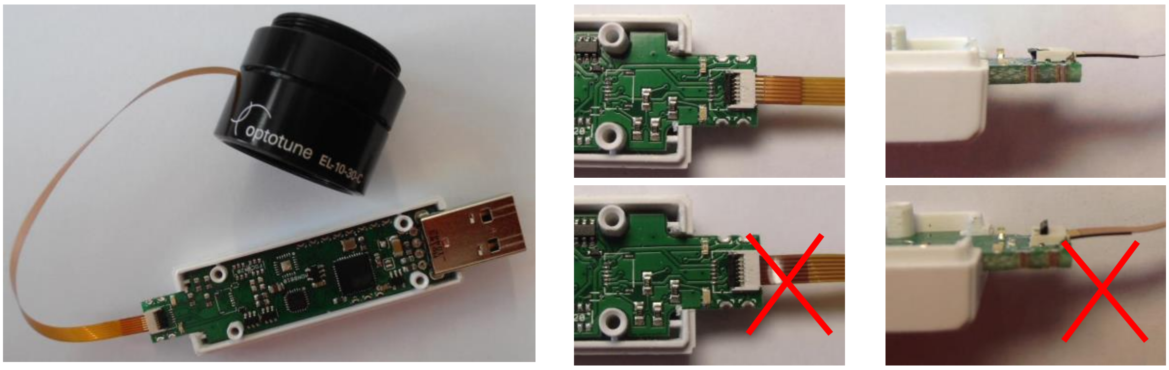


Figure 1: Connecting the EL-10-30-C directly to the lens driver.

Connecting the EL-10-30-C to the lens driver with an extension cable

If a larger distance is required, the easiest way is to use an USB extension cable for the lens driver. If a larger distance in between the lens and the driver is required, an extension as described in Figure 2 can be built. For large distances, shielded cables are recommended to ensure interference-free performance of the I^{2}C bus. The butterfly connector and the 5cm long transition cable are provided with the lens driver.

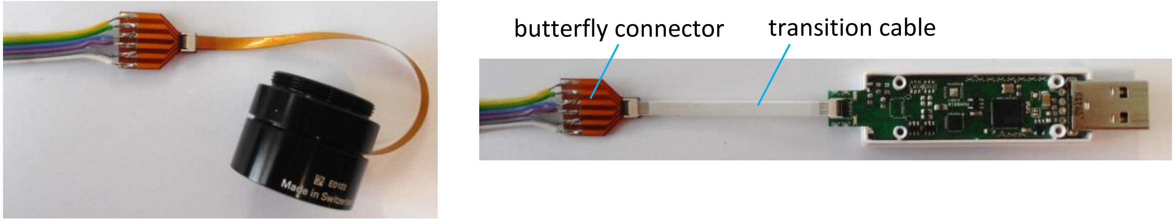


Figure 2: Connecting the EL-10-30-C with an extension cable to the lens driver.

Connecting the EL-10-30 compact to the lens driver

The 30cm long cables of the EL-10-30 compact can be directly soldered to the lens driver. The plus and minus poles are indicated in the figure below.

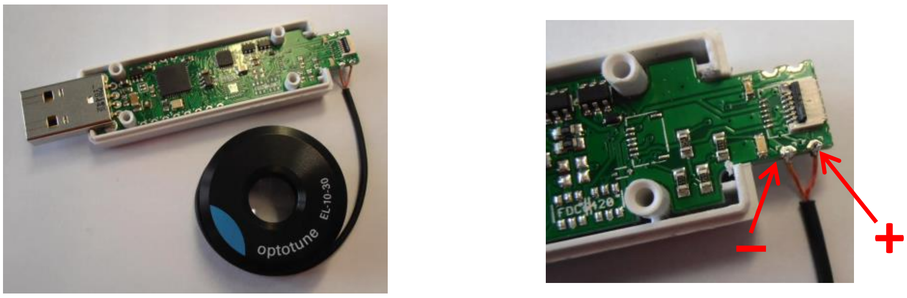


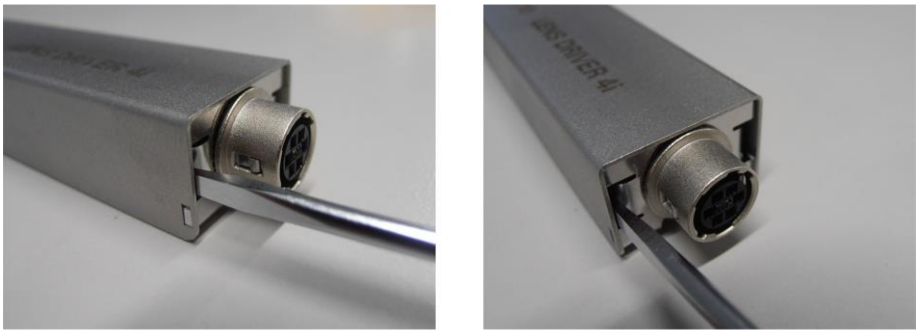
Figure 3: Connecting the compact EL-10-30 to the lens driver by soldering the plus and minus poles.

Connecting the EL-10-30-Ci to the lens driver

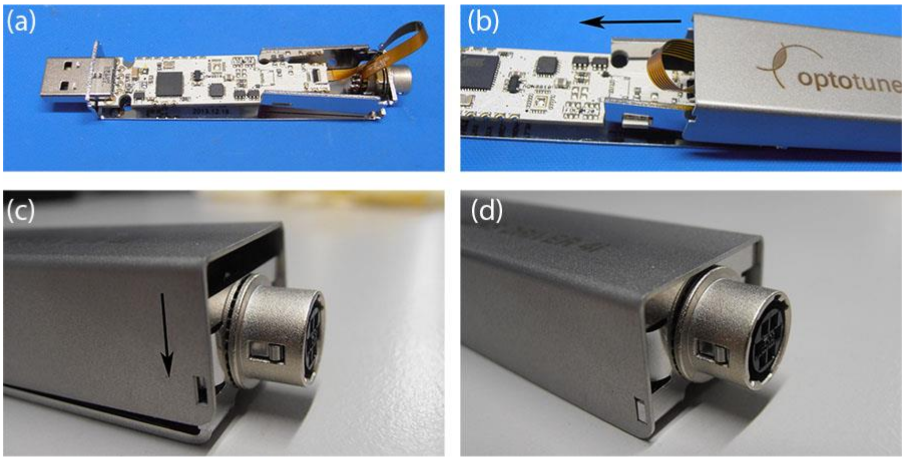
The connection of the industrial EL-10-30-Ci to the Lens Driver 4i is straight forward. Simply connect the cable to the plug, the position of the pins is unambiguous.

Open and close housing of Lens Driver 4i

To open the housing use the gaps on the side to spread the cover side-walls with an appropriate flat head screw driver. Hold the enclosure base and lift the cover to open the casing:



In order to close the housing, place the cover on top of the enclosure base (a) and move it towards the USB stick (b) until the front side clips fit in the openings of the cover. Then push the cover at the back downwards (c) until the back side clips snap into the square holes (d). If the clips do not snap in, gently apply pressure from the top/bottom/side to move the cover forward/backward or up/down relative to the base unit.



Connector Pinout

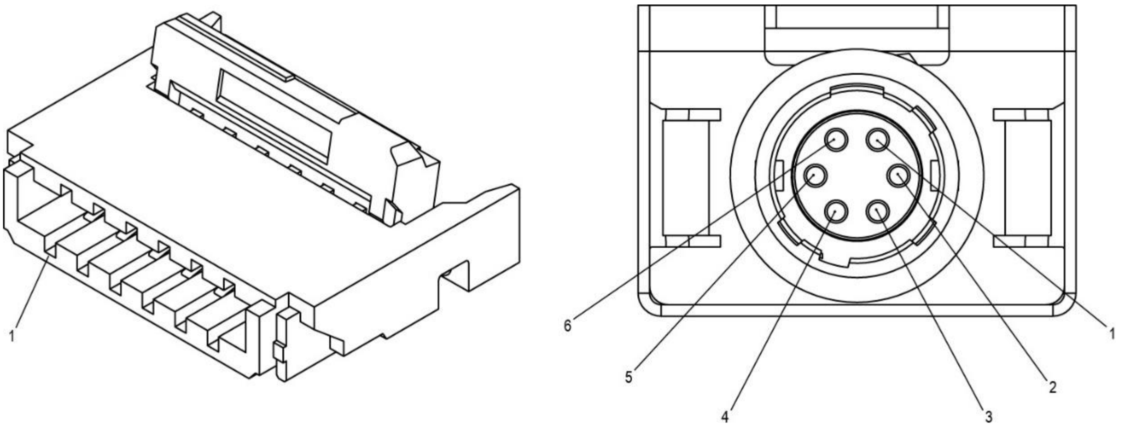


Figure 4: Connector pinout of the Lens Driver 4 (left) and Lens Driver 4i (right).

Lens Driver 4			Lens Driver 4i	
Position	**Function**		**Position**	**Function**
1	I2C Gnd		1	Lens (+ pole)
2	Lens (- pole)		2	Lens (- pole)
3	Lens (+ pole)		3	I2C Gnd
4	I2C SDA		4	I2C Vcc 3.3V
5	I2C SCL		5	I2C SCL
6	I2C Vcc 3.3V		6	I2C SDA

Lens Driver Controller Software

Software Installation

- Run Setup.exe
- Follow the installation wizard

Installation of the Windows driver in Windows XP and Windows 7

- Start Lens Driver Controller. Should the Windows driver not yet be installed, you will be shown a window describing a detailed installation procedure.

Installation of the Windows driver in Windows 8.1

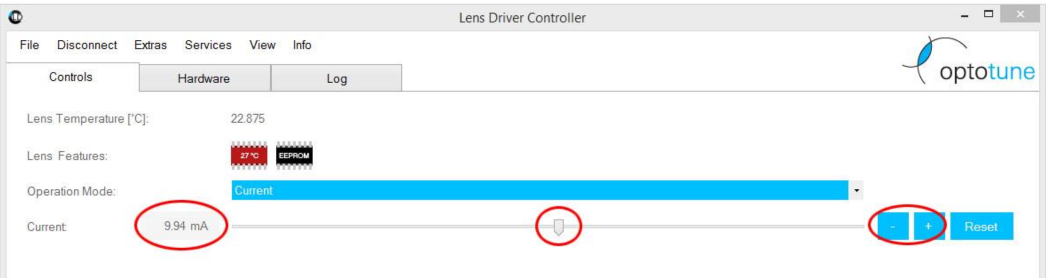
- Installation of the Windows driver in Windows 8.1 requires disabling the driver signature enforcement first. The following steps explain this procedure.
- Press the **Win + C** keyboard combination to bring up the charms bar, then click on the **Settings** charm.
- Click on **Change PC settings**.
- In Control Panel, click on **Update and recovery**.
- Now click on **Recovery**.
- On the right hand side in the **Advanced Startup** section click on **Restart now**.
- Once your computer has rebooted click on **Troubleshoot**.
- Now click on **Advanced Options**, then **Startup Settings** and finally **Restart**.
- Your computer will now restart again and show you a list of options. Select option **7) Disable driver signature enforcement** by pressing **F7** on your keyboard.
- Your computer will now restart with driver signature enforcement disabled and you may continue with the normal driver installation procedure. In order to do so, start Lens Driver Controller. Should the Windows driver not yet be installed, you will be shown a window describing a detailed installation procedure.

Operating Lens Driver Controller

Launch Optotune Lens Driver Controller and click on **Connect.** This will establish the hardware connection and open the main window with the current control and the temperature readout.

Controls

Figure 5 shows the main window. The output current can be changed, either by shifting the arrow or by using the +/- buttons. Using the +/- buttons together with the Shift key increases the step size. Alternatively, the desired value can be written in the gray box.



Lens Temperature [°C]: 22.875

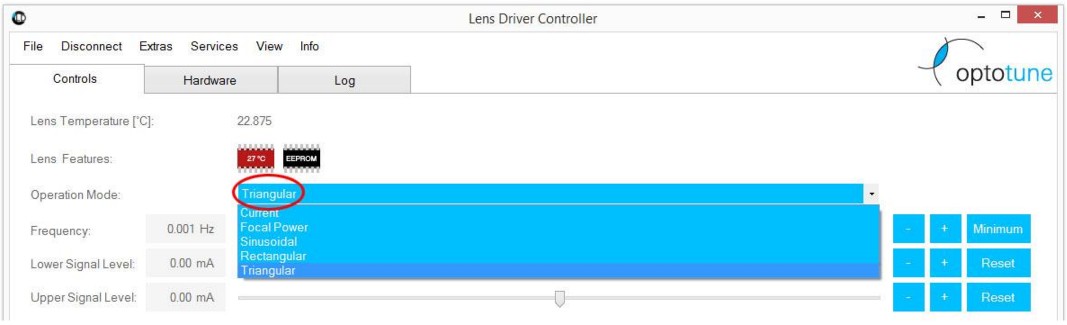
Lens Features:

Operation Mode: Current

Current 9.94 mA

Figure 5: Screenshot of the main window of the Lens Driver software with the current control.

The drive signal can be chosen to be a DC, sinusoidal, rectangular or triangular signal with the possibility to set the upper and the lower signal level as well as the driving frequency. This is indicated in Figure 6.



Lens Temperature [°C]: 22.875

Lens Features:

Operation Mode: Triangular

Frequency: 0.001 Hz

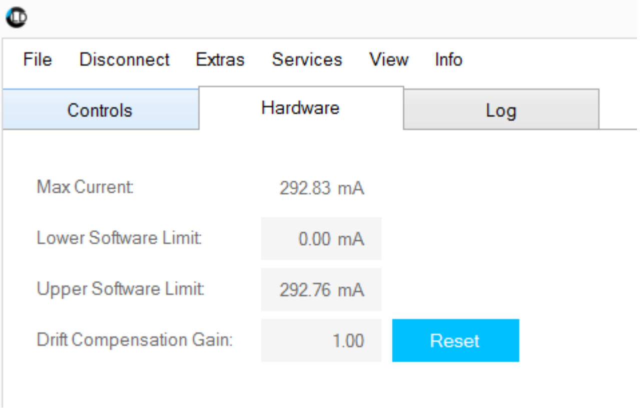
Lower Signal Level: 0.00 mA

Upper Signal Level: 0.00 mA

Figure 6: The drive signal can be chosen to be a DC, sinusoidal, rectangular or triangular signal.

Limiting the maximum current

The limits for the current are set in the **Hardware** tab, shown in Figure 7.



Max Current 292.83 mA

Lower Software Limit 0.00 mA

Upper Software Limit 292.76 mA

Drift Compensation Gain: 1.00

Figure 7: Setting the current limits.

The maximum current of the driver is limited to either 290mA, see Figure 8, or 2.8V divided by the resistance applied.

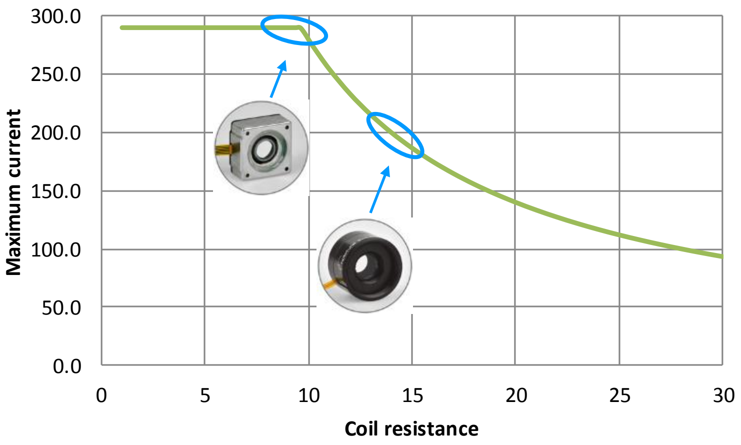


Figure 8: Dependence of maximum current on coil resistance.

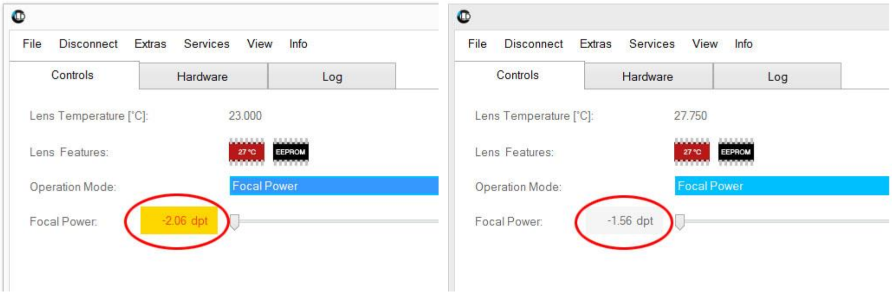
Temperature compensation

When heating up the lens, the fluid expands in volume and therefore, the focal length of the lens decreases. The focal length decreases linearly by approximately 0.67 diopters per 10°C temperature increase. This temperature effect is systematic and reproducible and is therefore accurately compensated via a temperature sensor SE97B with an I^{2}C sensor read-out, integrated in the lens (not in the compact EL-10-30). This allows controlling the focal power directly.

In **Focal Power mode**, see drop down menu in Figure 6, temperature independent lens operation is ensured. Depending on the present temperature, the current applied to the lens is adjusted for compensation of the temperature drift. A look-up table with the calibration data for the temperature compensation are stored directly on the EEPROM of each individual lens. With the temperature compensation enabled, the absolute reproducibility achieved over an operating temperature range of 10 to 50°C amounts to typically 0.1 diopters.

Achievable Range of Focal Power

If the focal power is outside the allowed diopter range then a warning message will appear in yellow “warning focal power outside guaranteed range” and the panel has a yellow background, see Figure 9. The user can then change the focal power until the displayed focal power no longer has a yellow color and is within the diopter range.



Lens Temperature [°C]: 23.000
Lens Features:
Operation Mode: Focal Power
Focal Power: -2.06 dpt

Lens Temperature [°C]: 27.750
Lens Features:
Operation Mode: Focal Power
Focal Power: -1.56 dpt

Figure 9: On the left, the focal power is outside the achievable range and the panel is yellow. On the right, the focal power is within the range.

The limits for the range of focal power are determined by the temperature limits and the maximum and minimum current. The maximum diopter limit is located at lowest encoded temperature limit and maximum current and the minimum diopter limit is located at the highest encoded temperature and the minimum current. This is explained in Figure 10. The highest and lowest temperature, indicated by the gray and green lines, are adjustable in **File → Options** under **Temperature Settings**. The closer together the minimum and maximum values are the larger is the achievable range in focal power.

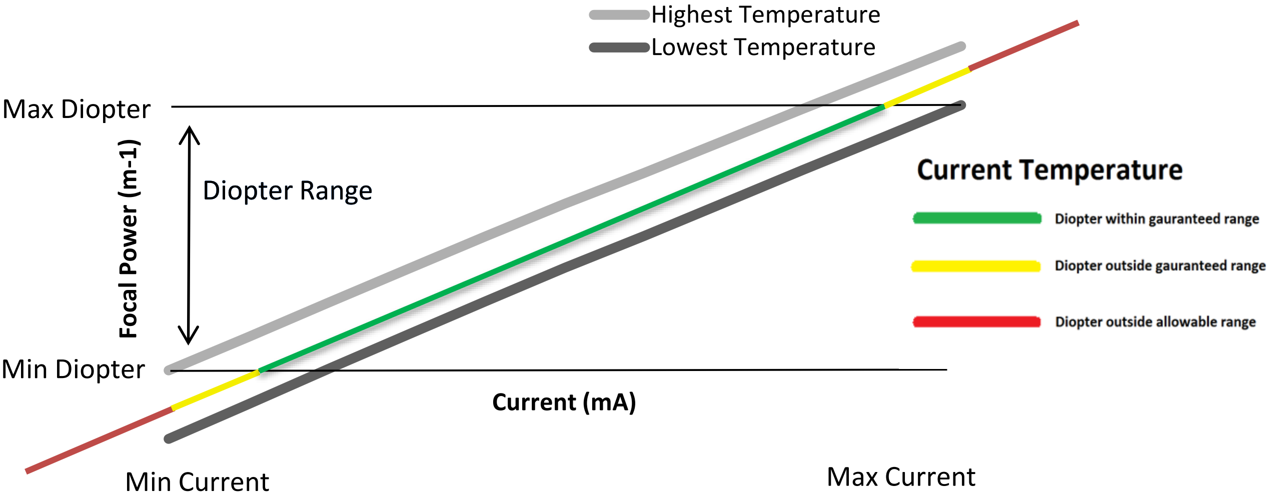


Figure 10: Illustration of the achievable range of focal power (vertical axis) versus current (horizontal axis). The achievable range also depends on the highest and lowest temperature, indicated by the gray and green lines.

Drift Compensation

When set to a certain focal power the lens shows a tiny drift over timescales of about 5min. If a highly stable focus position over long times is required, using the implemented drift compensation is recommended. To do so the *Drift Compensation Gain*, found under **Hardware** (see Figure 7) has to be set to a finite value, typically 1. Higher values increase the gain up to a maximum value of 5.

When a finite drift compensation is set it also scales the achievable diopter range. The maximum drift compensation is added to the minimum diopter limit, so that that diopter range is deceased (as seen in Figure 11). The maximum drift compensation is determined by the gain variable and a larger gain variable results in a larger value for maximum drift compensation.

For lenses with a small focal range the effect of increasing the gain variable to high values (e.g. 3 or higher) can be that the achievable diopter range is so small that focal power cannot anymore be changed by the user. In that case the gain variable has to be reduced.

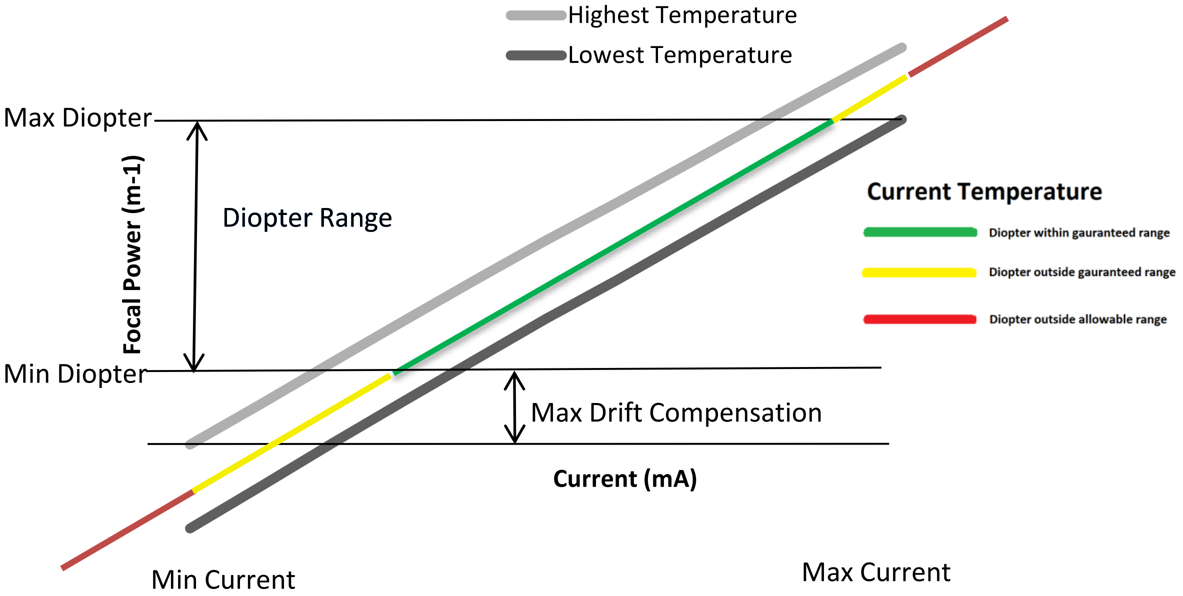


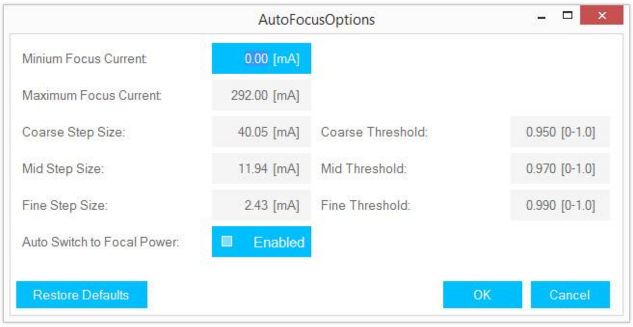
Figure 11: Illustration how a finite drift compensation affects the achievable range of focal power.

Autofocus

When using the Lens Driver in combination with a uEye camera, it also offers an autofocus feature. Please note that this feature is for testing purposes only and Optotune does not offer support for this part of the software. The autofocus software is only tested yet with the camera model *IDS UI-3580CP-C-HQ*. To use the autofocus, go to ***Extras → uEye Viewer***. The uEye viewer window will open.

Settings:

In *Lens driver → Auto Focus Settings,* the settings can be adjusted to optimize the autofocusing. The parameter shown in Figure 12 are standard values that work for most applications. In order to optimize e.g. the time it takes to autofocus one may set the thresholds for coarse, mid and fine scanning closer to 1, but the values have to be Coarse < Mid < Fine. The check box to enable auto switch to focal power acts only if a lens with stored calibration data on the lens's EEPROM is used. Camera settings like frame rate and exposure time can be changed in *Camera → Settings.* Autofocus only works reliably if the image is bright enough.



Minium Focus Current	0.00 [mA]		
Maximum Focus Current	292.00 [mA]		
Coarse Step Size:	40.05 [mA]	Coarse Threshold:	0.950 [0-1.0]
Mid Step Size:	11.94 [mA]	Mid Threshold:	0.970 [0-1.0]
Fine Step Size:	2.43 [mA]	Fine Threshold:	0.990 [0-1.0]
Auto Switch to Focal Power:	☐ Enabled		

Figure 12: Settings for the Auto Focus option in combination with a uEye camera.

For autofocus, click on the image in an area that contains structures with reasonable contrast. The current applied to the lens is then automatically adjusted and the image is in focus, as in *Figure 13*.

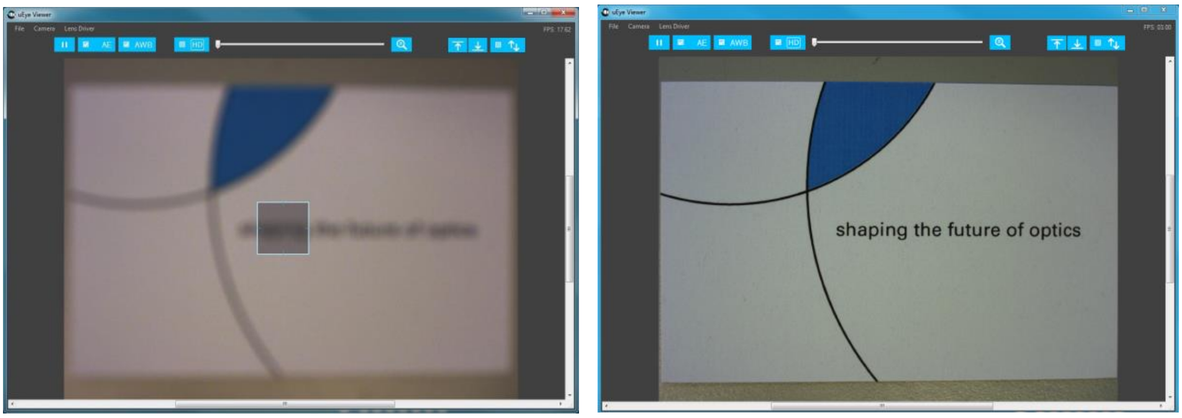
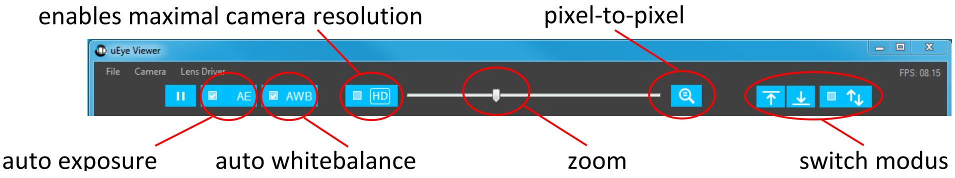


Figure 13: Autofocus is achieved by clicking on the image

Options:



Pixel-to-pixel modus: in the 'pixel-to-pixel' modus, the resolution is chosen in the way that one pixel of the image corresponds to one pixel of the screen.

Switch modus: In this modus, the driver switches between two current values with a rectangular driving signal. The lower and upper current level can be determined in the following way. First perform autofocusing for the lower level (click on the part of the image which should be in focus) and afterwards click on the lower limit button (arrow with an underline). Repeat the steps for the upper limit (arrow with an over line). If you than enable the button with the two arrows, the current will switch between the two values. The frequency is changed in the *Controls* window.

Trigger output signal

For sinusoidal, triangular and rectangular drive signals, hardware pin A (see Figure 14) outputs a trigger signal which can be used to synchronize the lens driver to another hardware component. The synchronization output signal toggles between 0 V and 5 V and provides a maximum of 15 mA. Please note that the pin "*Analog In A*" has been re-configured from an analog input to a digital output (trigger output).

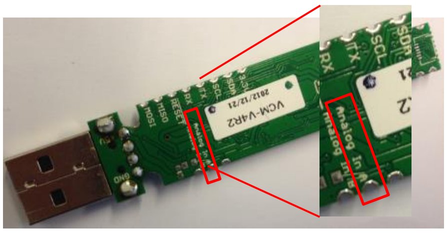


Figure 14: Hardware pin A for trigger output signal.

The relation between the trigger output signal and the corresponding output signal of the Lens Driver for different drive signals are shown in Figure 15:

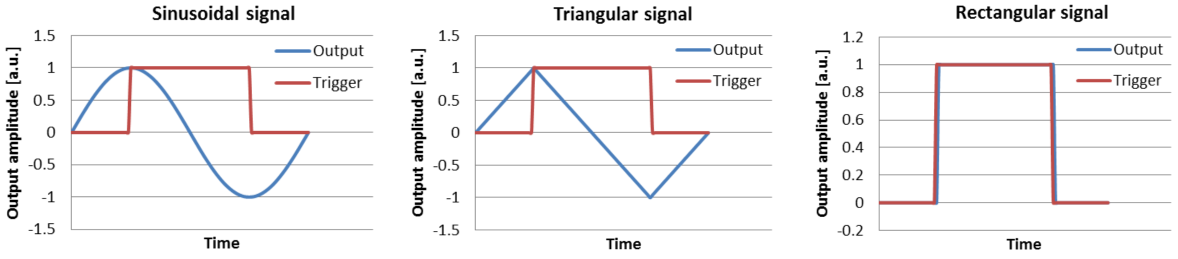


Figure 15: Relation between trigger and output signal for different drive signals.

Analog Signal Reading

Analog signal reading is built into the firmware, hardware pin B (see Figure 16). The analog voltage of the input signal must be between 0 V and 5 V. The ADC has a resolution of 10 bit and therefore the digital signal lies between 0 and 1023.

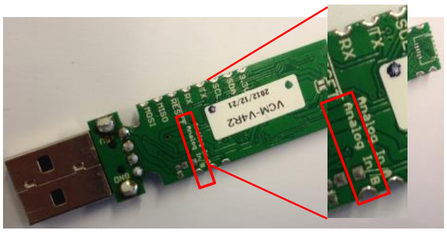
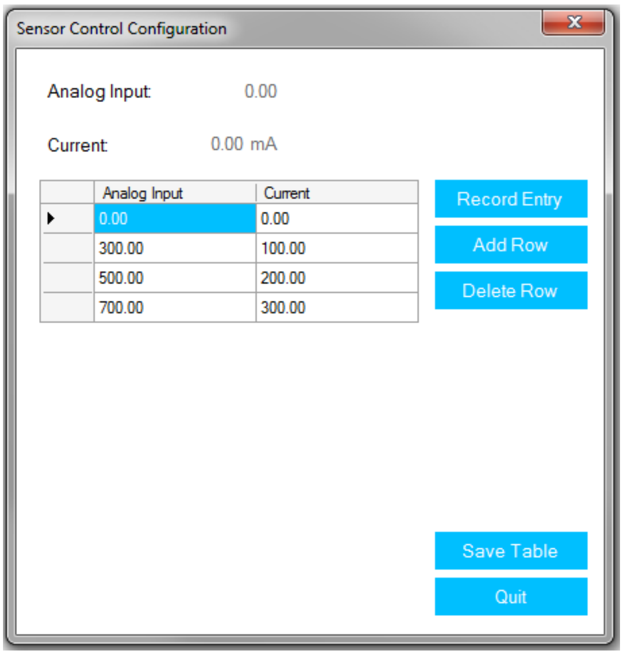


Figure 16: Hardware pin B for analog input signal.

Sensor Control

Through an analog signal that is read by the lens driver, an external voltage from an analog sensor can be used to control the focal power of the lens. Sensor control can be used in either **Current** or **Focal Power** mode, and is enabled under **Services -> Sensor Control**.

To enable this functionality, a set of calibration points must first be inputted. A calibration point consists of an analog input value set by a sensor and a desired current/focal power value. If sensor control is enabled, the software will linearly interpolate between the configured points, and set the current/focal power according to the analog input. It is important to note that sensor control will only be enabled if at least two calibration points have been defined. Calibration points can be stored using the sensor control configuration form found under **Extras -> Sensor Control Configuration**. The points are stored in a modifiable table, and are saved in the software for future use. The screenshot below represents the sensor configuration form in **Current** mode.



Analog Input 0.00

Current 0.00 mA

Analog Input	Current
0.00	0.00
300.00	100.00
500.00	200.00
700.00	300.00

Button Functions:

- **Record Entry:** Store the latest analog input and current/focal power value as a calibration point.
- **Add Row**: Create an empty row in the configuration table to manually input a calibration point.
- **Delete Row:** Every row with a highlighted cell will be removed from the table.
- **Save Table:** Save the configuration table. Note that sensor control will only work if at least two calibration points have been saved.
- **Quit:** Exit the sensor control configuration form. Users will be prompted to save any unsaved data.

Communication Protocol

This section describes the protocol used to control the lens driver in case customized software is written. The protocol can be implemented using any programming language. An example for the CRC calculation is only given in C.

The documentation shows commands sent out by user software (i.e. PC) and the answer to be expected as a response from the driver as well as the action that is taken by the driver as a result of the command. Commands in quotes (“) have to be sent as a string (or char array) with **each letter representing one byte** except for “\r” meaning “carriage return”, “\n” meaning “new line” and are represented by a single byte (ASCII code). Letters in red are sent as ASCII characters and letters in blue are coded as binary bytes.

While this section describes the protocol for the main commands used to control the lens driver, detailed information on other commands can be found in the command protocol file which is available as a **download on our website.**

Connection:

The Microcontroller used runs a virtual com port driver provided by Atmel. The connection settings are:

- Baudrate: 115200 (others may also work since the port is virtual)
- Parity: None
- Stop Bits: One

Handshake:

The handshake command is used to check if the hardware is ready and running. It can also be used as a reset function as it will reset the current to zero. Other commands are also valid without sending this command after initialization, it is optional.

PC sends	Driver answer	Driver action	Comment
"Start"	"Ready\r\n"	reset current to zero, flush input buffer	optional command

Current set commands

A current set command sets a new output current for a channel. A current set command is constructed as AwxxLH. It always starts with the channel identifier "A" for channel A[1], and the write identifier char “w”. The current sent out (i_o) is coded as a 12bit value (x_i) that is mapped to the maximum hardware current range (i_c). The maximum hardware current range (calibration value i_c) can be read and written from the driver, see section about calibration commands. By default, i_c = 293mA. The formula to calculate the value x_i for a certain current is:

$x_i = i_o / i_c * 4096$

Example for a current of 50mA for a calibration value of 293mA:
50mA / 293mA * 4096 = 699

Commands to set current are sent as one char selecting the channel, followed by two bytes containing the 16bit signed current value x_i [-4096 to 4096], followed by two bytes for the CRC16 checksum which is calculated over all the preceding data bytes (see CRC16 code section on how to implement it) with the low byte of the CRC sent first: “LH” means ‘**L**ow Byte’, ‘**H**igh Byte’. If a value is out of boundaries (i.e. 5000) it is limited to 4096 by the firmware, no overflow will occur. If the software current limit is set lower than 4096 (i.e. 3000, see calibration commands), current requests bigger than the software limit will be reduced down to the limit (i.e. 3000). The driver does not reply to correctly received current commands to make the possible update rate as fast as possible. Invalid commands (i.e. incorrect CRC) will be answered with “N\r\n”.

Char coding for the current set command:

"A"	Channel A
“w”	Write Identifier

---

[1] Currently, the Lens Driver only hosts one channel. In future, a second channel might be added to control two lenses simultaneously.

PC sends	**Driver answer**	**Driver action**	**Comment**
"AwxxLH"	none	set channel A current to xx	xx is a signed 16bit integer with a value between -4096 and 4096, high byte sent first
"AwxxLf"	"N\r\n"	none	f = error in CRC byte, this line applies for all detected CRC errors.

Exemplary current set command

Command type: Set current
Channel: A
Current value: 1202

Resulting command:
Byte 0: 0x41 (corresponds to ASCII "A")
Byte 1: 0x77 (corresponds to ASCII "w")
Byte 2: 0x04 (high current byte) / 0b00000100
Byte 3: 0xb2 (low current byte) / 0b10110010 -> with byte 2 the total current bits are 0b0000010010110010 which equals to 1202 dec
Byte 4: 0x26 (low CRC byte)
Byte 5: 0x93 (high CRC byte)

For verification of the command, the CRC checksum (see details below) over the whole 6 bytes of the command can be calculated which should be zero.

Focal Power set command

A focal power set command sets the focal power of the lens. The command starts with the prefix “P”, followed by the write identifier “w”, the coding char “D”, and the channel identifier “A”. The command then includes 2 bytes for the desired focal power value and 2 dummy bytes. Focal power is encoded with the following formula to have an effective range of -5 to 15.48 diopters:

$$x_i = (fp + 5) * 200$$

Example for a focal power of 5 diopters:
(5 + 5) * 200 = 2000

Similar to current set commands, the command is finished by a 16bit CRC calculated from the eight command bytes. The low byte of the CRC is sent first. It is important to note that this command will only work in controlled mode (see Mode Commands below).

Char coding for the focal power set command:

"P"	Prefix
“w”	Write Identifier
“D”	Coding
"A"	Channel A

PC sends	**Driver answer**	**Driver action**	**Comment**
"PwDAxxYYLH"	none	set focal power of lens through channel A	xx is a signed 16bit integer with a value between 0 and 4096, high byte sent first. YY are 16 dummy bits.
" PwDAxxYYLf "	"N\r\n"	none	f = error in CRC byte, this line applies for all detected CRC errors.

Exemplary focal power set command

Command type: Set focal power
Channel: A
Focal power value: 2000 (5 diopters)

Resulting command:
Byte 0: 0x50 (corresponds to ASCII "P")
Byte 1: 0x77 (corresponds to ASCII "w")
Byte 2: 0x44 (corresponds to ASCII "D")
Byte 3: 0x41 (corresponds to ASCII "A")
Byte 4: 0x07 (high focal power byte) / 0b00000111
Byte 5: 0xd0 (low focal power byte) / 0b11010000 -> with byte 1 the total current bits are 0b 0000011111010000 which equals to 2000 dec
Byte 6: 0x00 (required dummy byte)
Byte 7: 0x00 (required dummy byte)
Byte 8: 0x31 (low CRC byte)
Byte 9: 0xfd (high CRC byte)

Mode commands

Mode commands allow to access frequency modes (see table). A mode change command always starts with "M", followed by a "w" to write the command. Reading commands back is by sending an “r” instead of a “w” is possible but not tested and recommended. The “w” is followed by a char identifier that selects the mode type (“S”, “Q”, “T”,”C” or “D”, see table) and finally a char selecting the channel (“A”). The command is finished by a 16bit CRC calculated from the four command bytes. The low byte of the CRC is sent first.

To set the property of the selected mode (i.e. frequency and currents) a signal property change command is used. This command starts with a “P” followed by a “w” to write the command. Again, reading commands back is by sending an “r” instead of a “w” is possible but not tested and recommended. The “w” is followed by a char identifier that selects the property to be changed (“U”, “L” or “F”, see table). Next a char selecting the channel (“A”) is sent. The next four bytes sent represent the data for the command. It is a 32-bit unsigned integer for the frequency or a 16bit signed integer followed by two dummy bytes for the current. The frequency needs to be multiplied by 1000 (fixed comma representation). Example: For a frequency of 12Hz the integer sent out needs to be 12000 (0x00002EE0 as a 32bit hex). The bytes to be sent out (in this order) are: 0x00, 0x00, 0x2E, 0xE0. The command is finished by a 16bit CRC calculated from the eight command bytes. The low byte of the CRC is sent first.

The mode “C”, or Controlled Mode, allows the driver to maintain the focal power of a connected lens. This mode must be enabled in order to use focal power set commands.

Char coding for the mode change command:

"A"	Channel A
“w”	Write Identifier
"S"	Sinusoidal signal
"Q"	Square signal
"D"	DC signal
"T"	Triangular signal
“C”	Controlled Mode

Char coding for the signal property change command:

"U"	upper swing current [-4095 to 4095]
"L"	lower swing current [-4095 to 4095]

"F"	Frequency (in 'value in [mHz] = value in [Hz] *1000')

Mode command examples:

PC sends	Driver answer	Driver action
"MwSALH"	"MSALH\r\n"	set channel A to sinusoidal waveform
"MwCALH"	"MCALH\r\n"	set channel A to controlled mode
„PwFAyyyyLH"	nothing	set channel A frequency to yyyy (in mHz, 32bit value)
„PwUAyyddLH"	nothing	set channel A upper signal current to yy (12 bit value) (dd are two dummy stuffing bytes and can be 0)

Calibration commands

Calibration commands are used to set and read calibration values and software current limits. These values are stored in the EEPROM, a non-volatile memory, and are kept there for years.

Software current limits can be set to protect a lens from overcurrent or to fix the maximum focal power. Limits also apply in analog input mode.

> **Note**: in analog input mode the input voltage 0-5V is mapped to [-4095 to 4095] in a 10bit resolution. Software current limits apply.

A calibration command always starts with "C" followed by "r" to road or "w" to write the command value. The next byte determines the type of value. It can be "M" for the maximum current calibration value (in [mA*100]), "U" for the upper software current limit or "L" for the lower software current limit. Next a char selecting the channel ("A") is sent followed by two data bytes (16-bit integer). The maximum current calibration value is the output current measured for the x_i = 4095 value and does normally not need to be changed as design guarantees 1% accuracy to the default value. The default value is 29284 (292.84mA).The software current limits are sent as a 12bit current value and can also be negative like a normal current command. As usual the command is finished by a 16bit CRC calculated over the seven command bytes.

> **Note:** the upper and lower limits are always saved to EEPROM that allows "only" 100'000 write cycles before it starts wearing out. There is no limit on reading.

Char coding for the Calibration command:

"A"	Channel A
"M"	Maximum hardware current [mA]*100 (calibration value)
"U"	Upper software current limit [-4095 to 4095]
"L"	Lower software current limit [-4095 to 4095]

Calibration command examples:

PC sends	Driver answer	Driver action	Comment
"CrMAddLH"	"CMAyyLH\r\n"	none	read channel A calibration, "dd" are two dummy bytes and can be 0
"CwLAxxLH"	"CLAxxLH\r\n"	write xx as new lower limit of channel A	limit is immediately active, answer is for double checking
"CrUAddLH"	"CUAyyLH\r\n"	none	read channel A upper current limit, "dd" are two dummy bytes and can be 0

Temperature reading

If the lens connected supports temperature reading, this command will read back the temperature of the integrated sensor (NXP SE97B). To convert the value to a temperature the following formula can be used:

Temperature [°C] = data * 0.0625 [°C]

For more details about the conversion please refer to the datasheet of the temperature sensor.

The temperature read command starts with a "T" followed by the channel selection byte ("A"). The command is completed with the 16bit CRC calculated over the two command bytes. Low CRC byte is sent first.

Possible temperature commands:

PC sends	Driver answer	Driver action	Comment
"TALH"	"TAEddLH\r\n"	Get temperature reading of lens on channel A	E is 0x00 if read is successful, 0xff otherwise. "dd" is the temperature data.

CRC algorithm

A 16-bit CRC checksum (CRC-16-IBM) is used to check for communication errors. The code examples below use the reverse polynomial implementation.

- Reverse polynomial: 0xA001
- Initial value to be used: 0x0000

Checking for communication error

Checking for a communication error is done by calculating the CRC checksum over the whole command data array which includes two CRC bytes at the end that were added from the sender. CRC checksum calculation over the whole array results in a CRC checksum equal to zero if no data corruption is present.

The following example shows the *checkcrc* implementation in the Lens Driver firmware, which is written in C:

```
/* Example usage of the CRC function */
// Example:    Set current level to 1202 command sent from PC (includes CRC bytes)
//             With this data array the returned uint16_t crc checksum will be zero
//             (0x00) since no communication error has occurred.
//             uint8_t data[] = { 0x41, 0x77, 0x04, 0xb2, 0x26, 0x93 };

uint16_t calculatecrc(void)
{
        uint16_t crc = 0, i;
        for (i = 0; i < sizeof(data) / sizeof(data[0]); i++)
        {
                crc = crc16_update(crc, data[i]);
        }
        return crc; //returns checksum over all elements
}


uint16_t crc16_update(uint16_t crc, uint8_t a)
{
        int i;
        crc ^= a;
        for (i = 0; i < 8; ++i)
        {
                if (crc & 1)
                crc = (crc >> 1) ^ 0xA001;
                else
                crc = (crc >> 1);
        }
        return crc;
}
```

Adding a CRC checksum to a data array

In order to add a CRC checksum to a data array

1. Calculate CRC checksum over data array
2. Append the calculated CRC checksum at the end of the data array

   Example:
   `uint8_t dataarray[3] = {data byte 0, data byte 1, data byte 2}`

```
uint8_t dataarraywithcrc[5] = { data byte 0, data byte 1, data byte 2, crc&0xFF,
crc>>8}
```

If this is done correctly, a consecutive CRC checksum calculation over the whole array needs to result in a CRC checksum equal to zero (see method *AddAndCheckCRC*).

The following example written in c# shows the implementation in the Lens Driver Controller software:

```
private byte[] AddAndCheckCRC(byte[] command)
{
        UInt16 CRC = 0;

        byte[] commandWithCRC = new byte[command.Length + 2];

        CRC16IBM CRC16IBMCalculator = new CRC16IBM();
        CRC = CRC16IBMCalculator.ComputeChecksum(command);

        Array.Copy(command, 0, commandWithCRC, 0, command.Length);

        commandWithCRC[commandWithCRC.Length - 2] = (byte)(CRC&0xFF);
        commandWithCRC[commandWithCRC.Length - 1] = (byte)(CRC >> 8);

        //Check if calculation is correct (should equal 0)
        check = CRC16IBMCalculator.ComputeChecksum(commandWithCRC);

        return commandWithCRC;
}

public class CRC16IBM
{
        const ushort polynomial = 0xA001;
        ushort[] table = new ushort[256];

        public ushort ComputeChecksum(byte[] bytes)
        {
                ushort crc = 0; // initial CRC value
                for (int i = 0; i < bytes.Length; ++i)
                {
                        byte index = (byte)(crc ^ bytes[i]);
                        crc = (ushort)((crc >> 8) ^ table[index]);
                }
                return crc;
        }

        public byte[] ComputeChecksumBytes(byte[] bytes)
        {
                ushort crc = ComputeChecksum(bytes);
                return BitConverter.GetBytes(crc);
        }

        public CRC16IBM()
        {
                ushort value;
                ushort temp;
                for(ushort i = 0; i < table.Length; ++i) {
                        value = 0;
                        temp = i;
                        for(byte j = 0; j < 8; ++j) {
                                if(((value ^ temp) & 0x0001) != 0) {
                                        value = (ushort)((value >> 1) ^ polynomial);
                                }else {
                                        value >>= 1;
                                }
                                temp >>= 1;
                        }
                        table[i] = value;
                }
        }
}
```