BLK ARC Module API

Release 0.9.0

Dec 15, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Welcome to the documentation for the BLK ARC Module API, a Python package designed to provide an easy-to-use interface for controlling and communicating with BLK ARC devices using gRPC.

With the BLK ARC Module API, integrators can easily interact with the BLK ARC by writing Python code that sends gRPC requests to the device, and receives responses back. The user will have the full control of BLK ARC, from data acquisition to download of scan data (.b2g file). This data can be used for multiple purposes, such as reality capture and autonomous navigation tasks.

The BLK ARC Module API package provides a high-level API that abstracts away many of the complexities of working with gRPC, making it easy for integrators to get started quickly. The package also includes extensive documentation and examples to help integrators understand the various features and capabilities of the BLK ARC.

We hope you find this documentation helpful and informative. If you have any questions or feedback, please don't hesitate to reach out to us.

> LEICA GEOSYSTEMS WILL ONLY COMMIT TO OFFICIALLY RELEASED APIs. EXAMPLES ARE PROVIDED AS-IS. LEICA GEOSYSTEMS IS NOT LIABLE FOR ANY DAMAGES CAUSED BY USING THE EXAMPLES OR API.

CHAPTER

TWO

ABOUT THE BLK ARC

2.1 Overview

The BLK ARC is an autonomous laser scanning module based on the GrandSLAM technology, that combines multiple sensors in order to collect spatial data and define the position of the scanner relative to the space around it.

Specification	Value
Dimensions	Height: 183.6 mm / Width: 80 mm / Length: 92.5 mm
Net weight	690 g
Storage	24 hours of scanning (compressed data) / 6 hours (uncompressed data)
Lidar	830 nm wavelength with a range of 0.5m to 25m and relative accuracy of 6-15 mm
Point measurement rate	420.000 points / second
HighRes camera	12 MPixel, rolling shutter
Panoramic vision system	3-camera system, 4.8 Mpixel 360° x 125°, global shutter
Communication	USB 2.0 and Wi-Fi

2.1.1 Environmental

- Robustness - Designed for indoor and outdoor use
- Operating temperature 0 to +40 °C
- Dust & humidity protection IP54 (IEC 60529)

Note: Fan/Ventilation might require to be exchanged after a certain time. The measurement performance of the BLK ARC module can degrade over time in highly dusty environments due to its lack of airtightness (which depends on factors such as dust particle size and number of operation hours).

2.1.2 Network communication

Connectivity	
Wi-Fi	802.11gn
USB Ethernet	100 MBit/s

The device is reachable as a network device via USB. The IP is statically set to 192.168.42.1. DHCP is available.

The device runs in either of two wireless network modes:

- **BLK ARC as a Wi-Fi access point (AP)**. Applications with physical proximity to the BLK ARC, like the Leica Geosystems provided mobile application BLK ARC Live, can connect to the WiFi access point and communicate directly without any networking infrastructure. The AP password is printed on the "Connection Settings" card that you got together with your BLK ARC. The device is reachable on the static IP 10.1.1.1 if not configured otherwise.
- **BLK ARC as a Wi-Fi client**. BLK ARC can join an existing Wi-Fi network, and applications can also join the same Wi-Fi network to talk to the device. This approach increases the possible range between application and the BLK ARC and allows to use the 5Ghz frequencies for faster data transfer.

2.2 Power Requirements

- Supply power range: 9 V to 48 V
- Maximum power: 25 W

Please ensure that the power supply you use for the BLK ARC falls within the specified voltage range and meets the power requirement. Using a power supply outside of these limits or could cause damage to the device or result in unsafe operating conditions.

2.2.1 Connectors

- USB connector socket type C
- Power socket 5.5x2.5 DC Jack

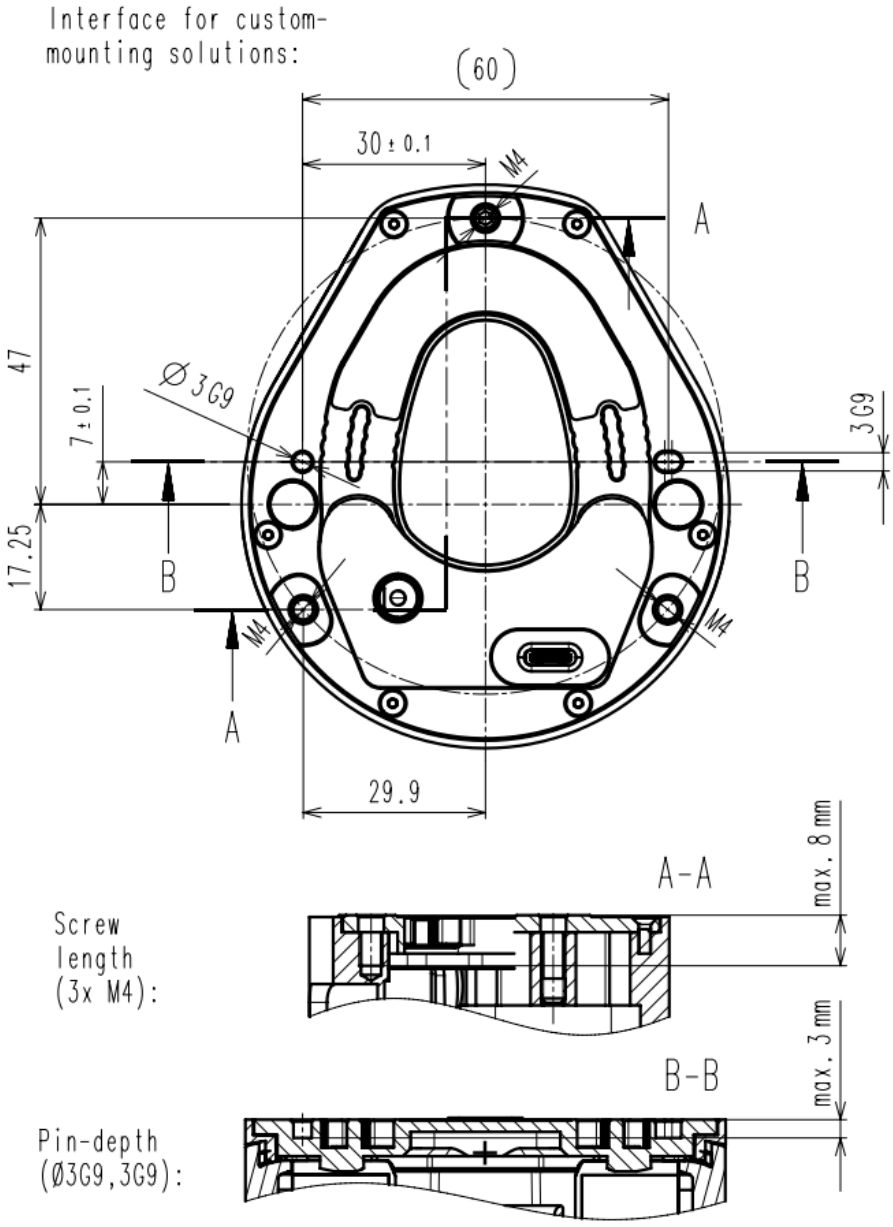
Note: Two (power) modes of operation are available:

1. Data reading only mode – BLK ARC module is powered over USB-C only (5V, 0.5A/2.5W max). In this mode it's not possible to scan – only to download data.
2. Regular operation mode – All functionalities available.

To start the BLK ARC module, a proper USB connection is required. If power is supplied before start, the regular operation mode will be entered. If only a USB connection is provided, the BLK ARC will enter the Data reading only mode. Once in Data reading only mode, if a supply voltage is subsequently connected, initiating a reboot gRPC command will transition it back to the regular operation mode.

2.3 Mounting Interface

The BLK ARC mounting interface is provided at the bed-plate, by three M4 srew-holes. Two additional pin-holes (one regular and one elongated hole) serve as an accurate reference if required.



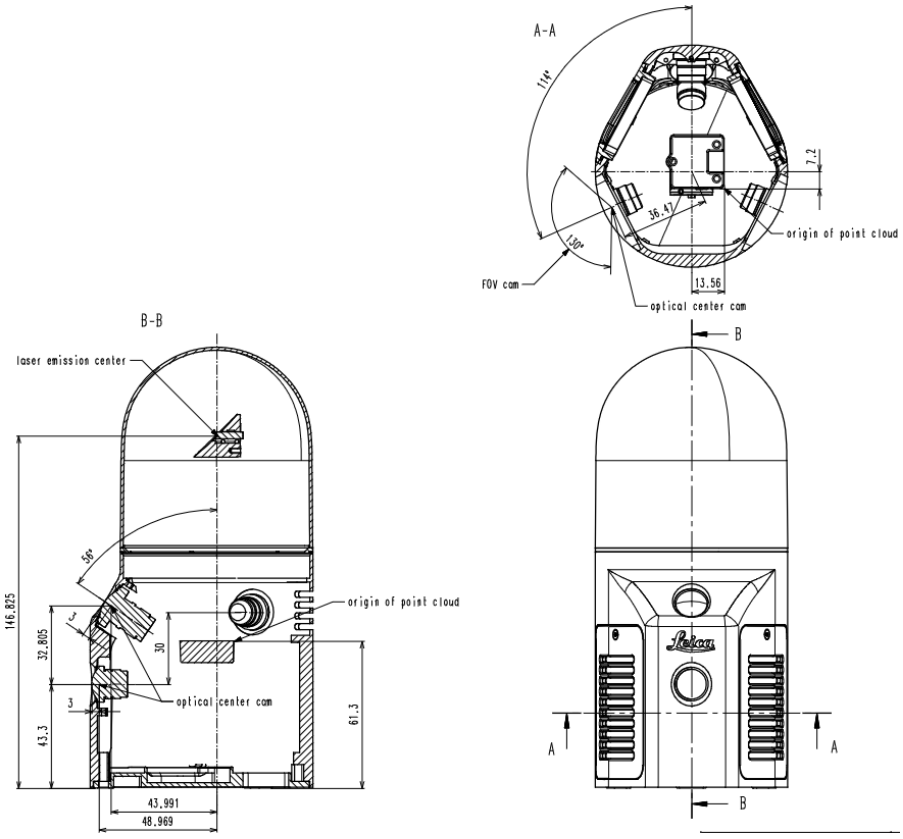
2.4 Field of View

The provided LiDAR FoV is 360 degrees horizontal, and 270 degrees vertical. The laser emiting source is centered at the spherical protection dome.

The three panoramic cameras are arranged in a landscape 360 degree arrangement, providing a vertical FoV of about 100 degrees each. The detail image provides a resolution of 12 MPixel with a horizontal FOE of 90 degress and a vertical FOV of 120 degrees.

Note: the actual camera center and FoV slightly varies due to manufacturing tolerances, but each camera is precisely calibrated and accounted for.

Please find the positions of the cameras in the drawing below:



CHAPTER

THREE

INSTALLATION

3.1 Requirements

The BLK ARC Module API requires BLK ARC **FW 4.0.0** and newer and is only supported on Ubuntu 20.04 and newer.

To use the BLK ARC Module API, you need the following requirements installed on your system:

- Python 3.8
- pip package manager
- virtualenv (pip install virtualenv)

The required python packages are listed within the **requirements.txt** file and are automatically installed when running the setup_environment.sh bash script.

3.2 Setup & Installation

To set up the environment, run the following command in a terminal:

```
source scripts/setup_environment.sh
```

This will create a virtual environment and installs the necessary dependencies for the BLK ARC Module API. Once the installation is done, the virtual environment will be activated automatically, and the BLK ARC Module API can be used.

If the `setup_environment.sh` script has been excecuted once before, the virtual environment can be activated by running:

```
source .venv/bin/activate
```

We recommend using virtual environments to manage dependencies and avoid conflicts with other Python projects that may have been installed on your system.

To show the details of the installed package and to verify that the installation was successfull, use the following command:

```
pip show blk_arc_sample_wrapper
```

3.3 Testing the Code

In order to test the BLK ARC Module API we use the PyTest framework (see requirements.txt).

The pytests can be excecuted by running the following command:

pytest ./src/python_sample_wrapper/pytests/

CHAPTER

FOUR

API DOCUMENTATION

4.1 BLK ARC Module API

class blk_arc_sample_wrapper.blk_arc.BLK_ARC

Bases: object

A simple API-layer to BLK ARC devices using the gRPC-interface.

connect(*connection_type=ConnectionType.WIRED*)

Connects to a BLK ARC device.

Parameters
connection_type (config.ConnectionType) – Connection method of the BLK ARC (Wired or wireless).

Returns
True if a connection could be established, else False.

Return type
bool

is_connected()

Checks whether a device is currently connected.

Returns
True if a device is currently connected, else False.

Return type
bool

disconnect()

Disconnects from a BLK ARC device.

Returns
True if the device could be disconnected, else False.

Return type
bool

get_device_info()

Reads the name, the serial-number and the article-number from the device. See the proto definition for further documentation.

Returns
Proto-message containing the device-information or None if the call failed.

Return type
Optional[about_message.ManufactureInfoResponse]

get_firmware_version()

Reads the firmware-version from the device.

Returns
Firmware version or None if the call failed.

Return type
Optional[str]

get_device_status()

Reads the status from the device. See the proto definition for further documentation.

Returns
Proto-message containing the device-state or None if the call failed.

Return type
Optional[device_message.DeviceStateResponse]

get_free_disk_space_percentage()

Reads the free disk-space from the device.

Returns
Free disk-space percentage or None if the call failed.

Return type
Optional[float]

start_capture()

Starts capturing a new scan.

Returns
Proto-message containing the scan-id, starting time and a uuid of the started scan or None if the call failed.

Return type
Optional[capture_message.StartCaptureResponse]

stop_capture()

Stops the currently running scan.

Returns
Proto-message containing the scan-id, starting time, duration, a uuid and some other statistics of the finished scan or None if the call failed.

Return type
Optional[capture_message.StopCaptureResponse]

begin_static_pose()

Allows you to mark the start of a static pose while scanning. This enables you to process static-parts of the scan separately, which can result in improved accuraccy of the data.

Returns
True if the start of a static scan could be marked in the running scan, else False.

Return type
bool

end_static_pose()

Allows you to mark the end of a static pose while scanning. See 'begin_static_pose()' for more details.

Returns
True if the end of the static scan could be marked in the running scan, else False.

Return type
bool

is_scanning()

Returns whether a scan is ongoing.

Returns
True if the call worked and a scan is ongoing, else False.

Return type
bool

is_in_static_pose()

Returns whether a static-scan is ongoing. Static-scans are started/stopped with 'begin_static_pose()' and 'end_static_pose()'.

Returns
True if the call worked and a static-scan is ongoing, else False.

Return type
bool

start_scan_data_stream(*max_point_frequency=10000*, *enable_low_latency_trajectory=False*)

Starts the streaming of scan-data. Scan-data contain the trajectory of the scanner and the pointcloud in scanner-frame.

Parameters

- max_point_frequency (int) – Upper limit for the number of pointcloud-points which should be streamed. The maximum number you will be able to actually receive depends on various factors like the connection or the efficiency of the code receiving the points. The number of received points will always be smaller than this limit due to point-filtering on the scanner.
- enable_low_latency_trajectory (bool) – Enables trajectory-output with lower latency compared to the slightly delayed SLAM-trajectory output. A lower latency is achieved by integration of IMU-data, thus also reducing the accuracy of the signal as compared to the usual SLAM-based output.

Returns
A generator streaming the scan-data or None if the call failed.

Return type
Optional[Generator[capture_message.CaptureStreamMessage, None, None]]

list_scans()

Lists metadata of all the scans which are stored on the scanner.

Returns
A list of metadata of all the scans on the scanner or None if the call failed.

Return type
Optional[List[scan_library_message.ItemInfo]]

get_scan_info(*scan_id*)

Returns the metadata of a specific scan identified by its scan-id.

Parameters
scan_id (int) – The id of the scan.

Returns
Metadata of the specified scan or None if the call failed.

Return type
Optional[scan_library_message.ItemInfo]

delete_scan(*scan_id*)

Deletes a scan from the scanner, identified by its scan-id.

Parameters
scan_id (int) – The id of the scan which should be deleted.

Returns
True if the scan could be successfully deleted, else False.

Return type
bool

download_scan(*scan_id*, *path*)

Downloads a scan from the scanner to the device calling this method. The scan is identified by the scan-id.

Parameters

- scan_id (int) – The id of the scan which should be deleted.
- path (Path) – The path to the directory where the scan should be downloaded to.

Returns
True if the scan could be successfully downloaded, else False.

Return type
bool

trigger_detail_image()

Triggers the capture of an image with the detailed-camera. Capturing a detail-image is only possible while a scan is ongoing. The image is stored in the scan (.b2g).

Returns
True if the capture of the detail-image could be triggered, else False.

Return type
bool

stream_pano_images_scanning(*count=1*)

Starts the streaming of panoramic images during a scan.

Parameters
count (int) – Number of pano image triplets (Left, Front & Right camera) to be streamed.

Returns

A generator streaming the panoramic
images or None if the call failed.

Return type
Optional[Generator[imaging_message.ImageStreamResponse, None, None]]

stream_pano_images_idle(*camera_id*, *count=1*, *exposure_in_ms=-1*, *gain=-1*)

Starts the streaming of panoramic images while no scan is on-going.

Parameters

- camera_id (imaging_message.PanoramaCameraStreamRequest.CameraID) – The ID of the camera of which the images should be streamed.
- count (int) – Number of images to be streamed.
- exposure_in_ms (int) – The exposure is the time the shutter stays open, expressed in milliseconds. A higher exposure value will generate a brighter image.
- gain (int) – The gain is the equivalent of the ISO value in a camera. It artificially increases the brightness of the image without requiring a longer exposure.

Returns

A generator streaming the panoramic
images or None if the call failed.

Return type
Optional[Generator[imaging_message.ImageStreamResponse, None, None]]

reboot()

Triggers a reboot of the scanner. The reboot will take a while and you will loose conntection to the device. Therefore you must reconnect using 'connect()' once the scanner is running again.

Returns
True if the reboot was triggered and the connection was closed, else False.

Return type
bool

power_status()

Reads information about the power-source from the scanner. This allows to determine whether the scanner is in USB-only mode (no scanning allowed) or properly connected to a power-supply.

Returns
A proto-message containing information about the power-source or None if the call failed.

Return type
Optional[power_message.PowerStatusResponse]

data_only_mode_active()

Returns whether the scanner is in USB-only mode (only connected by USB-cable). In this mode you can still interact with the scanner and download scans. However scanning is not possible.

Returns
True if the call succeeded and the scanner is in data-only-mode, else False.

Return type
bool

get_time_from_clock_sources()

Reads the different clock-sources from the scanner. This can be used in order to establish time-synchronisation.

Returns
A proto-message containing the time of all clock-sources or None if the call failed.

Return type
Optional[clock_message.TimestampsResponse]

acknowledge_faults()

Some less serious faults on the scanner are resolvable by acknowledgement. This call allows you to acknowledge all the active and acknowledgeable faults on the scanner. Please refer to the 'fault_example.py' for further documentation about the handling of faults on the scanner.

Returns
True if it was possible to successfully acknowledge active faults, else False.

Return type
bool

wait_till_idle(*timeout*)

get_coordinate_frames()

Reads the transformations between important coordinate-frames of the scanner. All transformations describe the transformation of the respective frame with respect to the baser-frame. The baser-frame is located in the middle of the dome of the scanner and is the frame of the streamed pointcloud-points. Also the trajectory tracks the baser-frame with respect to a gravity aligned global-frame.

Returns
A proto-message containing the transformations to different frames with respect to the baser-frame, or None if the call failed.

Return type
Optional[coordinate_frames_message.AlignedExtrinsics]

reset_mask(*mask_type*)

Resets a custom mask on the scanner, which was previously set by 'set_mask()'.

Parameters
mask_type (masking_message.MaskIdentifier) – The mask which should be reset.

Returns
True if it was possible to reset the mask, else False.

Return type
bool

get_mask(*mask_type*)

Gets a custom mask on the scanner, which was previously set by 'set_mask()'.

Parameters
mask_type (masking_message.MaskIdentifier) – The mask which should be read.

Returns
The custom mask or None if the call failed.

Return type
Optional[masking_message.MaskDescription]

download_mask(*mask_type*, *path*)

Gets a custom mask on the scanner, which was previously set by 'set_mask()' and saves it as an image.

Parameters

- mask_type (masking_message.MaskIdentifier) – The mask which should be read.
- path (Path) – The filename and path where the image should be stored.

Returns
True if it successfully saved the mask as an image, else False.

Return type
bool

set_lidar_mask(*mask_filename*)

Sets a custom lidar mask on the scanner. This allows you to filter out pointcloud-points. Please refer to 'lidar_masking_example.py' for an example on how to properly use masks.

Parameters
mask_filename (Path) – The path to the lidar mask image.

Returns
True if it was possible to set the lidar mask, else False.

Return type
bool

set_slam_mask(*mask_filename*, *camera_id*)

Sets a custom slam-camera mask on the scanner. This allows you to filter out pixels when colorizing the pointcloud-points. Please refer to 'slam_camera_masking_example.py' for an example on how to properly use slam-camera masks.

Parameters

- mask_filename (Path) – The path to the slam camera mask image.
- camera_id (masking_message.MaskIdentifier) – The camera id for which camera the mask should be used.

Returns
True if it was possible to set the slam mask, else False.

Return type
bool

get_wifi_client_status()

Returns current information about the Wifi client mode, such as the currently connected Wifi network and whether it has access to the internet.

Returns
The Wifi client status response.

Return type
Optional[wifi_client_message.WifiClientStatusResponse]

scan_wifi_networks()

Returns a list of nearby Wifi networks detected in the last scan.

Returns
A list of all visible Wifi Networks.

Return type
Optional[wifi_client_message.ScanWifiNetworksResponse]

connect_wifi_network(*ssid*, *pwd*, *encryption*, *autoconnect=False*)

Connects the BLK ARC to the specified network using WPA/WPA2/WPA3 encryption. The connection attempt is limited to 15 seconds. The method may block up to this time.

Returns
A list of all visible Wifi Networks.

Return type
Optional[wifi_client_message.WifiClientStatusResponse]

get_known_wifi_networks()

Returns a list of the currently saved Wifi networks, sorted by priority.

Returns
A list of all known Wifi Networks.

Return type
Optional[wifi_client_message.KnownWifiListResponse]

delete_known_wifi_network(*ssid*)

Deletes a network from the list of currently saved networks. If the device is currently connected to the specified network, it will be disconnected first.

Returns
True if the service was successfully triggered. Attention: No feedback if the Wifi network got successfully deleted or not.

Return type
bool

clear_known_wifi_networks()

Clears the list of known Wifi networks. If the device is currently connected to a network, it will be disconnected first.

Returns
True if the service was successfully triggered. Attention: No feedback if the Wifi networks got successfully cleared or not.

Return type
bool

get_network_information()

Get a list of the available BLK device network interfaces, paired with their respective IP.

Returns
A list of the available network interfaces.

Return type
Optional[network_message.NetworkInformationResponse]

CHAPTER

FIVE

WORKFLOW EXAMPLES

Within the folder src/python_sample_wrapper/examples/ several example-scripts demonstrate the general workflow on how to use the blk_arc_sample_wrapper python package.

To execute an example, e.g. the capture example, make sure you followed the installation steps above and that you have activated the environment. Afterwards run the following command in your terminal:

python3 src/python_sample_wrapper/examples/capture_example.py

5.1 Capture Example

This example demonstrates how to start and stop a scan. Additionally it shows how you can mark a static pose in the scan. Static poses allow you to mark parts of the scan where the scanner was stationary. This information can be used in post-processing in order to isolate the static poses and achieve a higher accuracy.

5.1.1 How to run this example:

python3 src/python_sample_wrapper/examples/capture_example.py

5.2 Coordinate Frames Example

This example shows how to read important extrinsics from the device.

The reference-frame for all these transformations is the baser-frame, which is the frame in which the local lidar-points are streamed. Also the trajectory tracks the baser-frame.

The transformations of the following frames can be read out with respect to the baser-frame:

- Front SLAM-camera
- Left SLAM-camera
- Right SLAM-camera
- User camera
- IMU
- Housing-Reference-Pin (See technical drawings above)

5.2.1 How to run this example:

python3 src/python_sample_wrapper/examples/coordinate_frames_example.py

5.3 Fault Example

This example shows how to read scanner-faults and how to deal with them based on the severity.

5.3.1 How to run this example:

python3 src/python_sample_wrapper/examples/fault_example.py

5.4 Imaging Example

This example shows how to capture a detail-image while scanning. The image is stored inside the b2g-file.

5.4.1 How to run this example:

python3 src/python_sample_wrapper/examples/imaging_example.py

5.5 Panorama Image Stream Example

These examples show how to stream panoramic images inside as well as outside of an active scan.

5.5.1 How to run this example:

Stream panorama images while a scan is on-going:

python3 src/python_sample_wrapper/examples/pano_image_stream_scanning_example.py

Stream panorama images while no scan is active:

python3 src/python_sample_wrapper/examples/pano_image_stream_idle_example.py

5.6 Lidar Masking Example

This example introduces lidar-masks. Setting a mask allows you to filter out points from the scan which would land on your carrier or generally in areas you are not interested in. The example shows how to set, read and reset lidar masks.

5.6.1 How to run this example:

python3 src/python_sample_wrapper/examples/lidar_masking_example.py

5.7 Pointcloud Accumulation Example

This example demonstrates how to accumulate a pointcloud from the live-stream of local points and the trajectory. The pointcloud is visualized using open3d (see requirements.txt for the correct version).

5.7.1 How to run this example:

python3 src/python_sample_wrapper/examples/pointcloud_accumulation_example.py

5.8 Reboot Data Only Example

This example shows how to detect whether the device is in data-only mode (no power source connected). The example also shows how to reboot the device if the device is in data-only mode.

5.8.1 How to run this example:

python3 src/python_sample_wrapper/examples/reboot_data_only_example.py

5.9 Record Scan-Data Example

This example demonstrates how to read the scan-data (local points and trajectory) and store it inside a csv. This allows you to see how you can collect sample-data from the scanner.

5.9.1 How to run this example:

python3 src/python_sample_wrapper/examples/record_scan_data_example.py

5.10 Scans Handling Example

This example shows how to handle scans on the scanner. It shows how you can read the scans stored on the scanner, download a scan based on its ID and also how to delete scans on the scanner.

5.10.1 How to run this example:

python3 src/python_sample_wrapper/examples/scans_handling_example.py

5.11 SLAM Camera Masking Example

This example introduces camera-masks. Setting a camera-mask allows you to not accidently colorize pointcloud-points of the scan with colors of close structure on your carrier. The example shows how to set, read and reset camera masks.

5.11.1 Mask Creation

In order to mask unwanted elements within panoramic images used for point cloud colorization during the post-processing, it is necessary to mount the BLK ARC in the final position on the carrier. Use the pano_image_stream_scanning_example.py to stream a triplet of the pano images and save them as a PNGs. Use GIMP or any other familiar image editting software to modify the panoramic image. Import one of the images in the workspace and create a new layer. Create a selection of the parts that should be masked, e.g. the visible parts of the carrier. Once the selection is ready, colour the interior of the selection with white colour (full white no transparency). This parts will be masked out. Later invert the selection and colorize the image in black (corresponds to visible parts). Export the black and white layer as an image in PNG format and apply the mask.

Before applying the mask, check if the image is correct:

- Dimension of the image: 1456 x 1088
- White = the mask itself. This part of the image will not be used for colorization. The points overlapping with this part of the image will get black color information in final point cloud after post processing.
- Black = area of the image that will be used for the point cloud colorization in post processing.
- Format: png

5.11.2 How to run this example:

python3 src/python_sample_wrapper/examples/slam_camera_masking_example.py

5.12 Status Example

This example shows how to read important state-information from the scanner. On one hand this are fields like the device-name or the serial-number and firmware-version. On the other hand also the state of the scanner, for example if it is scanning or not. It also shows how to read the free disk-space.

5.12.1 How to run this example:

python3 src/python_sample_wrapper/examples/status_example.py

5.13 Wifi Client Example

This example demonstrates how to connect the BLK ARC to a Wifi network and how to handle the Wifi client.

5.13.1 How to run this example:

python3 src/python_sample_wrapper/examples/wifi_client_example.py

5.14 Network Example

This example demonstrates how to get a list of the available BLK device network interfaces, paired with their respective IP and MAC addresses.

5.14.1 How to run this example:

python3 src/python_sample_wrapper/examples/network_example.py

5.15 Timesync Example

This example shows how to read the different clock-sources of the scanner. It then shows how to perform a simplified time-sync between the computer running the example and the scanner.

5.15.1 How to run this example:

python3 src/python_sample_wrapper/examples/timesync_example.py

CHAPTER

SIX

SAMPLE ROS WRAPPER

ros_blkarc is a collection of ROS packages for interfacing the BLK ARC using the python wrapper found under src/python_sample_wrapper. This guide assumes that you are already familiar with ROS.

6.1 System Requirements

- Ubuntu 20.04 with ROS Noetic installed. For the installation, please refer to the documentation provided by ROS.

6.2 Contained Packages

- ros_blkarc_driver: ROS driver for the BLK ARC
- ros_blkarc_msgs: Custom service and action messages for the BLK ARC

6.3 Installation

1. Copy the whole API (BLKARC-Module-API) into the src folder of your catkin workspace.
    - Alternatively, if you don't want to copy it into your catkin workspace, you would need to symlink the ROS packages to your catkin workspace.
        - Replace the [BLKARC-Module-API directory] and [catkin workspace directory] accordingly and run following terminal commands
            * ln -s [BLKARC-Module-API directory]/src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc/ [catkin workspace directory]/src/
            * ln -s [BLKARC-Module-API directory]/src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc_driver/ [catkin workspace directory]/src/
            * ln -s [BLKARC-Module-API directory]/src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc_msgs/ [catkin workspace directory]/src/
        - Example: For the first command, if both BLKARC-Module-API and catkin_ws folders are located in your home directory, run
            * ln -s ~/BLKARC-Module-API/src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc/ ~/catkin_ws/src/
2. Install the BLKARC-Module-API following the instructions found in the beginning of this guide.

3. Install the required catkin tools:
    - sudo apt-get install ros-noetic-catkin python3-catkin-tools
4. If you are using a virtual environment, you need to install additional packages:
    - python3 -m pip install -r src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc/requirements.txt
5. Build the ROS packages. In your catkin workspace, run
    - catkin build ros_blkarc
    - catkin build ros_blkarc_msgs
    - catkin build ros_blkarc_driver
6. Remember to source devel/setup.bash your catkin workspace after building before you can start using the package.

6.4 Running the BLK ARC ROS Node

1. Go to src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc_driver/launch/blkarc_bringup.launch and change the necessary configuration information (e.g. connection_type) if necessary
2. Connect your BLK ARC to your computer based on the connection type you specified in the launch file.
3. Run roslaunch ros_blkarc_driver blkarc_bringup.launch
    - If connected successfully, the message "[BLKARCROSWrapper] Successfully connected to BLK over ... connection" will be displayed in the terminal.
4. Using a second terminal, you can now command the BLK ARC using the topics/services/actions shown below.

6.5 Available Topics, Services and Actions

Below are the topics, services and actions available once the BLK ARC ROS Node has been launched, with terminal commands to call them. For more details on how to call the services/actions using Python 3 code, users can refer to the rostests found in src/python_sample_wrapper/ros_sample_wrapper/ros_blkarc_driver/test.

Note: All topic/service/action names are namespaced with the sensor_name specified in blkarc_bringup.launch, which by default is set to blkarc. Please replace the blkarc in the provided commands below with the sensor_name you have specified yourself.

6.5.1 Services

- /connect
    - Description: Connect to the BLK ARC sensor
    - Terminal command: rosservice call /blkarc/connect
- /disconnect
    - Description: Disconnect from the BLK ARC sensor
    - Terminal command: rosservice call /blkarc/disconnect

- /reboot
  - Description: Reboot BLK ARC sensor. Wait around 45 seconds before attempting to connect to device again
  - Terminal command: `rosservice call /blkarc/reboot`
- /get_device_state
  - Description: Provides the current state of the device, refer to GetDeviceState.srv for all possible states
  - Terminal command: `rosservice call /blkarc/get_device_state`
- /start_capture
  - Description: Start a capture session, device must be in IDLE state (use /get_device_status service to ensure this) to start successfully
  - Terminal command: `rosservice call /blkarc/start_capture`
- /stop_capture
  - Description: Stop a capture session
  - Terminal command: `rosservice call /blkarc/stop_capture`
- /begin_static_pose
  - Description: Begin scanning in a static pose. A capture must have already started
  - Terminal command: `rosservice call /blkarc/begin_static_pose`
- /end_static_pose
  - Description: End a static pose. BLK ARC must already be static pose scanning
  - Terminal command: `rosservice call /blkarc/end_static_pose`
- /trigger_detail_image
  - Description: Trigger a detailed image. BLK ARC must already be scanning/ static pose scanning
  - Terminal command: `rosservice call /blkarc/trigger_detail_image`
- /download_scan
  - Description: Download a scan with scan_id and save it to the directory specified by scan_save_directory in the launch file. Set scan_id to -1 to download the latest scan.
  - Terminal command (replace 0 with a appropriate scan_id): `rosservice call /blkarc/download_scan "scan_id: 0"`

6.5.2 Actions

- /timed_scan
  - Description: Automatically runs a scan session with a duration of scan_time_seconds
    - Note: You will not be able to use the other services until the action has completed or the action is cancelled
  - Running from terminal:
    - Sending goal: `rostopic pub -1 /blkarc/timed_scan/goal ros_blkarc_msgs/TimedScanActionGoal '{goal: {scan_time_seconds: 20}}'`
    - Receiving feedback: `rostopic echo /blkarc/timed_scan/feedback`

* Cancel action midway: rostopic pub -1 /blkarc/timed_scan/cancel actionlib_msgs/GoalID {}

6.6 Troubleshooting

- If there are issues with running the BLK ARC ROS Node, we recommend that you first ensure you can connect to the device using the BLK ARC Module API directly and successfully run the example scripts found under src/python_sample_wrapper/examples.

6.7 Running the rostests

1. Go to ros_blkarc_driver/test/test_setup.launch and change the necessary configuration information (e.g. connection_type) if necessary.
2. Connect your BLK ARC to your computer, either through USB or Wi-Fi, but not both. We recommend using USB connection for tests because during reboot test, Wifi connection will drop and you will have to manually reconnect yourself.
3. In your catkin workspace, run catkin test ros_blkarc_driver.

INDEX

T

W