Software Manual

XELA Sensor Server
Linux, Windows and ROS
v1.6.0

Table of contents

General Limitations for using the Sensors

- Applying too high forces or pressures will destroy the sensor module and will void the warranty. Keep the forces (applied to the whole surface of the Sensor Module) and pressures (depending on your contact area) below those values:

  z-axis (forces applied normal to the sensor's surface) limits: 10N/25kPa

  x-axis and y-axis: the sum of the shear forces has to stay below 6N/15kPa

  For example, if you contact only half of the Sensor Module's surface, only apply 5N normal force.

  Furthermore, apply forces only to the sensor surface, not to the sides of the sensor module.

- As a skin sensor, the contact geometry always influences the measurements. Therefore, we do not calibrate the sensors, as any calibration would be valid only for the same contact shape. We suggest that you use machine learning techniques or your algorithms to get various information out of the sensor measurements, relevant for your application.

- As our sensor uses magnetic field changes induced by the skin deformation as its sensing principle, other magnetic fields (including the earth magnetic field), nearby magnets, or nearby ferromagnetic materials can influence the sensor measurements. Also crosstalk between two of our sensor modules is possible. Please confirm within the inspection period (1 month from receipt of the product) with your application if those influences are prohibitory for your application. To counteract those influences, please also consider that a reference sensor could be used.

- Never bend the sensor modules. When you install the sensor modules on your robot, glue them to a sturdy and flat surface with thin double-sided sticky tape. Make sure to provide flat support to the whole backside of the sensor module

General Limitations of the software

- The software is in constant development and therefore is provided AS IS
- The code is compiled and tested on 1 PC and therefore might have bugs related to different hardware and software configurations. We do our best to iron them out before publishing.
- There might be code-breaking changes between the releases. You might need to adjust your code to get it working again. Please check the change log for details before installing the new version!

Installation

You can unpack Windows and Linux executables to the directory of your liking
ROS needs to be in catkin build directory (See more under Getting ready)
Before making new configuration, you can test the compatibility of the apps on your system by running `./xela_server -f xServ.ini` and `./xela_viz -f xServ.ini` as since 1.5.0 the package will include simulation configuration (for testing computer and SocketIO communication)

Upgrade

Before replacing your Windows or Linux executables, try to run them from temporary folder to check the compatibility of the apps on your system (See Installation for more details)

Major changes

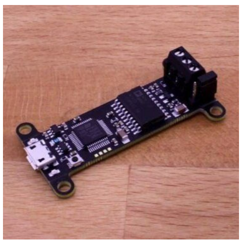
1.6.0a (build 33102)	Change in core functionality. All sensors now require a definition of Sensor model. The `sensor_height`, `sensor_width`, `stype` and `mode` are no longer used. All standard sensor configuration will be included in the binaries and there will be support for extra configurations when we make special sensors. Configuration files will have .xeladef extension and can be in the same folder with the executable on all OSs.
1.5.1rc (build 25905)	Addition of extra logging mechanisms and automatic folder generation for log files
1.5.1b (build 25310)	Fix some major issues with Windows where Visualization wouldn't work and the app would crash. (Intel CPUs) Fixed problem where simulation sensors above 1 would not show properly.
1.5.0a (build 23801)	Fix of memory leak in SDL2 (should fix also some random crashes on Windows)
1.5.0a (build 22701)	Release of Raspberry Pi server (server-only module) with automatic IP setting for public use
1.5.0rc (build 21201)	Visualization uses SDL2 instead of MatPlotLib and has a significant performance boost. Linux has 3 releases for Ubuntu 16.04, 18.04 and 20.04. All binaries are 64 bit and WILL NOT work on 32 bit machines All sensor orientations should be correct and rotations work on server side only

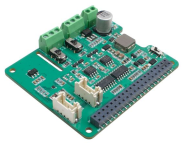
Warning
Version 1.6.0 will use new config file format
Version 1.4.0 is incompatible with older releases
Version 1.4.0 will not support old style config files (pre 1.2.0 files)
Version 1.4.0b has major change in data transfer
Version 1.4.1b is very strict with config files

Using the Server

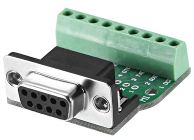
Compatible CAN converters

Note: The following list includes only devices we have tested on. Other devices might also work.

Converter	Compatibility	Pros	Cons
 Esd CAN-USB/2	Windows (esd) Linux (socketcan)	Sturdy design High speed	Works very differently in Windows and requires special drivers
 PEAK USB-CAN (PCAN)	Windows (pcan) Linux (pcan)	Cheap Small size	Requires special drivers in all OSs Can be difficult to set up in Linux with UEFI
 VScom USB-CAN Plus	Linux (socketcan) Raspberry Pi 4 (socketcan)	Mountable Sturdy design	Will not work in Windows Uses Serial-over-USB communication which is bit slower than standard CAN bus
 CANable (Pro)	Linux (socketcan)	Small size Micro-USB	No DB9 connector Needs “candlelight” firmware in Linux

Converter	Compatibility	Pros	Cons
 2-Channel CAN-BUS(FD) Shield for Raspberry Pi	Raspberry Pi 4 (socketcan)	Has 2x CAN bus	Requires compilation of kernel drivers No DB9 connector
 Waveshare RS485 CAN HAT	Raspberry Pi 4 (socketcan)	Has RS485 and CAN bus	No DB9 connector

If using CAN converter without DB9 connector, you can buy DB9 breakout board online



Connect cables between converter and DB9 breakout board as follows:

```
CAN_H → Pin 7
CAN_L → Pin 2
GND   → Pin 3
```

Warning

Even though other CAN devices might work, misconnection can break the sensor.

Please always use CAN devices with standardized DB-9 connector

OS

- Windows 10 (64bit)
- Ubuntu 16.04, 18.04 and 20.04 (64bit)
- ROS (requires ros-bridge-suite since 1.6.0)
- Raspberry Pi OS 10 (32bit) - server only, no extra tools, Raspberry Pi 4 minimum

Prerequisites

For connecting to XELA Server with Python, you would need to install following Python packages:

1. python-socketio
2. asyncio (requires Python 3.7+)
3. aiohttp

For other languages similar libraries would be required

Warning
Please do not forget to give sensors sufficient power. External USB power supply is recommended.

Visualization can be very CPU-heavy, please use a decent PC for running the applications or consider running visualization on another computer.

Set up

Desktop and Laptop

Activating CAN bus

Linux and ROS only
If you haven't activated CAN bus yet, please run the following commands:
Note: Make sure you have can-utils installed first

1. Esd or any other CAN device:
   user@localhost:~$ sudo ip link set can0 type can bitrate 1000000
   user@localhost:~$ sudo ip link set up can0
   Explanation: First we will set up CAN network on can0 with bitrate of 1Mbit/s, then we pull it up
2. Any slcan device:
   user@localhost:~$ sudo slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0
   user@localhost:~$ sudo ifconfig slcan0 up
   Explanation: First we set up SLCAN device running on /dev/ttyUSB0 with hardware bitrate of 3Mbit/s and CAN speed of 1Mbit/s (-s8), then we will pull it up

NOTE: replace can0/slcan0 with the corresponding CAN bus name and ttyUSB0 with the correct device name

Automating activation

As every CAN-USB converter acts differently, we cannot cover all of them, but the following example works for VScom USB-CAN Plus and can be used as guidance for other devices.

Step 1.
Make a new udev rule for your device type (slcan in this example) with
`sudo nano /etc/udev/rules.d/90-slcan.rules`
`Use of higher rule number is recommended in case of errors`

Add the following content:

```
#slcan autoloader

ACTION=="add", ATTRS{interface}=="VScom USB-CAN Plus",
ENV{SUBSYSTEM}=="tty", RUN+="/usr/bin/logger [udev] SLCAN device detected -
running slcan_add.sh!", RUN+="/usr/local/bin/slcan_add.sh $kernel"

ACTION=="remove", RUN+="/usr/bin/logger [udev] SLCAN device removed",
RUN+="/usr/local/bin/slcan_remove.sh $kernel"
```

Make sure that you remove the line breaks (each ACTION command needs to be in it's own line).

Step 2.
Make script to execute on device connection:

```
#!/bin/sh
# Bind the SLCAN device
slcand -o -c -f -s8 -t hw -S 3000000 /dev/$1
sleep 2
ifconfig slcan0 up
```

Step 3.
Make script to execute on device removal:

```
#!/bin/sh
# Remove the SLCAN device
trig=$1
if [ "$trig" = "ttyUSB0" ] || [ "$trig" = "ttyUSB1" ] ; then
 pkill slcand
fi
```

Note: You can write a grep function in the if, when it would be more helpful

Step 4.
Make both scripts executable with:
`sudo chmod +x /usr/local/bin/slcan_*`

Step 5.
Reload rules with:
`sudo udevadm control --reload-rules`

Step 6.
Test and adjust whenever needed

Installing drivers

Windows

For esd install the esd drivers (from esd.eu). The Python specific drivers are included in the executables.

Linux

Depending on the distribution, there might be a need for PCAN drivers.
Please download them from https://www.peak-system.com/fileadmin/media/linux/index.htm
If your system is using Secure Boot, you might need to sign the kernel modules. A good article about this is available at https://superuser.com/questions/1438279/how-to-sign-a-kernel-module-ubuntu-18-04.
Instead of `vboxdrv`, you have to use `pcan`.

Application Installation

Windows

Unpack the downloaded files to your preferred folder
Keep all configuration files in the same folder

Linux

Unpack the downloaded files to your preferred directory
Make /etc/xela directory and give it 777 permissions
Keep all configuration files in /etc/xela directory

ROS (Linux)

Unpack the downloaded files to your catkin workspace src directory and build as normal
Make /etc/xela directory and give it 777 permissions
Keep all configuration files in /etc/xela directory
Note: ROS comes without default configuration file

Getting ready

1. Unpack the files to a suitable location.
   In case of ROS, to your catkin workspace folder
2. Run the configuration (first use or when configuration has changed) or make it manually
   NOTE: Only to be used with 4x4 and 4x6 sensors, otherwise please make configuration file manually or consult with XELA Robotics
   NOTE: If XELA Robotics made you a configuration file, please honor that over this application
   Windows PowerShell: ./xela_conf.exe
   Windows cmd: xela_conf.exe
   Linux: ./xela_conf
   ROS: rosrun xela_server xela_conf
3. Run the server (mandatory)
   The server will send the data via SocketIO. See arguments section to see all configuration options.
   In case of ROS, you can use a launch file that activates everything.
   Windows PowerShell: ./xela_server.exe
   Windows cmd: xela_server.exe
   Linux: ./xela_server
   ROS: rosrun xela_server xela_server
4. Run additional tools
   • Visualizer for showing the sensor status on screen as dots (size and location based on data)
   Windows PowerShell: ./xela_viz.exe
   Windows cmd: xela_viz.exe
   Linux: ./xela_viz
   ROS: rosrun xela_server xela_viz

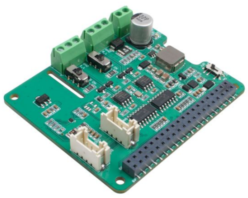
   • Logger to log sensor data in raw format with UNIX timestamps into LOG folder as CSV files (all sensors will have separate filename in format “sens-[sensor number].csv”. Please see arguments section to see more detailed information about using it
   Windows PowerShell: ./xela_log.exe
   Windows cmd: xela_log.exe
   Linux: ./xela_log
   ROS: rosrun xela_server xela_log

   • ROS Messaging service (Only on ROS) More info at ROS Messages
   ROS: rosrun xela_server xela_service
   Alternatively you can run the launch file for xela_server and xela_service together: roslaunch xela_server service.launch

Raspberry Pi

Note: Will work only on Raspberry Pi 4B and is tested on 4 Gb RAM version running Raspian 10 (Raspberry Pi OS)

Compatible HATs

	Seeed Studio 2-Channel CAN-BUS(FD) Shield for Raspberry Pi Pros: 2 CAN channels
	WaveShare RS485 CAN HAT

Activation of the CAN bus

Application Installation

Applications can be installed to any suitable directory and /etc/xela folder should be made with 777 permissions and your xServ.ini file should be placed there.

Limitations of Raspberry Pi version

Raspberry Pi version has only the server module and will require another computer to be present for visualization and logging. There is currently no automatic configuration tool either.

Using

Arguments

Following is the list of arguments that can be given to the executables (green – used, red – not used):
In Windows it might need .exe at the end of executable name.
In ROS you can add arguments like usual (rosrun xela_server xela_server -h)

-h Show all argument options for the script with short description
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_server -h

-f *filename* The name of configuration file (with .ini extension) to be used (full path or just name if in same folder) (default: */etc/xela/xServ.ini* (Linux) or *xServ.ini* (Windows)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_server -f myconf.ini

--ip *ip* The IP address for the computer running the server (if same computer, it would be *127.0.0.1* aka *localhost*)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_viz --ip 192.168.0.107

-p *port* The port for communicating the server (default: *5000*)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_viz -p 5007

--viz *sensor_number* The sensor number to display in visualization (defaults to 1)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_viz --viz 3

-s *sensor_config* The string for configuring logging script (default: *1:100*)
  more info in How to use XELA Logger
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_log -s 1-3:1000

-lf *path_to_log_folder* Folder, where csv files will be saved (default: *CSV/* in Windows or */etx/xela/CSV/* in Linux)
  more info in How to use XELA Logger
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_log -lf MyLogs

--log *log_file* The path to log file (default: */etc/xela/LOG/app.log* in Linux or *LOG/app.log* in Windows where app corresponds to app name)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_server --log my_server_log.log

-d *device* The can device name (default: *socketcan* in Linux or *esd* in Windows)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_conf -d pcan

-c *channel* The can device channel (default: *can0* in Linux or *0* in Windows)
  Usage: [xela_conf, xela_log, xela_server, xela_viz]
  Example: ./xela_conf -c PCAN_USBBUS1

~~--config *name* The name of special configuration (**do not use unless XELA has advised to do so**)~~

~~Usage: [xela_conf, xela_log, xela_server, xela_viz]~~

~~Example: ./xela_conf --config allegro~~ **will be removed from 1.6.1**

--hw Run the visualization on GPU (for dedicated GPUs only)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_viz --hw

Please see ROS Launch arguments for more detailed information when using Launch files.

How to use XELA Logger

Under the -s flag (stands for sensor config), you can write the info in following formats that can be combined:

Data	Meaning
1	Sensor* 1 will be read (default 100 broadcast cycles*)
1,2,3	Sensors 1, 2 and 3 will be read (default 100 broadcast cycles)
1-3	Sensors 1,2 and 3 will be read (default 100 broadcast cycles) (shortcut)
1:150	Sensor 1 will be read 150 broadcast cycles
1:150,2	Sensor 1 will be read 150 broadcast cycles and sensor 2 for default 100
1-3:150	Sensors 1, 2 and 3 will all be read for 150 broadcast cycles
1-5,2:150	Sensor 2 will be read 150 broadcast cycles and sensors 1, 3, 4 and 5 for default 100
1:60t	Sensor 1 will be read for 60 seconds
1-3:30t	Sensors 1, 2 and 3 will all be read for 30 seconds
1-5,2:30t	Sensor 2 will be read for 30 seconds and sensors 1, 3, 4 and 5 for default 100 broadcast cycles
1-5:200,3:30t	Sensor 3 will be read for 30 seconds and sensors 1, 2, 3 and 5 will all be read for 200 broadcast cycles

Example:

	A	B	C	D	E	F	G
1	time	1X	1Y	1Z	2X	2Y	2Z
2	1587029110.17588	16858	18758	35645	15128	15545	31502
3	1587029110.17615	16858	18758	35645	15128	15545	31502
4	1587029110.27562	19188	16018	35189	17276	15273	32799
5	1587029110.2757	19188	16018	35189	17276	15273	32799
6	1587029110.37499	13737	14693	34553	18699	16100	32949
7	1587029110.37511	13737	14693	34553	18699	16100	32949

Note

*When defining argument for -s flag, ensure that there are NO spaces in it (after a comma or a colon)
*If sensor is defined multiple times, latest entry counts
*Sensors can be from 1 to 16
*Time is recorded as a **Unix timestamp** (on all OSs).
The **Unix epoch** (or **Unix time** or **POSIX time** or **Unix timestamp**) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally speaking the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for Unix time. Some systems store epoch dates as a signed 32-bit integer, which might cause problems on January 19, 2038 (known as the Year 2038 problem or Y2038).
*Broadcast cycles depend on the sensor and computer configuration and is defined as sensor read cycle and backup update cycle combined

ROS

NOTE: For version 1.5.1 (for ROS Noetic) via SocketIO, from 1.6.0 ROS Kinetic onwards through ros-bridge

ROS Service

For single set of data, use one of the following service calls:

Service call	**Example output**
user@localhost:~$ rosservice call /xServXY 1 2 Get X and Y from taxel 2 on sensor 1	values: [16439, 16647]
user@localhost:~$ rosservice call /xServXYZ 2 6 Get X, Y and Z from taxel 6 on sensor 2	values: [16451, 16517, 35901]
user@localhost:~$ rosservice call /xServX 2 1 Get X from taxel 1 on sensor 2	value: 16681
user@localhost:~$ rosservice call /xServY 2 2 Get Y from taxel 2 on sensor 2	value: 16721
user@localhost:~$ rosservice call /xServZ 2 3 Get Z from taxel 3 on sensor 3	value: 37009
user@localhost:~$ rosservice call /xServStream 1 Get full sensor data from sensor 1	xyz: [1: [16457, 16553, 32057], 2: [16775, 16958, 31886]…]

ROS Service types

Service name	**Service message type**	**Description**
xServX	XelaSensorX	Get x coordinate for taxel
xServY	XelaSensorY	Get y coordinate for taxel
xServZ	XelaSensorZ	Get z coordinate for taxel
xServXY	XelaSensorXY	Get x and y coordinates for taxel
xServXYZ	XelaSensorXYZ	Get x, y and z coordinates for taxel
xServStream	XelaSensorStream	Get x,y and z coordinates for the sensor (all taxels on sensor)

ROS Topic

Topic name	Topic message type	Description
/xServTopic	xela_server/XStream	Get all sensors and taxels as a constant stream

Example output from topic:

```
user@localhost:~$ rostopic echo /xServTopic
sensors: 2
data:
  -
    sensor: 1
    taxels: 2
    xyz:
      -
        x: 16480
        y: 18265
        z: 34586
      -
        x: 14807
        y: 14999
        z: 35922
  -
    sensor: 2
    taxels: 4
    xyz:
      -
        x: 15695
        y: 17062
        z: 31000
      -
        x: 13514
        y: 15846
        z: 30999
      -
        x: 13869
        y: 14119
        z: 30066
      -
        x: 15575
        y: 13317
        z: 37027
---
```

ROS Launch arguments

To make it easier to run server separately from ROS or with different settings

Argument	Description
noserver:=1	Will not run xela_server (requires external launch of the app on same or different computer)
port:=5000	Define xela_server port, default is 5000
ip:=127.0.0.1	Define xela_server IP, default is 127.0.0.1
rbport:=9090	Define ros_bridge port, default is 9090
rbip:=localhost	Define ros_bridge IP, default is localhost
file:=/etc/xela/xServ.ini	Define configuration file for xela_server, default is /etc/xela/xServ.ini
d:=0	Define debug level, default 0, max 3. Prints info based on requests made via ROS 1 - Request type only 2 - Request type and sensor/taxel info 3 - Request type, sensor/taxel info and returned values

Example Code for ROS

```python
#!/usr/bin/env python

import rospy
from xela_sensors.srv import XelaSensorXYZ
import sys

rospy.init_node('use_service')

#wait the service to be advertised, otherwise the service use will fail
rospy.wait_for_service('xServXYZ')

#setup a local proxy for the service (we will ask for X,Y and Z data)
srv=rospy.ServiceProxy('xServXYZ', XelaSensorXYZ)

#use the service and send it a value.
#In this case, I am sending sensor: 1 and taxel: 3
service_example=srv(1, 3)

#print the result from the service
print(service_example)

#close the app
sys.exit(0)
```

Configuration file format

We recommend using XELA_Conf tool for making configuration file. If you need to adjust it or make a custom one, please refer to following format:

```
[CAN]
bustype = socketcan *1
channel = can0 *2
[viz]
max_offset = 40 *3
max_size = 50 *4
[sensor]
num_brd = 2 *5
refreshrate = 100 *6
ctr_ver = 2 *7
model = XR1944 *8
ctrl_id = 6 *9
channel = 0 *10
format = raw *11
[sensor2]
ctrl_id = 7
model = XR1922
channel = 0
rotate = 2 *12
force_act = 1 *13
```

Sections in config file:

[CAN] – defines section for CAN bus
[viz] – defines settings for visualization
[sensor] – defines primary sensor settings and number of total boards
[sensor*N*] – (*N* is a number above 2, sequential) – defines Nth sensors settings, if different from primary

Notes:

*1 Bus types for Linux are socketcan, slcan and pcan and for Windows esd, pcan
*2 Channel for **pcan** is *PCAN_USBBUS1*, for **socketcan** *can0* and for **esd** *0*
*3 Defines how far the bubbles will go in visualization
*4 Defines how big the pressure bubble will be
*5 Defines the total number of boards
*6 Sets refresh rate limit on server broadcasting
*7 Defines the controller version (optional, default: 2, meaning revision 2)
*8 Defines the sensor model
*9 Defines controller ID
*10 Defines channel (default 0, in multi-mode (2 XR1944 sensors on one controller) use 2 for second one)
*11 Tells the server to broadcast unmodified taxel information based on the directions shown in hardware manual (optional, if not set, will modify all axis to be same across different sensors)
*12 Defines for server to rotate the sensor by 90 degrees times specified number before outputting the data (optional)
*13 Requests the server to send activation command to the sensor (optional, for some special controllers only)

Explanations for sections

[CAN]

bustype	esd – Windows only for esd CAN-USB/2 pcan – PEAK CAN devices (both Windows and Linux) socketcan – Linux only for several standardized CAN devices sim – Simulated server (data is random, just to check connectivity)
channel	0 – In windows while using esd CAN-USB/2, we need to specify the channel number PCAN_USBBUS1 – PCAN bus name, numbers can increment, if using more than 1 CAN device on single computer can0 – Linux CAN bus name, numbers can increment, if using more than 1 CAN device on single computer slcan0 – Linux Serial based CAN bus name, numbers can increment, if using more than 1 CAN device on single computer

[debug]

Note: this section is only required for troubleshooting and therefore should be used only when necessary

sens_print	full – show all sensor config visually in terminal with taxel arrangement minimal – show only specifics about sensors (ID, Size and Type) (default) none – do not show any info about sensors in terminal

[viz]

max_offset	A number between 1 and 1000 to show the sensitivity of sensor on XELA visualizer for x and y axis
max_size	A number between 1 and 1000 to show the sensitivity of sensor on XELA visualizer for z axis (pressure)

[sensor]

num_of_sensor	A number representing the count of sensors on the CAN bus (if 2 sensors are connected to single controller, they are still considered as separate sensors)
~~sensor _height~~	~~A number representing the height of default sensor (default: 4 rows)~~ Removed since 1.6.0
~~sensor _width~~	~~A number representing the width of default sensor (default: 4 columns)~~ Removed since 1.6.0
ctrl_id	A number representing the controller ID of default sensor (0...15)
~~mode~~	~~A number representing the type and style of sensor-controller config:~~ ~~4 – for 4x6, 2x2, 2x1 and 1x1 sensors, and older 4x4 sensors with 4 channels~~ ~~5 – for 4x4 sensors with 2 channels~~ ~~0 – for old controller with 4x6 sensors~~ ~~1 – for old controller with 4x4 sensors with 4 channels~~ ~~2 – for old controller with 4x4 sensors with 2 channels~~ Removed since 1.6.0
channel	The number of starting channel If using 2 4x4 sensors on single controller, second sensor (port B on splitter cable) has to have `channel=2`
~~config~~	~~The special configuration options that are available:~~ ~~Allegro - it will run our default configuration of the sensor setup and line them up the right way~~ ~~(more to come in the future)~~ Will be removed in 1.6.1 and therefore shouldn't be used
refreshrate	Refresh rate speed limit. Default is up to 100 Hz, but it can be limited. Must be integer between 1 and 1000 (do not go over 100 unless you have a monster computer as it can cause delays and errors)
model	The model number of the sensor. Default ones are: XR1844 - old 4 channel 16 taxel sensor XR1944 - new 2 channel 16 taxel sensor XR1911 - 1x1 sensor XR1921 - 2x1 sensor XR1922 - 2x2 sensor XR1946 - 4x6 sensor
ctr_ver	Controller version. By default (if not specified) it will be 2, which is the current one. Version 1 was supplied at the beginning only.

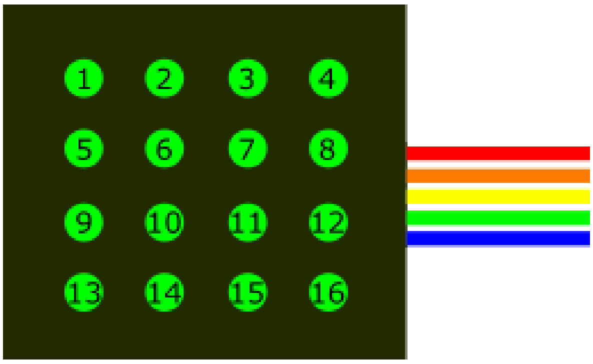
force_act	Used to state to the server that the sensor needs activation before use like old controllers, but is in new controller configuration
format	modified - (default), all axis will be set by the server to be same across all sensors raw - all axis as shown in hardware manual, can cause confusion when using several different sensors

[sensor*N*]

N – sensor's number (sequential) (ex. 2 for the second sensor on CAN bus). Used to specify settings that are different from defaults. Range is between 1 and 16
Settings that can be used, are following:
model, **ctr_ver**, **ctrl_id**, **force_act**, **channel** and **format**
Please see section **[sensor]** for details

Data format

Sensor Layout



Sensor data is read from top left towards right, line-by-line.
If 2 4x4 sensors are connected to the same controller, they are treated as 2 different sensors by the server (default behaviour, special configurations will be possible at request).
To determine the layout for the sensor, place it in front of you with cables extruding from the right as per the image on the left side.
Note: 4x6 will have 2 extra columns, meaning that the first row is 1 – 6 instead of 1 – 4.

Code

As the data is provided in a JSON string, there are few ways to access it and it has following format:

{'data':'FA00,FA00,FA00,FB00,FB00,FB00...' , 'sensor': '1','taxels':'16'}

data contains X, Y and Z data for all taxels (sensing points) in order, converted into HEX values (without leading '0x') as text

sensor has the number of the sensor in string format (important when requesting multiple sensors from server)

taxels has the number of the taxels meant to be sent (for confirmation only)

Note: sensor is considered up-right if the cables exit on the right (attachment) side

Note: if using ROS service, all data will be integers. See ROS->Use for more info

Example usage

Code

```python
#!/usr/bin/env python3

import asyncio
import socketio
loop = asyncio.get_event_loop()
sio = socketio.AsyncClient()

@sio.event
async def connect():
  print("Connection established")

@sio.event
async def sensor_data(data):
  #normally the taxel data is in data element as list type string
  print(data[u"data"])

@sio.event
async def disconnect():
  print("Disconnected from server")

async def start_server():
  """
  using loop makes the application wait for connection
  in case server takes too long to start
  """
  ncd = True
  while ncd:
    try:
      await sio.connect("http://localhost:5000", namespaces=["/sensor1"])
    except socketio.exceptions.ConnectionError:
      pass
    else:
      ncd = False
      break
  try:#make sure this is outside of while
    await sio.wait()
  except KeyboardInterrupt:
    exit()
#run server
loop.run_until_complete(start_server())
```

Situation examples

Running server and Visualizer on different machines

Situation:

You have bought 8 sensors (XR1944) and just connected to your Ubuntu machine. You have run the configuration tool and everything is working. When you run Visualizer, it is very slow and suddenly the server is throwing async errors.

Resolution:

In this case it shows that the computer CPU is not able to process data quickly enough and is causing bottlenecks. It would be advisable to run the server on this machine with --ip flag stating local IP address (not 127.0.0.1, but rather something like 192.168.0.103) and use another computer (Windows or Linux) to run Visualizer from there with the same IP flag. It reduces the stress on the CPU and makes it available for the server and other mission-critical applications. See [CAN] and Arguments sections for more info.

Server having async errors without Visualizer

Situation:

You have bought 16 sensors (XR1944) and just connected to your PC. You have run the configuration tool and everything is working, but it is still causing async errors.

Resolution:

It looks that the CPU is not fast enough. By default the server will relay sensor info up to 100 Hz (depending on sensor configuration) and on older machines, it can result in timeouts. The best way to tackle this is to set refresh-rate limits in the configuration file under [sensor] section by adding a line refreshrate = 20. It will shorten the queue of the SocketIO messages.

It is also advisable to keep your receiving app ready for new data at all times (discarding the info if it is unable to process it in time, is recommended).

Common errors

Error	Reason
Module not found	Use pip to install the mentioned module and try again. If the module is already installed, try reinstalling. On systems with several Python versions, you might need to specify which pip you are using
Could not start CAN: OSError: [Errno 19] No such device	Make sure to pull up the network with one of the following commands, if using Linux: user@localhost:~$ sudo ip link set can0 type can bitrate 1000000 user@localhost:~$ sudo ip link set up can0 or user@localhost:~$ sudo slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0 user@localhost:~$ sudo ifconfig slcan0 up In case of Windows, make sure the adapter is connected and using specified channel (esd channel might not be 0)
The MATPLOTLIBDATA environment variable was deprecated in Matplotlib 3.1 and will be removed in 3.3	This can be ignored as it is caused by incompatibility between PyInstaller and MatPlotLib Might be present in some 1.4.0 builds
Program not responding to CTRL+C	There are few functions that have disabled interrupt transfer during compiling. To exit, use pkill -9 xela_server (or whichever function is not responding)
Unable to register with master node [http://localhost:11311]: master may not be running yet. Will keep trying	Node couldn't communicate with the ROS master node. Make sure it is running.
Error connecting to CAN: IOError:[Errno 19] No such device	No CAN device found. Make sure your CANUSB device is connected, accessible for all users (pulled up) and set in the configuration correctly (see /etc/xela/xServ.ini or xServ.ini)
Error writing config file: IOError: [Errno 2] No such file or directory: '/etc/xela/xServ.ini'	Ensure there is /etc/xela folder (in Linux, or executable folder in Windows) and that it has 777 permissions

xela_viz is not starting and showing errors with xkcd	Your version of matplotlib might be outdated as it doesn't support the colors we use. Please update it and try again Might be present in some 1.4.0 builds
ROS service call is showing a message: "did you run 'make' on…"	If you are using environments, make sure the terminal window used for service call is using the same environment. I.e. run `source <your_catkin_folder>/devel/setup.bash`
Unable to install Python 3.7 on Ubuntu 20.04 for ROS	To install different Python versions on Ubuntu, use deadsnakes repository: `sudo add-apt-repository ppa:deadsnakes/ppa`
I see warning for Yaml deprecation when using ROS	ROS might use some outdated elements and it might be incompatible with newer Python packages. You might need to downgrade Yaml to a version prior to the change until ROS team has fixed their code

Permission denied when connecting to MBED	Add your user to dialout group:
	`sudo useradd $USER dialout`
	or for temporary use:
	`sudo chmod a+rw /dev/ttyACM0`

Warning: Tax N out of range when using MBED	It means that MBED has sent out data containing values that are abnormal. Repower the MBED to see if it fixes the issue, if not, please contact us.
My catkin_tools are giving invalid syntax error on Noetic	There has been an issue with the Python package of catkin_tools for Noetic and the workaround has been to install it from git: `python3 -m pip install git+https://github.com/catkin/catkin_tools.git`

If you find errors, not listed in this file, please send an email regarding it to info@xelarobotics.com
Do not forget to attach files from /etc/xela/LOG folder in Linux or LOG folder in Windows (if using default log folder) with the terminal log (if you copy-paste, there might be some strange characters, don't worry about them, we can read them) and description of the failure. If you have made your own code that is causing errors, please attach it as well.