DepthScan 3D Imaging
Users Guide

Contents

Chapter 1

DepthScan System Setup

1.1 Introduction

The Ajile DepthScan 3D Imaging System is the ideal solution for machine vision and metrology applications. Fast and accurate, the system has the flexibility to successfully meet the needs of a wide variety of applications, including vision guided robots and industrial inspection. Key features of the DepthScan include

- Area scan snapshot 3D imaging system
- Designed for machine vision applications such as:
  - robotic guidance for pick and place
  - industrial inspection and metrology
- Depth accuracy better than 100 microns
- Captures 4 million color points at 2 Hz with full accuracy
- GPU and FPGA processing for real-time output
- Factory calibrated to a wide working range (25-150cm)
- Maintains spatial resolution over entire working range
- Flexibility in speed and accuracy for up to 30 point clouds per second
- High brightness LEDs provide improved depth of field and tolerance to ambient light

This document describes the hardware and software operation of the DepthScan 3D Imager. This chapter describes how to get the DepthScan up and running quickly, including setting up the hardware for the first time, and installing and running the Ajile 3D Imaging software package. Chapter 2 describes the basics of how to capture 3D images with the Ajile 3D GUI. Chapter 3 describes the DepthScan hardware in more detail to ensure proper care and operation of the device. Chapter 4 discusses the various imaging capabilities of the DepthScan and how to control its settings. The Ajile 3D Software Development Kit (SDK) is explained in Chapter 5. Finally, Chapter **??** provides some instruction on how to work with 3D data captured by the DepthScan, including performing measurements, fitting point clodus to 3D models, and 3D filtering.

1.2 Hardware Setup

The DepthScan 3D Imaging System is shown in Figure 1.1 and Figure 1.2. The main components of the system are labeled, which include the power, data and trigger interface connections, the lenses and their focus mechanisms, and the mounting holes.

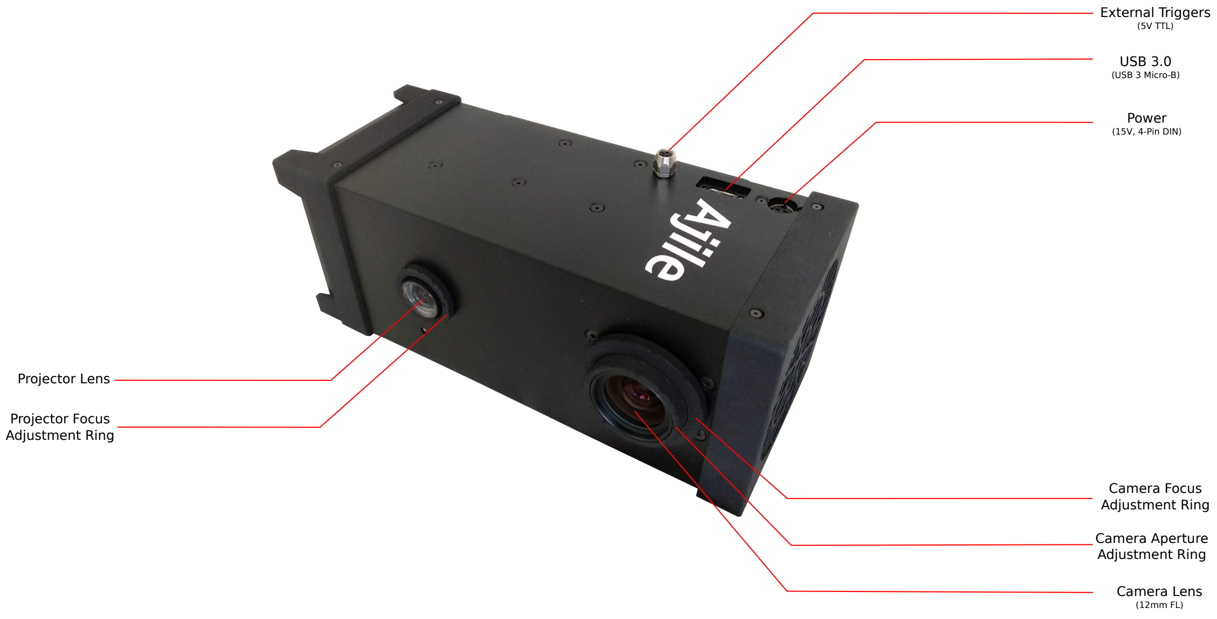


Figure 1.1: Front view of the DepthScan and labeling of its parts.

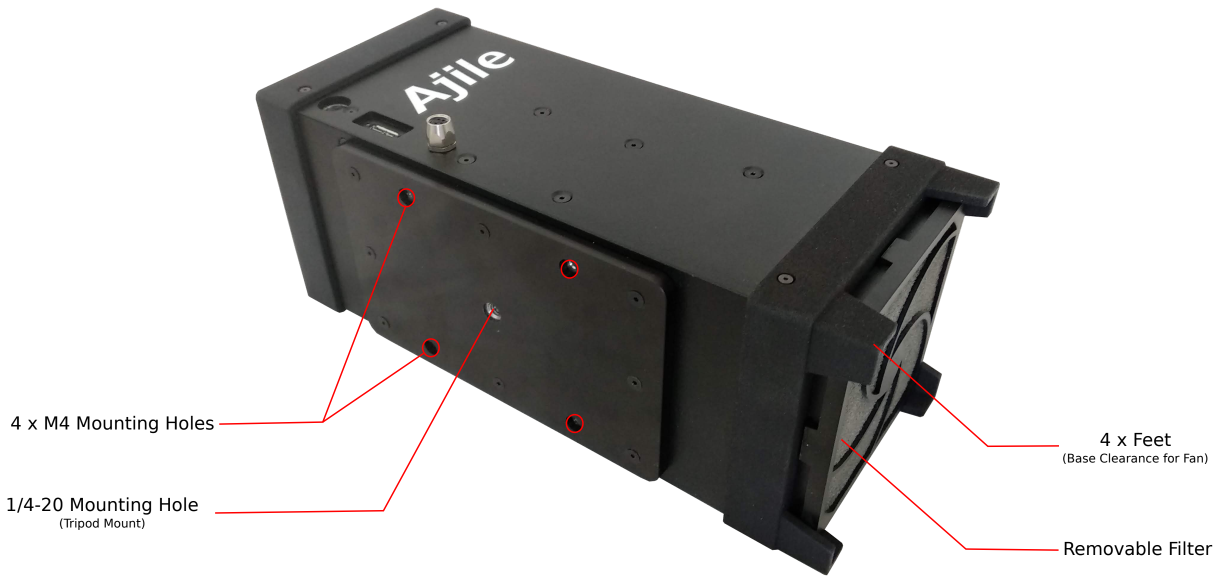


Figure 1.2: Back view of the DepthScan and labeling of its parts.

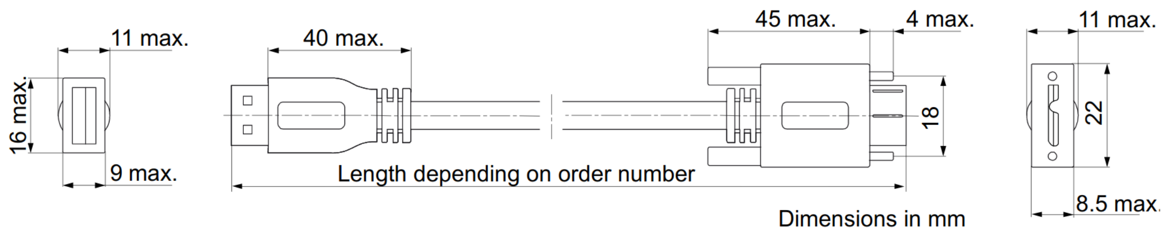


Figure 1.3: USB 3.0 Micro-B cable with locking thumbscrews. Part number Basler 2000033239.

1.2.1 Device Power

An external power supply is required to power the DepthScan. The input power connector is a DIN 4 pin plug. The recommended power supply to use is a Mean Well GST160A15-R7B, which has an output voltage of 15 V and output current of 9.6 A, for 144 W of power. See `https://www.meanwell-web.com/en-gb/ac-dc-industrial-desktop-adaptor-with-pfc-output-gst160a15--r7b` for details.

It is recommended to plug in the DIN 4 pin plug first, before plugging the power supply into the wall plug. This avoids possible safety issues of power pins making contact with the plug prior to the ground.

1.2.2 USB 3.0 Connection

The DepthScan uses USB 3.0 for communications with the host PC. The USB 3 connector type is Micro-B. Locking thumbscrew hardware as per the USB3 Vision standard is included to avoid the cable becoming dislodged, preventing communications errors. The recommended cable type is a Basler 2000033239, which is shown in Figure 1.3. Cable lengths of 3 meters or less are recommended. See `https://www.baslerweb.com/en/products/vision-components/cable/cable-usb-3-0-micro-b-screw-lock-a-3-m/` for details, or order from Mouser Electronics (`https://www.mouser.com`).

1.2.3 Lens Covers

Soft deformable lens covers are included with the DepthScan to protect the camera and projector lenses from dust and damage. Please ensure that these covers are removed prior to beginning imaging. It is recommended that you replace the lens covers when the device is not in use. To replace them you may need to stretch them slightly to fit them around the projector and camera focus rings.

1.2.4 Cooling Requirements

A high air-flow fan is included inside the DepthScan. The fan intake is at the removable filter end which is shown in Figure 1.2. Four feet are located around the filter. Clearance around the four feet is required to make sure there is sufficient air intake for the unit. The DepthScan operates best at or below 30°C, however the components are toleranced up to 40°C, albeit with reduced camera sensitivity and increased noise characteristics. For ambient temperatures above 40°C, additional cooling is required such as an external chiller - please contact Ajile for support.

The filter should be regularly cleaned and/or replaced. The filter plastic fan guard retainer un-clips, and the replacement filter part number is Qualtek part number 09362-F/45 (or Qualtek 09362-M/45 which does not include the fan guard.) The filter can be purchased from electronics reseller Digikey (`https://www.digikey.com`).

1.2.5 External Trigger Connector

The Depthscan can synchronize with external devices by way of external trigger signals (i.e. rising edge logic signals) which are transmitted across a cable and reach the DepthScan on its external trigger connector. The location of the external trigger connector can be seen in Figure 1.1. It is a 4-pin M8 female connector. Industry standard M8-4 male cables may be used to connect to the trigger interface.

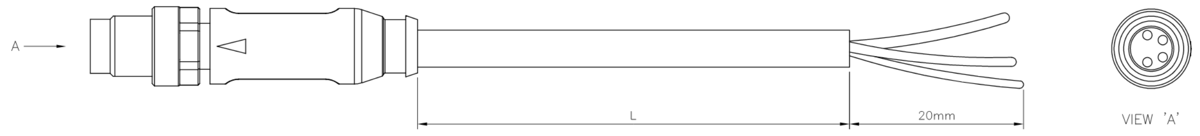


Figure 1.4: M8-4 cable used to connect to DepthScan external trigger interface. TE Connectivity part number 1-2273002-1 shown.

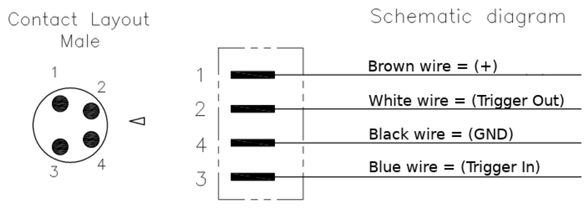


Figure 1.5: External trigger cable pinout and wire descriptions. TE Connectivity part number 1-2273002-1 shown.

The recommended cables are TE Connectivity part number 1-2273002-1, which is M8-4 to discreet wires, or TE Connectivity part number T4062113004-001, which is M8-4 female to M8-4 male. A diagram for TE Connectivity 1-2273002-1 (with discrete wires) is shown in Figure 1.4, and the description of the pin numbers and wire colors is shown in Figure 1.5. The external trigger cables can be purchased on standard electronics components resellers, such as Digikey (`https://www.digikey.com`).

NOTE: All trigger signals are opto-isolated and operate at +5V DC. Rising edge input trigger signals between +1.5V and 5V can be detected. All output trigger signals are at +5V. **Trigger voltages above +5V are not support and will cause damage to the Depthscan trigger hardware.**

1.3 Software Setup

1.3.1 PC System Requirements

Currently supported operating systems for the Ajile GUI and SDK include Windows 7 / 8.1 / 10 and Ubuntu Linux 18.04 / 20.04. Other distributions of both Windows and Linux may be possible but are not officially supported and so their operation cannot be guaranteed.

The recommended minimum PC hardware requirements are:

- Intel i5 / AMD Ryzen 5 processor or equivalent.
- 4 GB of RAM
- NVIDIA Graphics Card (GPU) with 4 GB of memory or more. See `https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units` for a list of NVIDIA GPU cards and their memory sizes. In the GeForce 10 series, the GTX 1050 Ti is the minimum which satisfies this memory requirement.
- USB 3.0 port. Since different USB 3 host controllers vary widely in terms of performance and reliability, we recommend USB 3.0 PCIe host cards from IOI. IOI part number U3-PCIE1XG205 can be supplied by Ajile and will provided with the purchase of your first DepthScan unit at no additional charge.

The Ajile software will still run with lower PC specifications, however the performance will be suboptimal, especially without a suitable NVIDIA GPU card.

1.3.2 Obtaining Ajile Software

To obtain the Ajile software suite packages, go to the Ajile downloads section of the Ajile website at `http://ajile.ca/depthscan-downloads` and select the files for your operating system of choice. You will need to enter in the login username and password which were supplied to you.

For Windows installations, download the most recent .exe installation package (e.g. ajile3d_installer_win64_2021-02-10_21.1.exe). For Ubuntu installations, download the most recent .tar.gz archive for your Linux distribution (e.g. for Ubuntu 18.04, ajile3d_suite_18.04_x86_64_2021-02-10_21.1.tar.gz).

1.3.3 Software Installation in Windows

Installing Prerequisites

Prior to installing the Ajile software please install all Windows updates by opening the Windows Update tool in Windows, checking for new updates, and selecting and installing all updates which are recommended by Windows. Note that you may need to restart your computer several times to complete the updates and each time after a restart you should again check for updates in the Windows Update tool as new updates may become available once the previous updates have been installed.

Installing Ajile Software

Once all Windows updates have been installed, double-click and open the obtained Ajile software package which is a .exe Windows installer (i.e. ajile3d_installer_win64_xxx.exe). This will open an installer program. Follow the prompts to install the Ajile 3D GUI and Ajile 3D SDK to the Program Files directory into the Ajile directory (e.g. to C:\Program Files\Ajile). The installer also creates a shortcut in the start menu for the Ajile GUI.

Starting the Ajile 3D GUI

To run the Ajile 3D GUI, find the Ajile folder in the start menu and select the Ajile GUI application. Alternately, use the Windows search tool to search for Ajile 3D GUI and select it from the Windows search. This will launch the Ajile 3D GUI and it will be ready for use.

Loading SDK Libraries

Python Libraries

Installing Python and NumPy

To use the Ajile 3D SDK for Python in Windows, you will first need to install Python and the NumPy package for Python. Currently supported versions of Python are Python 2.7.x and 3.x, 64-bit versions only. Download and install the latest 64-bit version of either Python 2.7.x or 3.x from `http://www.python.org`.

Next, the NumPy package must be installed in your Python distribution since it is used as the main tool to pass images between Python and the Ajile SDK. Details for installing NumPy can be found at `http://www.python.org`. The easiest way to install NumPy however is with Python pip tool. If your Python executable is installed at 'C:\Python\python.exe', then you can install NumPy from the command line terminal with:

```
C:\> c:\Python\python.exe -m pip install numpy
```

Installing the Ajile 3D Python Library

With Python and NumPy installed, the Ajile 3D Python libraries are located at
'C:\Program Files\Ajile\Ajile3D\lib_python' by default for Python 2.7.x, and at
'C:\Program Files\Ajile\Ajile3D\lib_python3' for Python 3.x. To use them with your existing Python2.7 (or Python3) installation, copy the files `_ajile3d.pyd` and `ajile3d.py` to your Python installation folder at %PYTHON_HOME%\Lib\site-packages (where %PYTHON_HOME% is the location of your Python

installation where python.exe is located, e.g. 'C:\Python\'). Also copy all the .dll libraries from 'C:\Program Files\Ajile\Ajile3D\lib_thirdparty' to %PYTHON_HOME%\Lib\site-packages. These are third party libraries (Pthreads, OpenCV, PCL, VTK) which the AjileDriver uses and needs to load. Once this is done you should be able to load and use the Ajile Python libraries from you Python programs. To test it try:

```
C:\> python
>>> import ajile3d
>>> aj3d = ajile3d.Ajile3DImager()
>>> print (aj3d)
<Swig Object of type 'aj3d::Ajile3DImager *' at 0x7efbf8aaa1b8>
```

If you get similar output and the type of the Ajile3DImager is printed then your Python installation was successful.

C++ Libraries

The Ajile C++ libraries are located at
'C:\Program Files\Ajile\Ajile3D\lib' by default. In addition the include headers are located at
'C:\Program Files\Ajile\Ajile3D\include. For building C++ applications using the Ajile SDK we use Visual Studio 2017. In addition we use the freely available CMake (`https://cmake.org/`) to generate Visual C++ projects that use the Ajile 3D SDK. It is possible to use the include files and library files with other build tools, but we show an example here based on CMake and Visual C++ 2017.

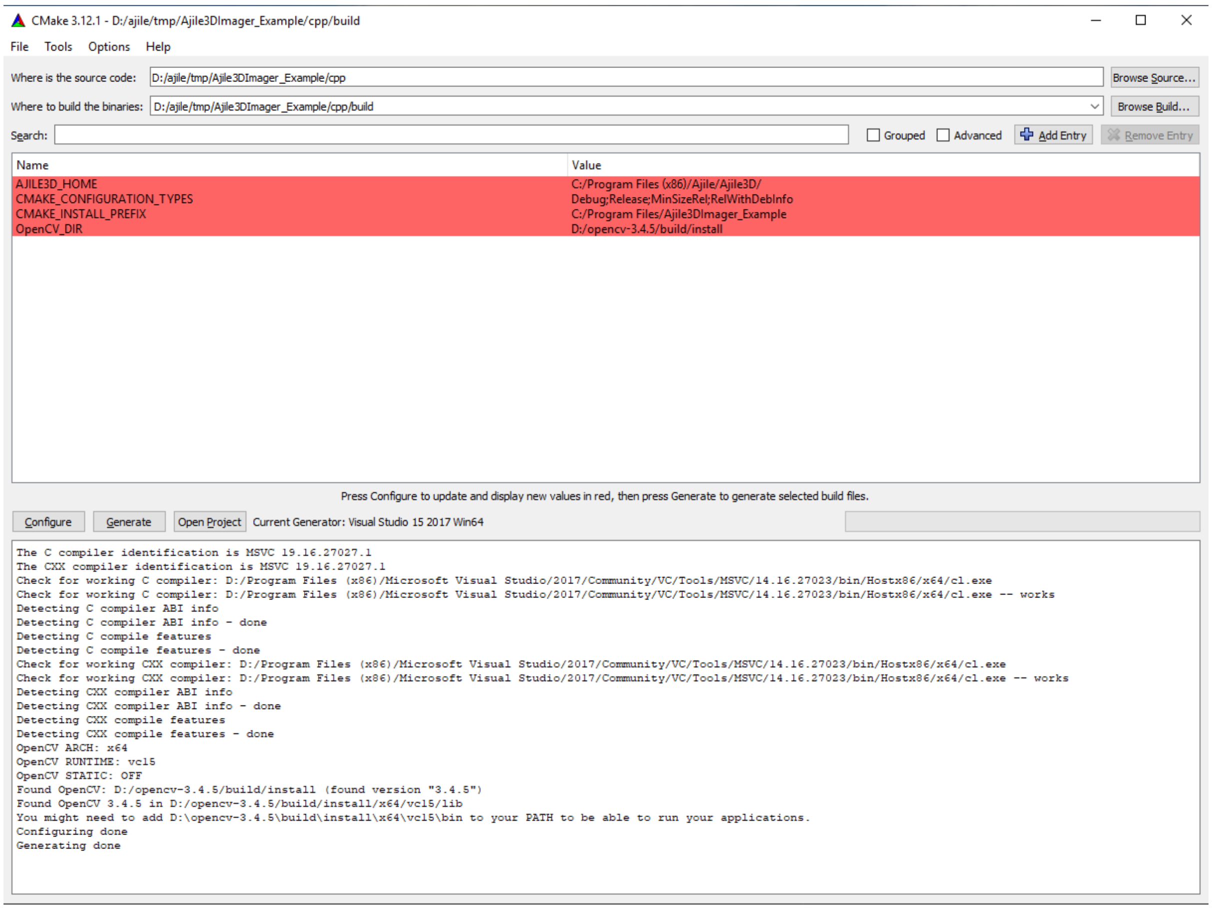
To build with CMake we first need a CMakeLists.txt file. The following is a minimal CMakeLists.txt file which uses the AjileDriver.

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
PROJECT(ajile3d_test)
SET(AJILE3D_HOME "C:/Program Files/Ajile/Ajile3D/")
ADD_DEFINITIONS(-DWIN32_LEAN_AND_MEAN)
SET(SOURCE main.cpp)
INCLUDE_DIRECTORIES(
  ${CMAKE_CURRENT_BINARY_DIR}
  ${CMAKE_CURRENT_SOURCE_DIR}
  ${AJILE3D_HOME}/include)
ADD_EXECUTABLE(project_test ${SOURCE})
TARGET_LINK_LIBRARIES(
  ajile3d_test
  ${AJILE3D_HOME}/lib/ajile3d.lib )
```

A minimal main.cpp source file to accompany it would be as follows:

```
#include <ajile3d/AJObjects.Ajile3DImager>
#include <iostream>
void main() {
    aj3d::Ajile3DImager aj3d;
    std::cout << aj3d.IsConnected() << endl;
}
```

You can create the CMakeLists.txt file and main.cpp file in the same directory, then run CMake and point the source directory where you created those files (i.e. run Configure followed by Generate, see Figure 1.6). You can then open the generated Visual Studio solution (.sln file) and build it. This will output an executable, ajile3d_test.exe, into the Release or Debug directory. If you try to run the executable you will likely get an error indicating that the ajile3d.dll library is missing. To fix this problem, copy the file 'C:\Program Files\Ajile\Ajile3D\lib\ajile3d.dll' as well as all the .dll libraries from 'C:\Program Files\Ajile\Ajile3D\lib_thirdparty' to the location of your executable (e.g. in Release or Debug). As an alternative to copying all .dll from the Ajile installation directory each time, you can also install the .dll



Where is the source code: D:/ajile/tmp/Ajile3DImager_Example/cpp

Where to build the binaries: D:/ajile/tmp/Ajile3DImager_Example/cpp/build

Search:

Name	Value
AJILE3D_HOME	C:/Program Files (x86)/Ajile/Ajile3D/
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX	C:/Program Files/Ajile3DImager_Example
OpenCV_DIR	D:/opencv-3.4.5/build/install

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 15 2017 Win64

```
The C compiler identification is MSVC 19.16.27027.1
The CXX compiler identification is MSVC 19.16.27027.1
Check for working C compiler: D:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x64/cl.exe
Check for working C compiler: D:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x64/cl.exe -- works
Detecting C compiler ABI info
Detecting C compiler ABI info - done
Detecting C compile features
Detecting C compile features - done
Check for working CXX compiler: D:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x64/cl.exe
Check for working CXX compiler: D:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x64/cl.exe -- works
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Detecting CXX compile features
Detecting CXX compile features - done
OpenCV ARCH: x64
OpenCV RUNTIME: vc15
OpenCV STATIC: OFF
Found OpenCV: D:/opencv-3.4.5/build/install (found version "3.4.5")
Found OpenCV 3.4.5 in D:/opencv-3.4.5/build/install/x64/vc15/lib
You might need to add D:\opencv-3.4.5\build\install\x64\vc15\bin to your PATH to be able to run your applications.
Configuring done
Generating done
```

Figure 1.6: CMake GUI for Ajile3DImager_Example after running configure and generate for Visual Studio 2017.

libraries to a system directory which is in the user's PATH, or add 'C:\Program Files\Ajile\Ajile3D\lib\' and 'C:\Program Files\Ajile\Ajile3D\lib_thirdparty\' to the %PATH% variable.) You should then be able to run the executable in a console window and see the 'false' displayed (since aj3d.IsConnected() evaluates to false).

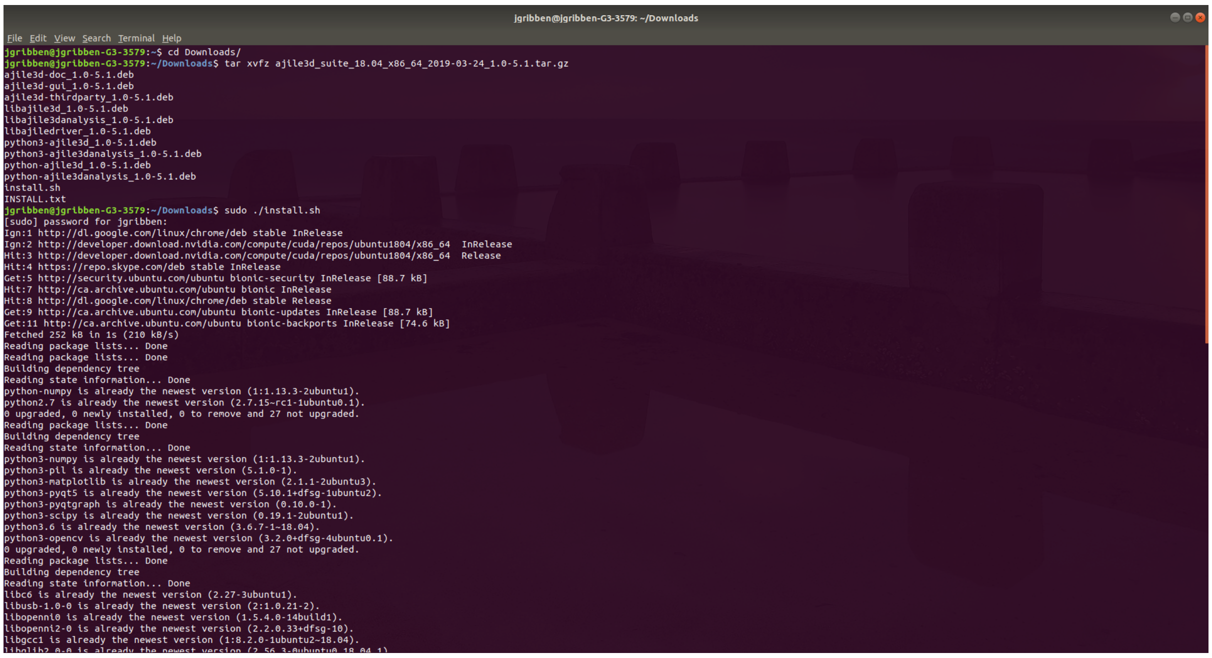
1.3.4 Software Installation in Ubuntu

Installing Ajile Software

To install the Ajile 3D software in Ubuntu you will need to first extract the files from the .tar.gz archive which you downloaded. This will extract a number of .deb Debian packages in the same directory as the archive, as well as an installation script, install.sh. The install.sh script will install all prerequisite software and each of the packages into the proper system directories. The steps to install the Ajile 3D software in Ubuntu is therefore as follows:

1. Open a new Terminal window to get a system command line interface. On Ubuntu systems this may be called Terminal in the Application → Accessories menu.
2. Change directories to the location where you downloaded the .tar.gz archive. For example if you downloaded the file to the /Downloads directory, the command to type in the terminal is `cd /Downloads`.
3. Extract the files from the .tar.gz archive with

   ```
   tar xvf ajile3d__linux64_xxx_xxxx.tar.gz
   ```


jgribben@jgribben-G3-3579: ~/Downloads
File Edit View Search Terminal Help

```
jgribben@jgribben-G3-3579:~$ cd Downloads/
jgribben@jgribben-G3-3579:~/Downloads$ tar xvfz ajile3d_suite_18.04_x86_64_2019-03-24_1.0-5.1.tar.gz
ajile3d-doc_1.0-5.1.deb
ajile3d-gui_1.0-5.1.deb
ajile3d-thirdparty_1.0-5.1.deb
libajile3d_1.0-5.1.deb
libajile3danalysis_1.0-5.1.deb
libajiledriver_1.0-5.1.deb
python3-ajile3d_1.0-5.1.deb
python3-ajile3danalysis_1.0-5.1.deb
python-ajile3d_1.0-5.1.deb
python-ajile3danalysis_1.0-5.1.deb
install.sh
INSTALL.txt
jgribben@jgribben-G3-3579:~/Downloads$ sudo ./install.sh
[sudo] password for jgribben:
Ign:1 http://dl.google.com/linux/chrome/deb stable InRelease
Ign:2 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
Hit:3 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release
Hit:4 https://repo.skype.com/deb stable InRelease
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:7 http://ca.archive.ubuntu.com/ubuntu bionic InRelease
Hit:8 http://dl.google.com/linux/chrome/deb stable Release
Get:9 http://ca.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:11 http://ca.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Fetched 252 kB in 1s (210 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-numpy is already the newest version (1:1.13.3-2ubuntu1).
python2.7 is already the newest version (2.7.15~rc1-1ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 27 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-numpy is already the newest version (1:1.13.3-2ubuntu1).
python3-pil is already the newest version (5.1.0-1).
python3-matplotlib is already the newest version (2.1.1-2ubuntu3).
python3-pyqt5 is already the newest version (5.10.1+dfsg-1ubuntu2).
python3-pyqtgraph is already the newest version (0.10.0-1).
python3-scipy is already the newest version (0.19.1-2ubuntu1).
python3.6 is already the newest version (3.6.7-1~18.04).
python3-opencv is already the newest version (3.2.0+dfsg-4ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 27 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
libc6 is already the newest version (2.27-3ubuntu1).
libusb-1.0-0 is already the newest version (2:1.0.21-2).
libopenni0 is already the newest version (1.5.4.0-14build1).
libopenni2-0 is already the newest version (2.2.0.33+dfsg-10).
libgcc1 is already the newest version (1:8.2.0-1ubuntu2~18.04).
libglib2.0-0 is already the newest version (2.56.3-0ubuntu0.18.04.1)
```

Figure 1.7: Screenshot of installing the Ajile 3D Debian packages.

where _xxx_xxxx should be replaced with the characters in the file you just downloaded.

4. Install the software by running the install script with the command

   ```
   sudo ./install.sh
   ```

5. Follow the prompts to select which prerequisites and packages get installed.

See Figure 1.7 for commands and the typical output that will be seen when installing from the command line. Once completed, the Ajile 3D Software will be installed with both the C++ and Python interfaces as well as the Ajile 3D GUI.

Starting the GUI

To start the Ajile 3D GUI, enter the following command at the Terminal prompt:

```
ajile3d-gui
```

You can also start the Ajile 3D GUI with Run Application dialog by pressing Alt-F2 in Ubuntu and entering the command `ajile3d-gui`.

Loading SDK Libraries

The Debian packages install the C++ Ajile 3D library to /usr/lib, the C++ Ajile 3D headers to /usr/include/ajile3d, and the Python Ajile 3D library to /usr/lib/python2.7/dist-packages for Python 2.7 or /usr/lib/python3/dist-packages for Python 3. On most systems these paths are automatically found by the C++ compiler (GCC) and by the Python interpreter and you should be ready to start designing your own software using the Ajile SDK.

1.4 Upgrading Ajile 3D Imaging Software and Firmware

Ajile regularly releases new software and firmware versions which improve performance, add new features and fix certain bugs from previous versions. To check for the most recent version of the Ajile 3D imaging software and firmware, go to `http://ajile.ca/depthscan-downloads`.

The software and firmware upgrade steps are detailed below.

1.4.1 Upgrading Ajile Software

The most recent versions of software can be found at `http://ajile.ca/depthscan-downloads`. Click on the link to download the installer for the operating system that you are using and save it to your computer. For example, for Windows 10 the file to download will have a name similar to ajile3d_installer_win64_2021-02-10_21.1.exe.

For Windows 10 users, the installer is a .exe executable. If you have a previous version of the Ajile 3D Software installed on your PC you may need to first uninstall the previous version before beginning installation of the new version. To uninstall the Ajile 3D imaging software in Windows, use the 'Add or remove programs' Windows utility, select the 'Ajile 3D Software, and click the 'Uninstall' button (note that there may be minor variations depending on your Windows version). When the uninstall completes you can install the newly downloaded software release on the PC by double clicking the executable then following the on-screen instructions in the same way as Section 1.3.

For Ubuntu Linux users, the install.sh script inside the downloaded archive (which will have a name similar to ajile3d_suite_18.04_x86_64_2021-02-10_21.1.tar.gz) takes care of first removing the existing Ajile 3D software before attempting to install the new version, and so the software upgrade instructions are the same as the software installation instructions in Section 1.3.

1.4.2 Upgrading DepthScan Firmware

To upgrade the DepthScan device firmware, first download the most recent firmware release from `http://ajile.ca/depthscan-downloads`. The downloaded filename will be of the format DEPTHSCAN_release_X.ajm, where X is the release version number.

NOTE: It is recommended to first download and install the most recent version of the Ajile software on the host PC following the Upgrading Ajile Software instructions above, since the firmware update tool may have important updates.

Next open the Ajile 3D GUI and click on the Connect button in the top left corner to connect the device. Then click on the System menu item, then on Upgrade Device Firmware. Select the downloaded firmware image (i.e. DEPTHSCAN_release_X.ajm) with the browse file '...' button. Finally, click the 'Start Upgrade' button. The system will ensure that the selected firmware image is valid for the target device, then a dialog will appear to confirm whether or not to proceed with the firmware upgrade, and so click 'Yes'. The firmware update can take several minutes to complete so do not close the GUI or power off the device during this procedure. The progress percentage will increment during this update. The firmware update procedure can take several minutes. Note that it may take one minute or more for the progress percentage to increase from 0%, so please be patient with the update. If the progress percentage stays at 0% for 10 minutes or more it is possible that there was a communication error when sending the firmware image to the device. If this happens, try closing the Ajile 3D GUI, then reconnect and try again. Finally, when the firmware upgrade completes a new prompt will appear and you will be instructed to power cycle the device. Do so, then when the device restarts it is ready for use. You can verify the updated firmware version checking the System → Hardware Properties menu item under Device Properties.

NOTE: It is very important to not disconnect or power off the device during the firmware update process. Doing so may cause the hardware to stop responding and require returning the unit to Ajile for repair.

Chapter 2

Capturing 3D Images

In the previous chapter we saw how to set up the DepthScan hardware and software. In this chapter we will begin working with the DepthScan system, starting with capturing 3D images using the Ajile 3D GUI and viewing the results as point clouds.

2.1 Ajile 3D GUI Overview

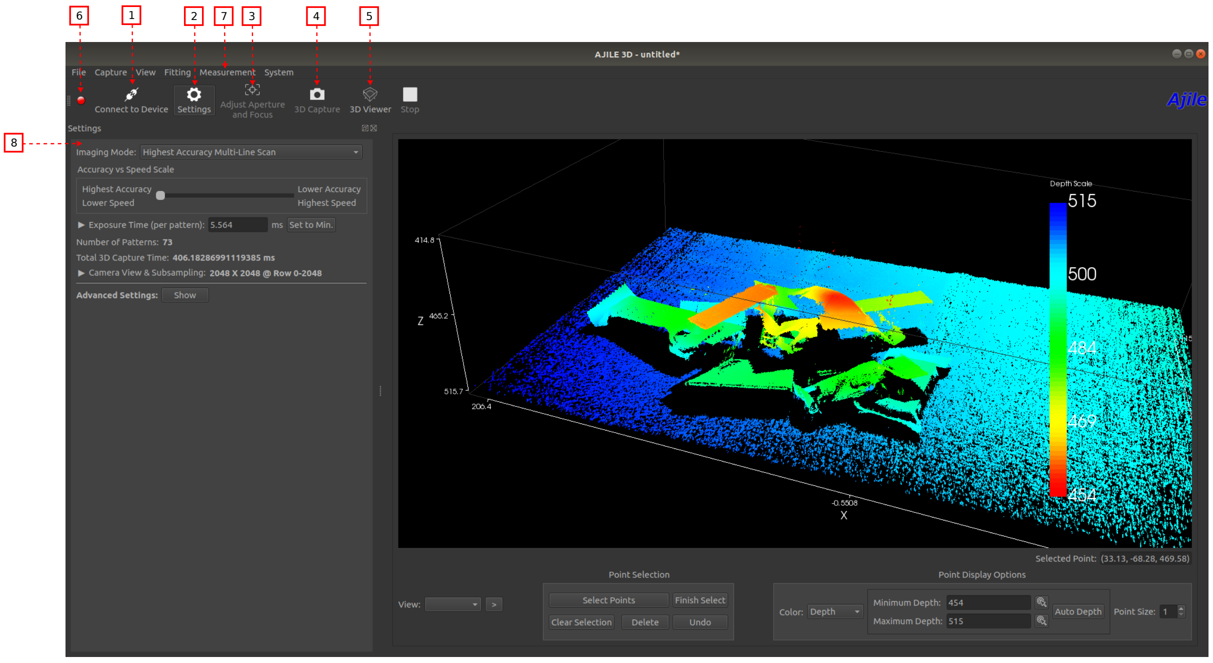
The Ajile 3D GUI is designed to work with the DepthScan and performs the following main tasks:

- Displays and configures the DepthScan settings
- Captures 3D images with the DepthScan
- Displays captured 3D images with its built in point cloud viewer
- Performs basic measurements on captured 3D images
- Performs a number of standard 3D image analysis algorithms such as fitting and comparing point clouds to CAD models and shapes.

The main components of the Ajile 3D GUI are shown in Figure 2.1. The top toolbar of the GUI controls the overall flow of capturing high quality 3D images. The steps to go from first opening the Ajile 3D GUI to having available a 3D point cloud to manipulate is as follows:

1. Connect to the DepthScan by clicking on the Connect button (Figure 2.1, item 1).
2. Open the imaging settings by clicking on the Settings button (Figure 2.1, item 2), then adjust imaging settings (exposure time, etc.) in the Settings panel (Figure 2.1, item 8).
3. Adjust the camera and projector aperture and focus by clicking on the Adjust Aperture and Focus button (Figure 2.1, item 3).
4. Start the 3D capture by clicking on the 3D Capture button (Figure 2.1, item 4).
5. The integrated 3D viewer will automatically appear, displaying the captured 3D point cloud for viewing and analysis. Click on the 3D Viewer button (Figure 2.1, item 5) at any time to open the 3D viewer.
6. Perform 3D fitting, alignment, and measurement tasks on the 3D image with operations available in the menu bar (Figure 2.1, item 7), or save the 3D image to one of the available point cloud formats with the File $\rightarrow$ Export $\rightarrow$ Point Cloud menu.

The steps in this capture flow will be described in the remainder of this chapter.



Imaging Mode: Highest Accuracy Multi-Line Scan

Exposure Time (per pattern): 5.564 ms

Number of Patterns: 73

Total 3D Capture Time: 406.18286991119385 ms

Camera View & Subsampling: 2048 X 2048 @ Row 0-2048

Selected Point: (33.13, -68.28, 469.58)

Color: Depth

Minimum Depth: 454

Maximum Depth: 515

Point Size: 1

Figure 2.1: Main components of the Ajile 3D GUI.

2.2 Connecting to the Device

With the external power supply connected to the DepthScan and powered on, and the USB 3.0 cable connecting the PC to the DepthScan, we are ready to connect to the device. The DepthScan takes just over a minute to initialize. You will know it is ready to connect to after a white 'Ajile' logo has been projected from the structured light projector, and the fan starts up for 1 second.

To connect to the device, click on the 'Connect' button in the GUI (Figure 2.1, item 1). A dialog box will pop up indicating that we are trying to connect, and the status indicator light in the top left corner of the GUI (Figure 2.1, item 6) will change color from red (disconnected) to yellow (connecting). When it is finished, the dialog box will disappear, and the 'Connect' button will stay pressed down, and the status indicator lighting will change color to green indicating that we are connected. If an error message appears indicating that the connection failed, check the device power, USB 3 connection, and check that all drivers are properly installed with the Ajile 3D Software installer. See the troubleshooting section or contact Ajile support if you continue to have problems connecting.

2.3 Configure Imaging Settings

The DepthScan has many possible imaging settings that can be adjusted to optimize trade-offs between speed, accuracy, resolution, number of localized points, etc. Chapter 4 will describe many of the settings in detail. For now, we will just describe the most basic settings which are the exposure time and the imaging mode.

The imaging settings can be displayed by clicking on the Settings button (Figure 2.1, item 2). The first imaging setting that can be adjusted is the imaging mode. This is a drop down menu which shows the different imaging modes available by default in the DepthScan. These imaging modes trade off between accuracy and speed by adjusting the number of structured light images used per scan, and the camera sensor resolution. In general, more patterns and higher camera sensor resolution means higher 3D measurement accuracy, at the cost of lower speed. The default imaging mode is the highest accuracy mode which uses 73 structured light patterns at full resolution. This imaging mode is recommended for most applications since it gives the best accuracy, and the speed is still relatively high with a total 3D capture time of 406 ms using the default exposure time settings.

The exposure time can be easily adjusted in the settings window. Simply type in a new exposure time in the ‘Exposure Time (per pattern)’ text box. This is the exposure time per structured light pattern and camera image, in milliseconds. Note that this is *not* the exposure time for the entire 3D capture sequence, since each 3D image capture is performed by scanning multiple patterns. The total 3D capture time is the entered in exposure time, multiplied by the total number of patterns, so for the default case of a 5.564 ms exposure (minimum full camera resolution exposure) with 73 patterns per 3D image this is 5.564ms $\times$ 73 = 406.2ms. Also note that while the camera exposure time can be very short, below 1 ms, the total 3D capture time may remain the same since the camera image readout time will be longer than the camera exposure time, with a minimum of 5.564 ms at the full 2048 $\times$ 2048 resolution.

2.4 Adjusting Aperture and Focus

The first and probably most important step to capturing high quality 3D images is setting up the camera and projector in the DepthScan so that the scene is in focus and so that we have enough image contrast for our structured light patterns. For this we use the setup mode where we have a live preview of the camera and projector images.

2.4.1 Starting Aperture and Focus Image Setup Mode

To enter setup mode so that we can adjust the camera and projector, click on the ‘Adjust Aperture and Focus’ button (Figure 2.1, item 3). This will cause the projector to project line patterns onto the scene, and a live video stream from the camera to appear in the GUI. Additional information will also be displayed in the GUI which indicates the number of successfully located points in the scene and the quality of the focus with the current exposure time, aperture and focus settings.

2.4.2 DepthScan Focus and Aperture Adjustment

The DepthScan has a variable focus system for both its camera and projector. This makes it extremely versatile since it is able to capture 3D images of small objects at a working distance of only a few centimetres, all the way up to larger objects at working distances of meters. However, it is very important to carefully focus both the camera and projector prior to imaging objects at a new depth, since proper focus directly affects the 3D measurement accuracy and the number of points which can be captured in the scene.

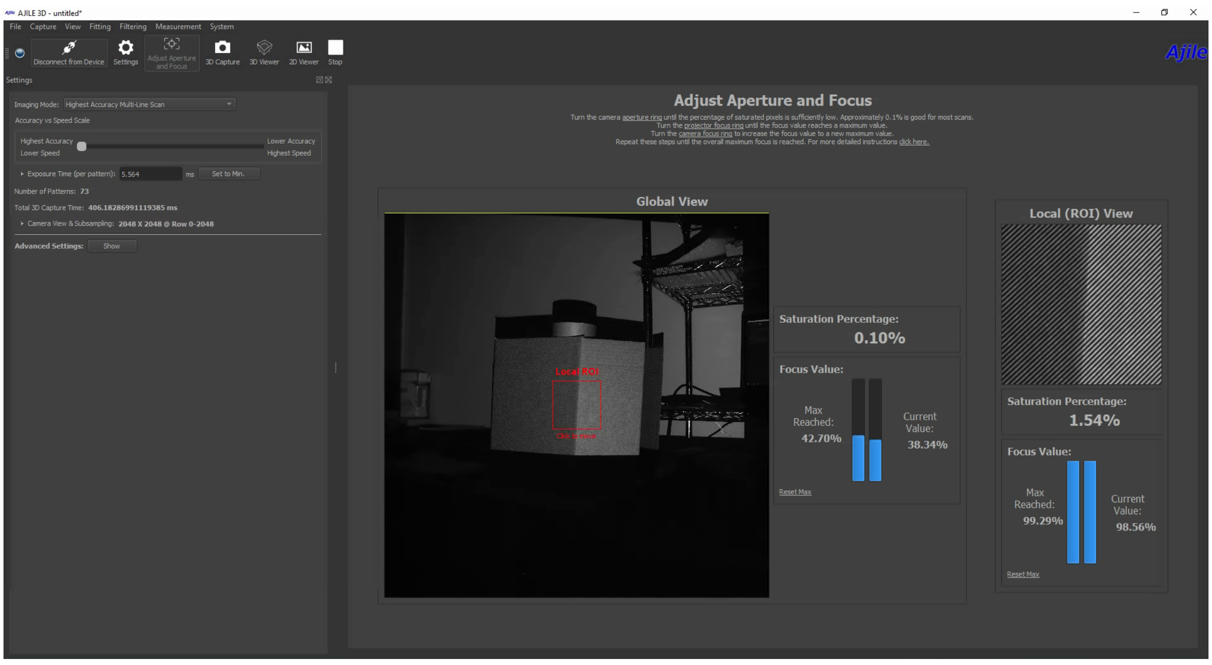
To get the best possible focus across the scene and the correct camera aperture setting the Ajile 3D GUI has assistive tools which give the user feedback on the current focus quality and the scene saturation. The ‘Adjust Aperture and Focus’ tool can be seen in Figure 2.2. The ‘Global View’ on the left shows the camera image of the entire scene, and the ‘Local (ROI) View’ on the right shows a zoomed in view of a selected portion of the scene. Left clicking on a different part of the scene in the Global View will cause the Local ROI to move to the clicked location, so that the focus can be optimized for certain objects and ignored for others.

The overall recommended technique for focusing the DepthScan is summarized by the following steps:

1. Open the camera aperture to make it easier to focus the projector and camera.
2. Adjust the projector focus, maximizing the GUI focus value (and the sharpness of the projected line patterns).
3. Adjust the camera focus, maximizing the GUI focus value.
4. Adjust the camera aperture until the saturation percentage is in the correct range.
5. Repeat steps 2-4 to refine the focus and aperture, until no further improvement can be observed.

1. Initial Camera Aperture Setting

To begin with focusing the camera and projector, we first open the camera aperture slightly which does two things: first it allows for more light to reach the camera making it easier to see the scene for focusing,



Settings

Imaging Mode: Highest Accuracy Multi-Line Scan

Accuracy vs Speed Scale

Highest Accuracy / Lower Speed — Lower Accuracy / Highest Speed

Exposure Time (per pattern): 5.564 ms [Set to Min.]

Number of Patterns: 73

Total 3D Capture Time: 406.18286991119385 ms

Camera View & Subsampling: 2048 X 2048 @ Row 0-2048

Advanced Settings: [Show]

Adjust Aperture and Focus

Turn the camera aperture ring until the percentage of saturated pixels is sufficiently low. Approximately 0.1% is good for most scans.
Turn the projector focus ring until the focus value reaches a maximum value.
Turn the camera focus ring to increase the focus value to a new maximum value.
Repeat these steps until the overall maximum focus is reached. For more detailed instructions click here.

Global View: Saturation Percentage: 0.10%; Focus Value: Max Reached: 42.70%, Current Value: 38.34%

Local (ROI) View: Saturation Percentage: 1.54%; Focus Value: Max Reached: 99.29%, Current Value: 98.56%

Figure 2.2: Main components of the Adjust Aperture and Focus tool.

and secondly it reduces the depth of field of the camera so that changes in the projector and camera focus will have a more pronounced effect on the image focus. Opening the aperture until the areas of interest in the camera image are bright but not completely washed out (saturated) is best for this.

The camera aperture ring is shown in Figure 2.3. Note that for the camera there are two knobs that are around the camera lens. The inner knob located beneath the green arrow in Figure 2.3 is the aperture adjustment knob, while the outer knob beneath the red arrow in Figure 2.3 is the camera focus knob.

For very dark or very bright scenes, it may be necessary to also adjust the camera exposure time. This can be done by changing the Exposure Time value, in milliseconds, shown in the GUI settings (Figure 2.1, item 8), then clicking 'Stop' followed by clicking 'Adjust Aperture and Focus' to restart the camera so that the new setting takes effect.

2. Adjust Projector Focus

The next step is to focus the projector. When running the 'Adjust Aperture and Focus' tool, a repeating line pattern is projected onto the scene by the DepthScan structured light projector. The lines may be focused on the scene by gently turning the projector focus adjustment ring until the projected lines become as sharp as possible onto the object(s) of interest. The projector focus knob and the way that it rotates to change its focus is shown in Figure 2.4.

Once the projected lines are visible on the object of interest, the focus can be optimized by clicking on the object of interest in the Global View so that it becomes visible in the Local (ROI) View, then very slowly turn the projector focus ring until the Focus Value number is as high as possible. The Focus Value is a number between 0% and 100%, with 100% being perfectly in focus, so try to get this as high as possible. The actual property that the Focus Value is showing is the number of pixels in the scene which can correctly return a 3D point. Pixels which do not correctly return 3D points are either occluded (due to the camera/projector angle), or they are out of focus.

3. Adjust Camera Focus

To focus the camera, turn the camera focus ring which is the outer knob shown in red in Figure 2.3. Unlike the projector focus adjustment, we cannot see the pattern on the scene change when we change

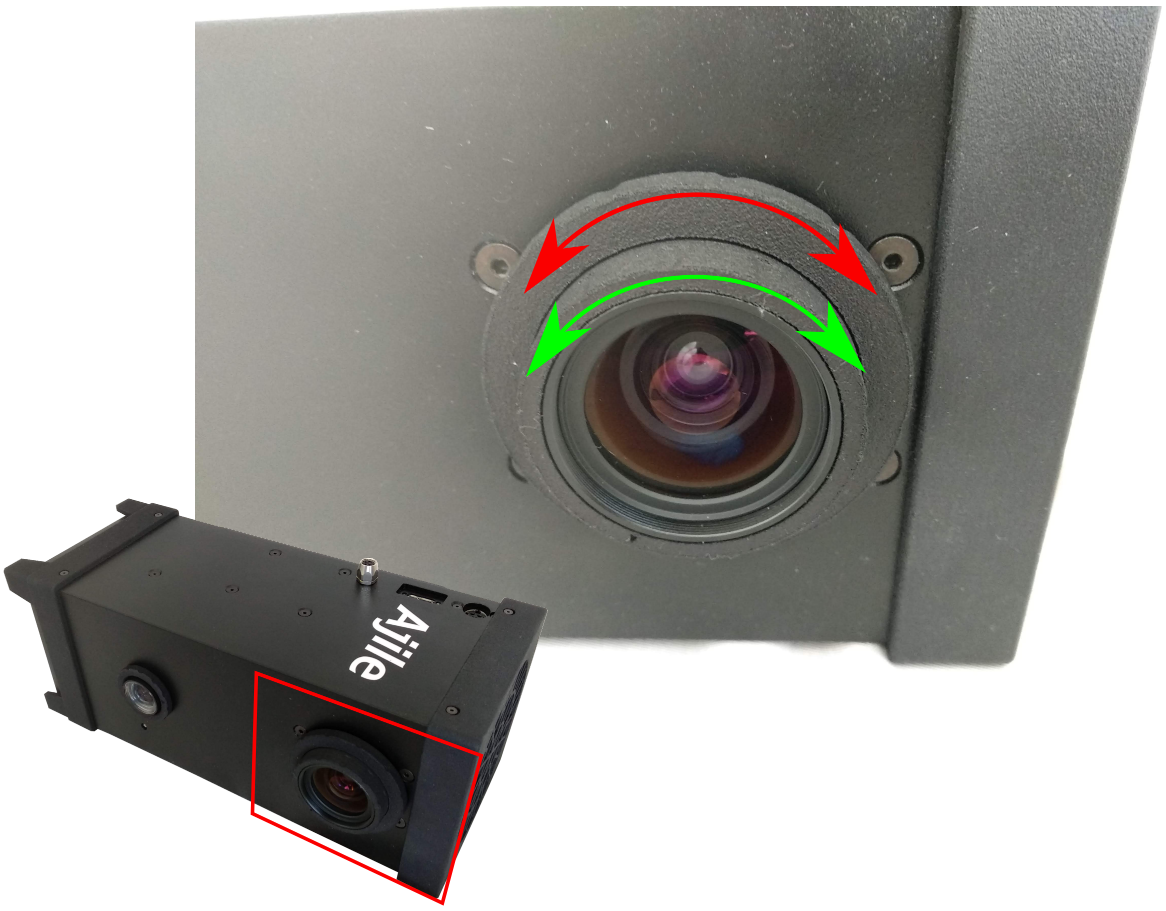


Figure 2.3: Camera focus knob (red) and camera aperture knob (green).

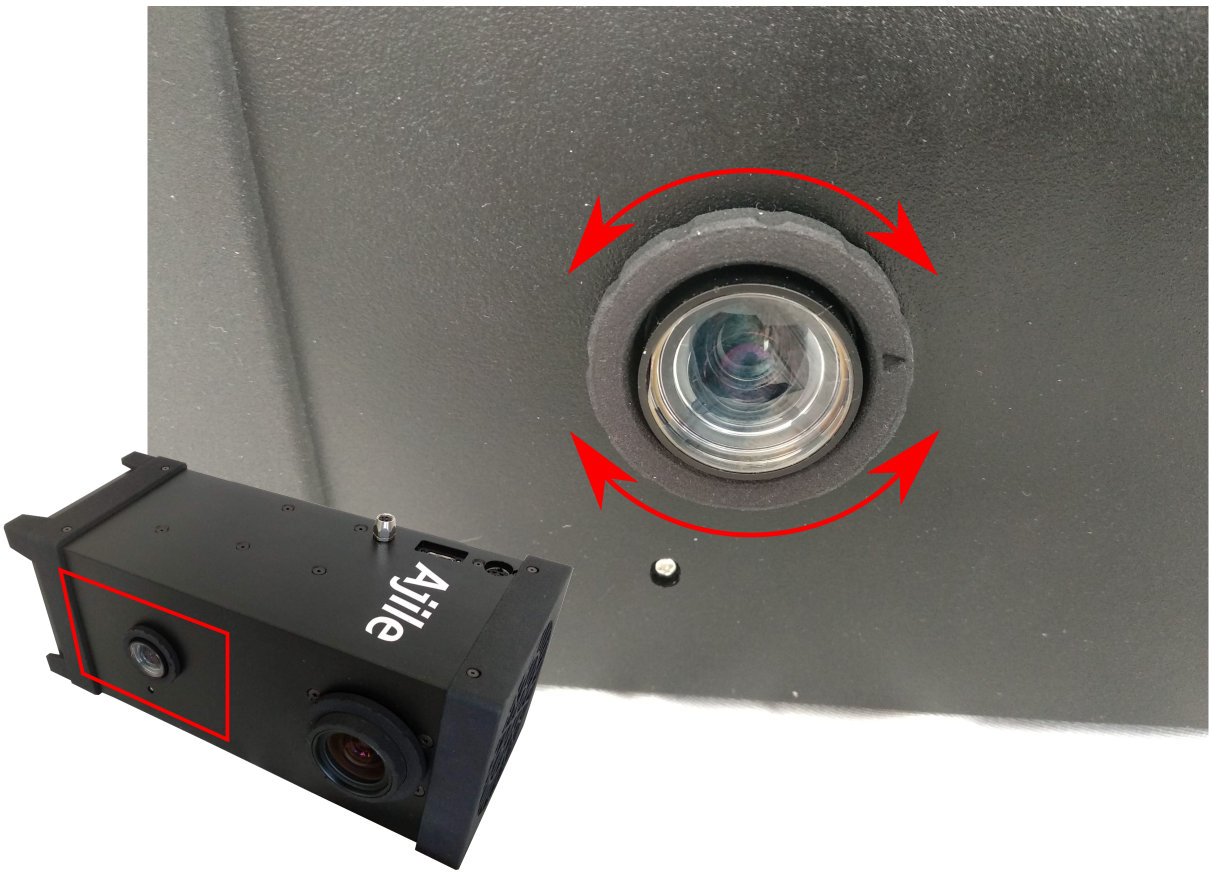


Figure 2.4: Projector focus knob.

the camera focus but instead can observe the live camera image shown in the GUI. As with the projector focus adjustment, we optimize the camera focus by turning the camera focus ring until the Focus Value is at its maximum. You will also see the camera image in the GUI being its sharpest when the camera focus is best.

4. Adjust Camera Aperture

The final step is to adjust the camera aperture to bring the light intensity level in the camera into the correct range so that pixels are neither saturated (washed out) or underexposed (too dark). We do this by turning the camera aperture knob, shown in green in Figure 2.3.

While turning the aperture knob (ring) we watch both the camera image in the GUI and the Saturation Percentage value which is reported on the screen. The Saturation Percentage reports the percentage of pixels which are saturated in the scene. A good value to aim for with the Saturation Percentage is around 0.1% rather than 0. This will ensure that the pixels are not underexposed (too dark), but is a low enough value to also ensure that pixels are not overexposed (washed out). Note that the saturation threshold reported in the GUI for the Saturation Percentage value is actually slightly lower than the actual camera sensor saturation limit, so setting a low but non-zero Saturation Percentage will ensure that we are within the correct range for the camera and also prevent actual saturation of the camera sensor.

5. Refining the Focus and Aperture

After adjusting the camera aperture (Step 4) it may be a good idea to repeat the projector focus and camera focus adjustment (Step 2 and 3) before taking the 3D image, since camera focus and projector focus are coupled with each other. You can repeat this as many times as you need to to be confident that you have maximized the Focus Value (and have a well focused live image from the camera) and that the saturated pixels are minimal.

The Local (ROI) View is very useful for imaging scenes which contain objects at different depths, for example when there is a foreground object of interest closer to the camera and background clutter further from the camera which we are not interested in. In this case, click on the object of interest in the Global View to highlight it in the Local (ROI) View. You may then optimize the focus by looking at the Focus Value beneath the Local (ROI) View.

A final note on the aperture: if longer exposure times are not a problem for your application, it is recommended to first adjust exposure times when not enough light is reaching the camera sensor, and keep the aperture small to have better depth of field. This is done by changing the Exposure Time value in the Settings panel, which is highlighted in Figure 2.1. The exposure time is in milliseconds in the GUI, so in the case of Figure 2.1 it is set to 5.5 ms.

2.5 Capturing 3D Images

With the DepthScan camera and projector components are properly set up and focused, it is time to capture a 3D image. We do so by clicking on the 'Capture 3D Image' button in the top toolbar (the camera icon). With the default settings, a series of binary structured light patterns will be projected onto the scene and simultaneously imaged by the camera. The camera images will be automatically processed into a 3D point cloud, which will then be displayed in the 3D point cloud viewer.

2.6 Viewing 3D Point Clouds

Once a 3D image is captured, by default it will be displayed in the 3D point cloud viewer window of the Ajile 3D GUI. Use the mouse to view and navigate the point cloud. The basic controls are as shown in Table 2.1.

Control	**Description**
Left-click and drag mouse	Rotate the camera around the current focal point.
Double-click	Change the focal point of camera to the nearest point to the mouse pointer.
Right-click and drag mouse	Move the camera in the direction of the mouse movement.
Mouse wheel	Zoom the camera in and out.

Table 2.1: List of controls for the Ajile 3D GUI point cloud viewer.

Chapter 3

DepthScan Hardware Details

This chapter describes the hardware details of the Depthscan, including the Depthscan optics, thermal management, and electrical characteristics.

3.1 Imaging Volume

The Depthscan measures 3D coordinates of objects in the scene by triangulating rays which are projected from its structured light projector and which are captured by pixels on its camera sensor. In order to perform this triangulation depth measurement at a given point, the corresponding projected pixel but be visible by the camera and not occluded by other parts of the scene. Since the camera in the Depthscan always has a larger field than the projector, the Depthscan field of view is completely governed by the projector optics.

The field of the view of the Depthscan at up to 70 cm working distance is shown in Figure 3.1. The area in green shows the image size as it is projected from the structured light projector, which completely governs the field of view. For other distances beyond 70 cm, the field of view can be computed by linear fitting, and Table 3.1.

NOTE: All working distance and field of view measurements are approximate only, since there is slight variation from unit to unit. When designing mechanical fixtures for mounting the Depthscan, allow for variation of up to 1 cm in these measurements from unit to unit.

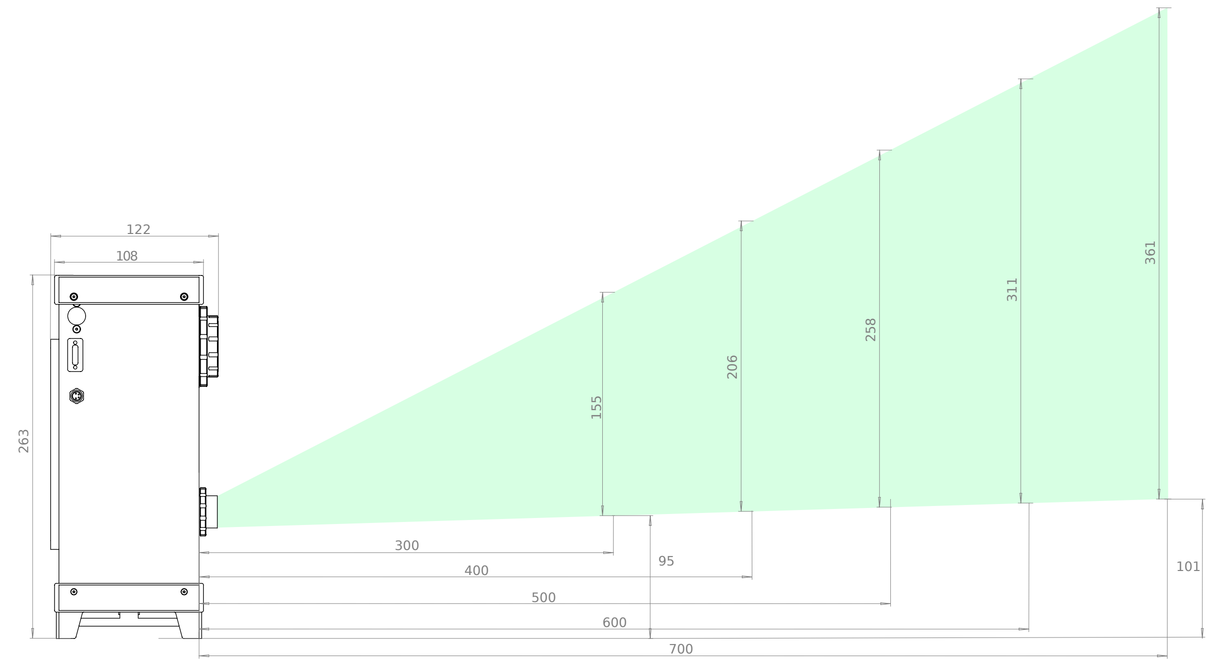
3.2 Mechanical Dimensions and Mounting

For mounting purposes, the Depthscan includes 4 x M4 mounting holes and 1 x 1/4-20 mounting hole, located at the back of the unit on its mounting plate. The 4 x M4 holes are recommended for rigidity and mechanical stability and are best suited for mounting on a gantry. The 1/4-20 hole is also included for quick and easy mounting on standard tripods. A mechanical drawing of the Depthscan and the positions of the mounting holes is shown in Figure 3.2.

NOTE (IMPORTANT!): When screwing in your bolts to the mounting plate **DO NOT EXCEED 9 mm** for the M4 holes or **13 mm** for the 1/4-20 hole, from the outside surface of the back plate, since there is not much clearance inside the unit for longer bolts to stick in. Screwing in further may cause damage to the unit!

3.3 System Cooling

The Depthscan is designed to run its 3 high powered LEDs simultaneously and continuously at full power and full duty cycle. To make this possible, there is a powerful fan housed inside the Depthscan which helps to keep all components cooled close to ambient temperature during operation.



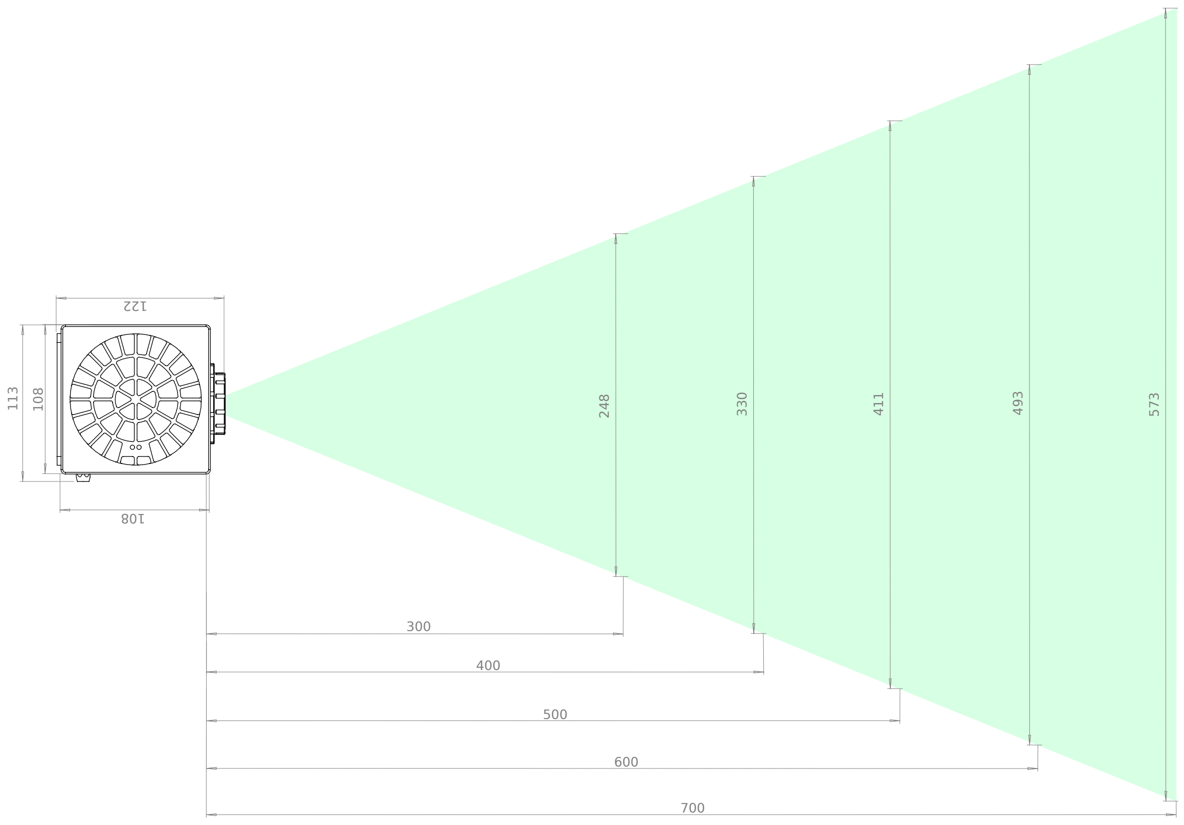


Figure 3.1: The field of view of the Depthscan at various working distances. The top image is the side view, and the bottom image is the top down view. All measurements are in millimetres (mm), and working distances are relative to the front face of the Depthscan.

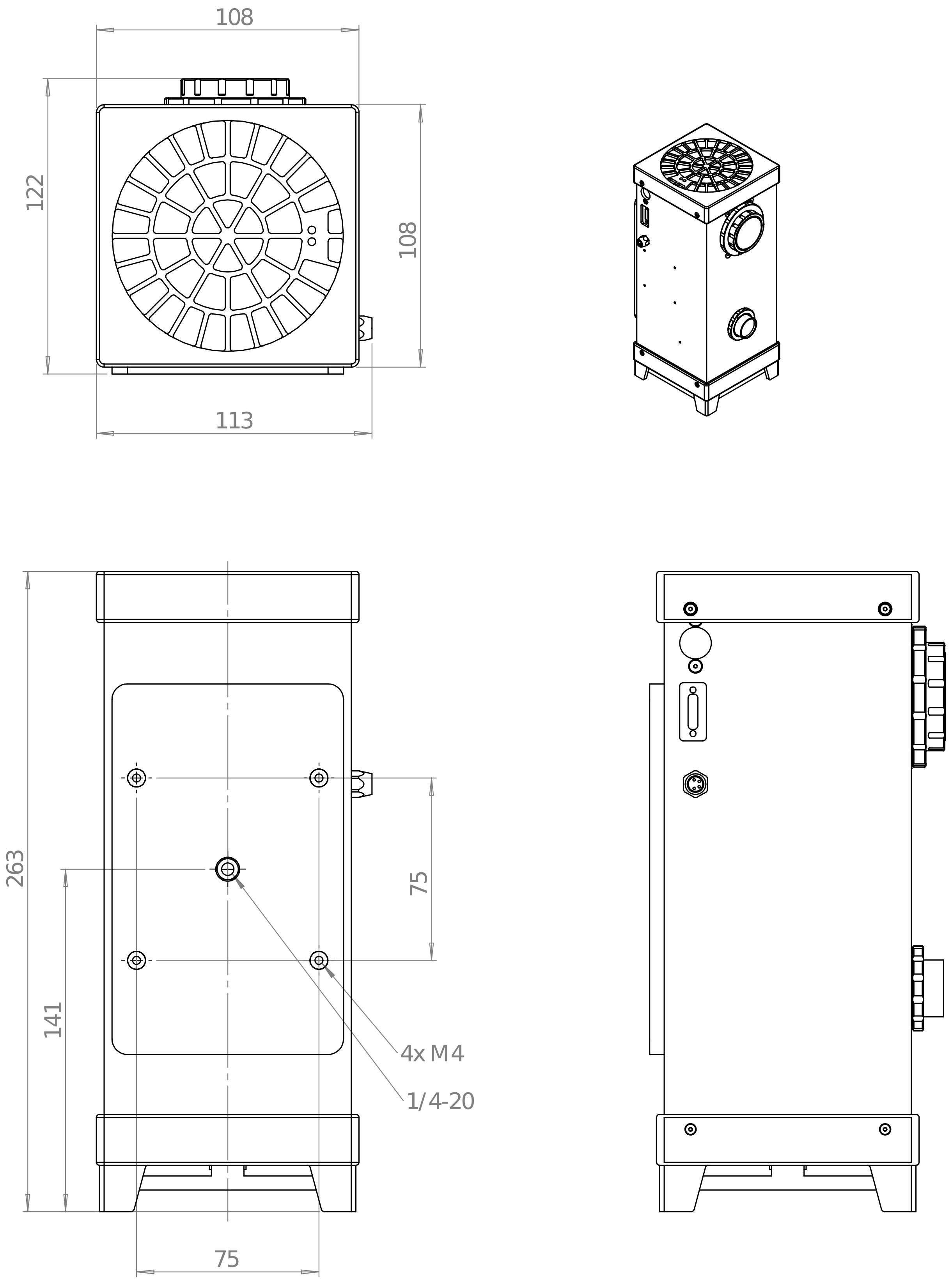


Figure 3.2: Mechanical dimensions of the Depthscan and the positions of its mounting holes. All measurements are in millimetres (mm).

Working Distance (mm)	Field of View Width (mm)	Field of View Height (mm)
300	248	155
400	330	206
500	411	258
600	493	311
700	573	361
1000	818	517
1500	1224	775
2000	1631	1034
2500	2037	1292
3000	2444	1551
3500	2850	1809
4000	3257	2068
4500	3663	2326
5000	4069	2584

Table 3.1: Field of view at various working distances, up to 5 metres.

A small amount of clearance is required at both ends of the Depthscan to ensure proper air flow through the unit. We recommend at least 2-3 cm (1 inch) of clearance at both ends. 4 'feet' are built into the base of the Depthscan at the air intake end of the unit to help prevent the intake from being block when set standing on a tabletop.

The Depthscan is rated to work safely at ambient temperates up to 40°C, but works best at temperatures below 30°C. For ambient temperatures above 40°C, an external chiller providing cool clean air directed towards the Depthscan fan intake is strongly recommended. Speak with Ajile support for advice on doing this.

3.3.1 Changing the Filter

A removeable filter comes installed with the Depthscan to prevent excess dust and debris from entering the system. This filter may be removed and cleaned or replaced as needed to ensure good air flow and cooling for the lifetime of the device. The filter fan guard un-clips from the Depthscan (shown in Figure 1.2). The replacement filter part number is Qualtek 09362-F/45 (or Qualtek 09362-M/45 which does not include the fan guard.) The filter can be purchased from electronics reseller Digikey (`https://www.digikey.com`).

3.4 Lens Care

To protect the lenses during transit or when the system is not in use, the Depthscan comes with lens caps for the camera and projector lenses. These are of a flexible plastic material, and are installed by stretching them around the lens focus rings.

The front surfaces of the camera and projector lenses on the Depthscan are exposed to the environment and therefore may be exposed to dust. While these surfaces never come into focus, dust and marks on the lenses may cause light attenuation, and in extreme cases even shadows across the captured images. Occasional cleaning may be required if degradation in image quality is noticed.

To clean the lenses, the recommended approach is to use a soft fabric cloth and a standard glass cleaning solution such as Windex. The cloth may be first dampened by cleaning solution, then gently rub the dampened cloth across the lenses, wiping away any dust or other marks such as finger prints. Do not use paper tissues, since they may shed material onto the lenses or even scratch them.

3.5 System Operation

3.5.1 Normal System Operation

When power is applied to the Depthscan, the normal system behaviour is as follows:

1. Blue power LED on the top of the unit is illuminated, indicating system power.
2. After several seconds of inactivity, a white Ajile splash screen logo image is projected onto the scene.
3. The internal fan of the Depthscan comes on briefly at full speed for 1 second, then slows down.
4. The system is now ready for operation and can be connected to over its USB 3 interface.

Power and Status LEDs

On the top of the Depthscan, when power is supplied to the unit a blue LED will be illuminated indicating the power on status. If this blue LED is not illuminated check your power supply.

Next to the blue power LED there is a red LED which should not be illuminated in normal operation. If you do notice the red LED lighting up, please contact Ajile support since this indicates excess current draw in the device which could mean that the internal device electronics are damaged.

Fan Operation

The fan speed is automatically regulated by a control loop in the Depthscan firmware. The fan speed is increased when the unit is under higher thermal load. When the Depthscan projector LEDs are active the fan speed will slowly increase to help keep them cooled. When the system is inactive the fan speed should slow down, which helps keep noise levels to a minimum. In warm ambient temperatures however the fan may always run at a reasonably high speed.

USB 3.0 Connectivity

A USB 3 Micro-B cable is needed to connect the Depthscan to a PC. Locking USB 3 cables, as per the USB3 Vision standard, are supported by the Depthscan hardware. It is highly recommended to use these locking cables (i.e. the one provided with the unit, Basler 2000033239) to prevent the cable from becoming dislodged, avoiding random data transmission errors that cause the device to stop operating.

USB 3 cable lengths of up to 3 metres have been tested with the Depthscan. For longer cable lengths it may be required to use optical cables.

External Trigger Wiring

The 4-pin M8 external trigger connector accepts a male cable such as TE Connectivity part number 1-2273002-1 (see Figure 1.4. The cable pins are aligned and the cable screws into the Depthscan trigger connector to keep it secure. The external trigger pinout and cable coloring is shown in Figure 1.5.

3.6 Hardware Troubleshooting

In case of unexpected difficulties in connecting to and controlling the Depthscan, this section describes common problems and ways to get around them. For most solvable problems however, the following solution is the most common:

Default Troubleshooting Fix: Power off the Depthscan completely by unplugging its power. Check that the USB 3 cable is securely plugged into both the Depthscan connector and to the PC. Wait at least 1 minute, then power the unit back on, and try connecting to the device again.

3.6.1 System Power

Is the Blue Power LED Illuminated?

If not, check your external power supply. We recommend using a Mean Well GS160A15-R7B power supply. Check that your power supply is delivering 15 V. If it is behaving properly, check that the power cable is properly oriented in the connector, and it is pushed in fully. When it is fully inserted and engaged it will be locked into place. If you are sure that the power is supplied correctly and the blue power LED is not illuminated, contact Ajile support.

Was the Ajile Splash Screen Logo Projected at Startup?

First, check to make sure that the lens caps for both the camera and projector are removed prior to operation. If the lens cap was installed, remove it and try power the unit off and back on.

If the Ajile splash screen does not appear with the first minute of powering on the device, it means that the projector was not detected by the system. After around 1 minute the Depthscan fan will also continuously run at full speed indefinitely. In this case, try unplugging the power to the device for 1 full minute, then plug the power back in and try again. See if the splash screen appears next time. If it still does not appear, try powering the device off for 5 minutes and try again. If the splash screen still does not appear and the fan is running at full speed, contact Ajile support.

Did the Ajile Splash Screen Logo Appear, but the fan is funning Full Speed?

If you are running in a warm environment the fan might run at high speed regardless of the system operation. However if the fan is immediately running at full speed without slowing down it could again indicate that the projector is not detected properly. Try powering down the unit for at least 1 minute, then power it back up and try again.

3.6.2 No USB 3 Connectivity

For any USB 3 connectivity issue, the first step is to simply completely power off the unit for 1 minute, then power it back on and try again.

Is No USB 3 device (labeled with Vendor ID Cypress) Visible is Device Manager?

If this is the case, try the following steps:

1. Power off the unit for at least 1 minute, check that the USB 3 cable is securely connected, then try again by powering the unit on.
2. If the problem persists, try rebooting your PC, then try again by power cycling the unit (powering off for 1 minute or more.)
3. If rebooting the PC does not help, try reinstalling the Ajile 3D software suite, rebooting the PC and the Depthscan, then try again.
4. If the above does not work, try using a different USB 3 cable, since your cable may be damaged.
5. Try using the Ajile recommended USB 3 host controller card instead of one of your built in USB 3 ports. The recommended USB 3 card is IOI U3-PCIE1XG205, and may be obtained from Ajile.
6. Try connecting to the device using a different PC, to rule out faulty hardware or software settings on the PC.

If the above steps still do not solve your problem and you cannot see a USB 3 device in your hardware devices, please contact Ajile support as it may indicate a hardware problem.

A Valid USB 3 device (with correct Cypress Vendor ID) is seen, but still cannot connect?

If the problem persists after power cycling (leaving the device power off for 1 minute), reinstall your Ajile 3D software suite, reboot your PC, and try again.

3.6.3 Imaging Problems

Camera does not start in Aperture and Focus Setup?

1. Try power cycling the device and try again.
2. If you still cannot get a live image from the camera, check to make sure your lens caps are removed.
3. Place a light colored object in front of the unit, it should be visible.

If you still do not get a live image from the camera in the Aperature and Focus Setup window, it could indicate a hardware issue so contact Ajile support.

Can successfully capture a 3D image, but there are no visible points?

1. Did you focus the camera and projector and set the aperture properly. See section 2.4 for details.
2. Is the object being imaged very dark, or completely black? Try increasing your aperature size or exposure. Try changing from color view to depth view in the 3D viewer. Try changing the 3D viewer background color to a lighter color.
3. Try testing 3D image capture with a more simple object, such as a white wall at 50 cm working distance, and check to see if you get valid points measured.
4. Try restarting the GUI, and possibly even power cycling the unit, since certain imaging or display settings could have been inadvertantly corrupted.

Chapter 4

3D Imaging Capabilities

In this chapter we will explore the advanced imaging capabilities of the DepthScan. Each section will describe a specific imaging capability, when to use it and how to use it in the Ajile 3D GUI as well as the SDK. Most of the imaging capabilities are available in the 3D GUI in the settings dock. The settings dock can be accessed in the Ajile 3D GUI by pressing the settings icon on the capture tool bar or it will be automatically displayed after connecting to a device. The advanced settings section in the settings dock will contain all the imaging capabilities described in this chapter. The settings dock is highlighted in Figure 4.1 along with the settings icon button and advanced settings section.

4.1 Camera Exposure Time

The camera exposure time dictates the amount of light that enters the camera. The camera exposure time and openness of the camera aperature can be used interchangeably to control camera image saturation of the capture result. Camera exposure time offers a dynamic and programatic way of changing the saturation of the output. It is useful to adjust the camera exposure time when imaging reflective (lower) or matte objects (raise). The camera exposure time also determines the total time of the capture - exposure time should be as small as possible for the fastest capture speed.

Camera exposure time can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

Setting the exposure time in the SDK is simple, listing 4.1 in C++ and Listing 4.2 in Python illustrate how.

```
// Setting exposure time to 10 ms
ajile3DImager.SetExposureTime(10);
```

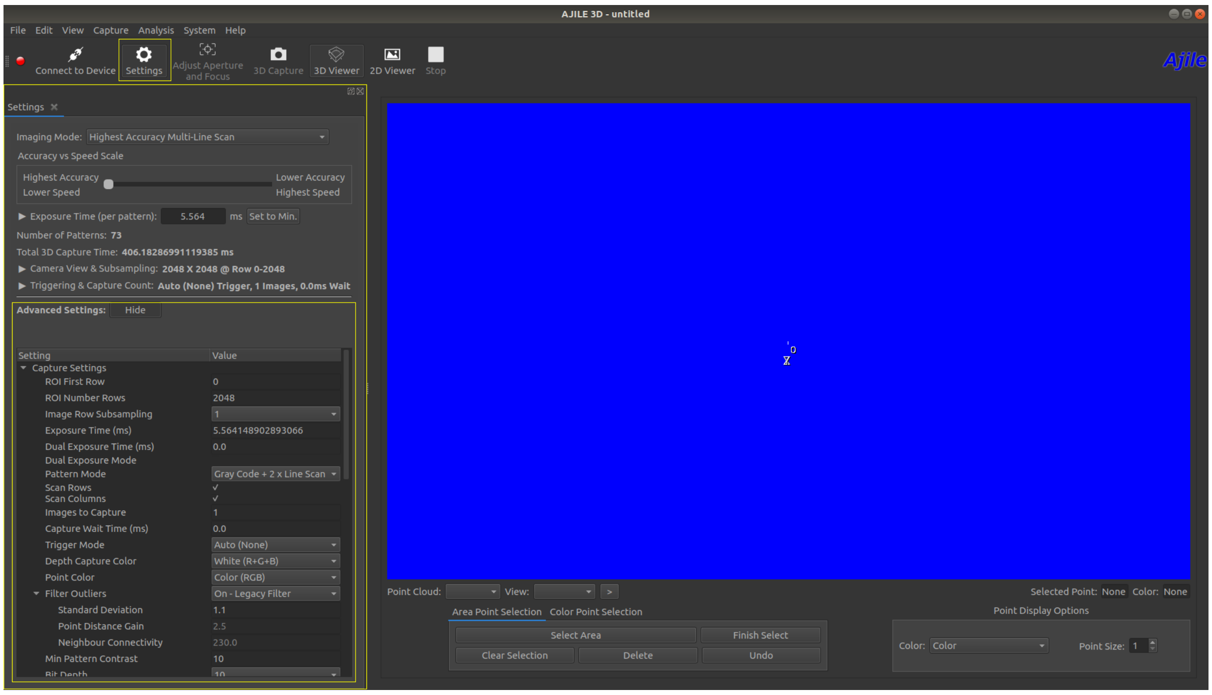
Listing 4.1: C++ example of setting exposure time.

```
# Setting exposure time to 10 ms
ajile3DImager.SetExposureTime(10)
```

Listing 4.2: Python example of setting exposure time.

4.2 Dual Exposure

When capturing a scene, different areas or objects in the scene may require different exposure times for optimal imaging. Dual exposure allows the user to accomplish this by specifying a second exposure time and combining the results from two different exposures. This is useful for difficult to image objects in which a long and short exposure time can be used in tandem to capture a high quality point cloud. By setting a dual exposure time, the Depthscan will run all patterns for the original exposure time then



Settings

Imaging Mode: Highest Accuracy Multi-Line Scan

Accuracy vs Speed Scale

Highest Accuracy / Lower Speed — Lower Accuracy / Highest Speed

▶ Exposure Time (per pattern): 5.564 ms Set to Min.

Number of Patterns: **73**

Total 3D Capture Time: **406.18286991119385 ms**

▶ Camera View & Subsampling: **2048 X 2048 @ Row 0-2048**

▶ Triggering & Capture Count: **Auto (None) Trigger, 1 Images, 0.0ms Wait**

Advanced Settings: Hide

Setting	Value
Capture Settings	
ROI First Row	0
ROI Number Rows	2048
Image Row Subsampling	1
Exposure Time (ms)	5.564148902893066
Dual Exposure Time (ms)	0.0
Dual Exposure Mode	
Pattern Mode	Gray Code + 2 x Line Scan
Scan Rows	✓
Scan Columns	✓
Images to Capture	1
Capture Wait Time (ms)	0.0
Trigger Mode	Auto (None)
Depth Capture Color	White (R+G+B)
Point Color	Color (RGB)
Filter Outliers	On - Legacy Filter
Standard Deviation	1.1
Point Distance Gain	2.5
Neighbour Connectivity	230.0
Min Pattern Contrast	10
Bit Depth	10

Figure 4.1: Settings Dock in the Ajile 3D GUI.

a second time for the dual exposure. Ajile software will then process the images and create a capture result with the best available data between the two exposure times.

Dual exposure time can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

The following code snippets exemplify how to set the dual exposure time in C++ and Python. The ajile3dImager is an Ajile3DImager object from the SDK. Listing 4.3 provides an example for C++ and Listing 4.4 provides an example for Python.

```cpp
// Setting dual exposure time to 20 ms
ajile3DImager.SetDualExposure(20);
```

Listing 4.3: C++ example of setting dual exposure time.

```python
# Setting dual exposure time to 20 ms
ajile3DImager.SetDualExposure(20)
```

Listing 4.4: Python example of setting dual exposure time.

4.3 Structured Light Pattern Types

Generating a 3D point cloud with a DepthScan is accomplished by projecting patterns and analyzing them. The pattern types built-in to the Ajile software include Graycode, Linescan and Double Linescan. Choosing the pattern type for a capture depends on the speed required for the task. The Doule Linescan pattern type provides the most accurate capture result. The Graycode pattern type sacrifices accuracy for speed and cuts the number of patterns from 73 to 48. This makes the Graycode pattern type useful for high framerate imaging and fast imaging response times.

The pattern type can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1. A basic option in the settings dock is the Accurary-Speed slider. The Accuracy-Speed slider allows the user to choose between high resolution accurate or high framerate outputs.

In the SDK, changing the pattern type can be done using the Ajile3DImager object. Listing 4.5 provides an example for C++ and Listing 4.6 provides an example for Python.

```
// Setting pattern type to Graycode
ajile3DImager.SetPatternMode(aj3d::GRAYCODE_PATTERN_MODE);
```

Listing 4.5: C++ example of setting pattern type.

```
# Setting pattern type to Graycode
ajile3DImager.SetPatternMode(ajile3d.GRAYCODE_PATTERN_MODE)
```

Listing 4.6: Python example of setting pattern type.

The available pattern types are available as an enum and are as follows:

```
typedef enum PatternModeEnum {
    UNDEFINED_PATTERN_MODE=0,
    GRAYCODE_PATTERN_MODE=1,
    LINESCAN_PATTERN_MODE=2,
    DOUBLE_LINESCAN_PATTERN_MODE=3,
    DEBRUIJN_PATTERN_MODE=4,
```

Listing 4.7: Pattern Type Enum.

4.4 Region of Interest

The region of interest is defined by the first camera row and the number of camera rows. The specified region is captured by the camera instead of exposing every pixel. This can increase processing speeds and allow for rapid imaging. Currently, region of interest mode is constrained to only reducing the number of rows exposed and no column limits can be set.

Region of interest can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

In the SDK, the region of interest can be manipulating according to the example in Listing 4.8 for C++ and Listing 4.9 for Python.

```
// Setting region of interest to start at row 256 and go for 1024 rows
ajile3DImager.SetRegionOfInterest(256, 1024);
```

Listing 4.8: C++ example of setting region of interest.

```
# Setting region of interest to start at row 256 and go for 1024 rows
ajile3DImager.SetRegionOfInterest(256, 1024)
```

Listing 4.9: Python example of setting region of interest.

4.5 Row Subsampling

The row subsampling provides the user with another option to reduced the number of rows captured along with the region of interest. Row subsampling determines what portion of rows are skipped. Valid row subsampling values are 1, 2 or 4. A value of 1 represents no row subsampling while 4 represents processing every 4th row. Row subsampling is useful for high fps capturing and is used in the Ajile 3D GUI in high speed imaging modes.

Row subsampling can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

Row subsampling is set in the SDK according to Listing 4.10 and Listing 4.11 in C++ and Python respectively.

```
// Setting row subsampling to every second row
ajile3DImager.SetRowSubsampling(2);
```

Listing 4.10: C++ example of setting row subsampling.

```
# Setting row subsampling to every second row
ajile3DImager.SetRowSubsampling(2)
```

Listing 4.11: Python example of setting row subsampling.

4.6 Capture Color

Capture color is defined by the point color and structured light capture color. The point color determines the output color channels while the structured light capture color affects which LEDs are used during pattern projection.

4.6.1 Point Color

Point color determines the color type of points in the capture result. This can change the patterns projected depending on other capture proprties and will change the color image and point cloud colors. The following enum lists the available point color types:

```
} TriggerMode_e;

typedef enum CaptureColorEnum {
    UNDEFINED_CAPTURE_COLOR_TYPE=0,
    RED_CAPTURE_COLOR_TYPE=1,
    GREEN_CAPTURE_COLOR_TYPE=2,
    BLUE_CAPTURE_COLOR_TYPE=3,
```

Listing 4.12: Point Color Type Enum.

The RGB point color type is the default color type. The grayscale point color type will output a graytone image without color. Binary contains no color information and each pixel is represented in a binary fashion.

Point color can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

The point color type can be changed in the SDK by following the examples in Listing 4.13 and Listing 4.14.

```
// Setting point color to grayscale
ajile3DImager.SetPointColorType(aj3d::GRAYSCALE_POINT_COLOR_TYPE);
```

Listing 4.13: C++ example of setting point color type.

```
# Setting point color to grayscale
ajile3DImager.SetPointColorType(ajile3d.GRAYSCALE_POINT_COLOR_TYPE)
```

Listing 4.14: Python example of setting point color type.

4.6.2 Structured Light Capture Color

Structured light capture color type determines which LEDs are used during pattern projection. This will affect the color of the capture result directly. The three LED colors are red, green and blue and thus the following enum defines the available capture color types:

```
} TriggerMode_e;

typedef enum CaptureColorEnum {
    UNDEFINED_CAPTURE_COLOR_TYPE=0,
    RED_CAPTURE_COLOR_TYPE=1,
    GREEN_CAPTURE_COLOR_TYPE=2,
    BLUE_CAPTURE_COLOR_TYPE=3,
```

Listing 4.15: Capture Color Type Enum.

The default capture color type is white which means all three LEDs are turned on. For the other available capture color types, the corresponding LED color is the only LED enabled during the capture.

Capture color can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

To change the capture color in the SDK follow the C++ example in Listing 4.16 or the Python example in Listing 4.17.

```
// Setting capture color to red
ajile3DImager.SetCaptureColorType(aj3d::RED_CAPTURE_COLOR_TYPE);
```

Listing 4.16: C++ example of setting capture color type.

```
# Setting capture color to red
ajile3DImager.SetCaptureColorType(ajile3d.RED_CAPTURE_COLOR_TYPE)
```

Listing 4.17: Python example of setting capture color type.

4.7 Triggering and Synchronization

The DepthScan provides multiple ways to time and synchronize imaging processes. The DepthScan is equipped with 2 input trigger channels and 2 output trigger channels. These triggers can be used to synchronize processes with the DepthScan and initiate captures using an external change in voltage.

The following enum describes the possible trigger modes that will be discussed later in this section:

```
} PatternMode_e;

typedef enum TriggerModeEnum {
    UNDEFINED_TRIGGER_TYPE=0,
    AUTO_TRIGGER=1,
    EXTERNAL_HW_TRIGGER=2,
```

Listing 4.18: Trigger Mode Enum.

Triggering options can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

4.7.1 Auto Trigger (Default)

The default case automatically starts the capture without waiting for any trigger.

4.7.2 Internal Triggering

Internal triggering waits for an internal software trigger to initiate the capture. Until the software trigger is fired, the DepthScan will idle with all frames loaded. This mode is most useful when used in conjunction with multiple depth images to capture as each trigger will only initiate depth image capture.

4.7.3 External Triggering

External triggering is available as rising or falling edges as inputs or outputs.

Input Triggers

The DepthScan is configured to react to a falling edge as an input. If the trigger mode is set to external and a capture is started, the DepthScan will wait for a falling edge on the input pin before capturing. The external triggering capability gives the DepthScan flexibility in integrating with other systems.

Output Triggers

An output trigger can be configured to be sent at the end of each capture sequence to enable synchronization with other DepthScans or system parts.

4.8 Depth Images To Capture

The depth images to capture property enables the user to capture multiple point clouds using the same capture properties. An advantage to the depth images to capture property its compatibility with internal and external triggering. Each trigger will initiate only one capture sequence. This allows the user to queue multiple captures and send triggers to initiate each capture. The number of depth images to capture must be a multiple of 2 or be the value 1. Setting the depth images to capture to 0 will cause the DepthScan to capture continuously until it is stopped. If dual exposure mode is enabled, the depth images to capture must be doubled to account for the second exposure.

Depth images to capture can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

Listing 4.19 provides a C++ example of changing this value in the SDK and Listing 4.20 provides a Python example.

```cpp
// Setting depth images to capture to 4
ajile3DImager.SetDepthImagesToCapture(4);
```

Listing 4.19: C++ example of setting depth images to capture.

```python
# Setting depth images to capture to 4
ajile3DImager.SetDepthImagesToCapture(4)
```

Listing 4.20: Python example of setting depth images to capture.

4.9 Capture Wait Time

The capture wait time dictates the amount of time between captures that the DepthScan will idle. The capture wait time also influences the capture rate as it increases the amount of time required for each capture. The capture rate is the amount of time the patterns are projected plus the capture wait time. Precisely, the capture wait time is the amount of time between the end of the last capture frame and the beginning of the next first capture frame while the capture rate is the time between the first capture frame and the next first capture frame. This holds true with internal or external triggering enabled.

Capture wait time can be adjusted in the Ajile 3D GUI using the settings dock or the advanced settings section seen in Figure 4.1.

The capture wait time can be changed in the SDK by following the provided examples in Listing 4.21 (in C++) and Listing 4.22 (in Python).

```cpp
// Setting Capture Wait Time to 100 ms
ajile3DImager.SetCaptureWaitTime(100);
// Setting Capture Rate to 1 capture per 1000ms
ajile3DImager.SetCaptureRateMSec(1000);
```

Listing 4.21: C++ example of setting capture wait time.

```
# Setting Capture Wait Time to 100 ms
ajile3DImager.SetCaptureWaitTime(100)
# Setting Capture Rate to 1 capture per 1000ms
ajile3DImager.SetCaptureRateMSec(1000)
```

Listing 4.22: Python example of setting capture wait time.

4.10 Default LED Current

The default LED current can be changed for each of the red, green and blue LEDs. To use the factory default for an LED, the LED current for the channel must be set to -1. The LED current affects the color bance of the capture result. For example, increasing the red LED current (channel 0) would cause a red shift in color of the resulting point cloud.

LED currents can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

The default LED currents can be changed in the SDK by following the provided examples in Listing 4.23 and Listing 4.24 (C++ and Python respectively).

```
// Setting the red LED current to 500 mA
ajile3DImager.SetDefaultLedCurrent(0, 500);
```

Listing 4.23: C++ example of setting default LED current.

```
# Setting the red LED current to 500 mA
ajile3DImager.SetDefaultLedCurrent(0, 500)
```

Listing 4.24: Python example of setting default LED current.

4.11 Color Exposure Gain

The color exposure gain affects the contribution of LED channel color to the final capture result color. Color gain has a default value that is embedded into a calibration file for each individual DepthScan. This default value has been determined by calibrating the individual DepthScan against a calibrated color board to provide the optimal color balance for a range of colors. If the color exposure gain value of a channel is -1, this will force the software to use the optimized color exposure gain from the calibration file. The color exposure gain is applied before a histogram equalization process that normalizes the color values.

Color exposure gain can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

The color exposure gain can be set in the SDK as done in Listing 4.25 for C++ and Listing 4.26 for Python.

```
// Setting blue color gain to 0.8
ajile3DImager.SetColorExposureGain(2, 0.8);
```

Listing 4.25: C++ example of setting color exposure gain.

```
# Setting blue color gain to 0.8
ajile3DImager.SetColorExposureGain(2, 0.8)
```

Listing 4.26: Python example of setting color exposure gain.

4.12 Pixel Coordinate System

The pixel coordinate system determines the coordinates of 3D and 2D outputs. The camera has a pixel coordinate system which is 2048 by 2048 while the projector is 2280 by 1824. The default pixel coordinate system is projector. The Listing 4.27 defines the options for pixel coordinate system:

```
} USBError_e;

typedef enum PixelCoordinateSystemEnum {
    DEFAULT_PIXEL_COORDINATE_SYSTEM=0,
    PROJECTOR_PIXEL_COORDINATE_SYSTEM=1,
```

Listing 4.27: Pixel Coordinate System Enum.

Pixel coordinates system can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

The pixel coordinate sytem can be changed in the SDK following the examples in Listing 4.28 and Listing 4.29.

```
// Setting pixel coordinate system to camera
ajile3DImager.SetPixelCoordinateSystem(aj3d::CAMERA_PIXEL_COORDINATE_SYSTEM);
```

Listing 4.28: C++ example of setting pixel coordinate system.

```
# Setting pixel coordinate system to camera
ajile3DImager.SetPixelCoordinateSystem(ajile3d.CAMERA_PIXEL_COORDINATE_SYSTEM)
```

Listing 4.29: Python example of setting pixel coordinate system.

4.13 Output Transformation Matrix

The output transformation matrix defines a transform that will be applied to all output point clouds. Any capture result retrieved from an Ajile3DImager object will have the transformation applied after processing if the Output Transformation Matrix is properly set in CaptureProperties. The transformation matrix is a flat vector representation of a 4x4 transformation matrix. A 4x4 matrix can represent all possible affine transformations such as translation, rotation, reflection and scale.

Pixel coordinates system can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

Setting the output transformation matrix in the SDK is shown in Listing 4.30 in C++ and Listing 4.31 in Python.

```
// Setting the output transformation matrix
std::vector<float> matrix { 1, 0, 0, 50, // translating 50 in positive x direction
                            0, 1, 0, 0,
                            0, 0, 1, 0,
                            0, 0, 0, 1};
ajile3DImager.SetOutputTransformationMatrix(matrix);
```

Listing 4.30: C++ example of setting output transformation matrix.

```
# Setting the output transformation matrix
matrix = ajile3d.FloatList()
matrix.append(1); matrix.append(0); matrix.append(0); matrix.append(50) # translating 50 in positive x direction
matrix.append(0); matrix.append(1); matrix.append(0); matrix.append(0)
matrix.append(0); matrix.append(0); matrix.append(1); matrix.append(0)
matrix.append(0); matrix.append(0); matrix.append(0); matrix.append(1)
ajile3DImager.SetOutputTransformationMatrix(matrix)
```

Listing 4.31: Python example of output transformation matrix.

4.14 Communication Error

In the event of a communication error, the DepthScan will undergo a software reset in an attempt to reconnect. The response during a reconnect depends on the behavior selected in software. These responses only apply to a communication error that occurs during a capture, otherwise the idle system will reconnect once the communication error has resolved. In the case of a live view, the live view will always be restarted. The following enumeration in Listing 4.32 outlines the possible responses during a communication error.

```

//! List of behaviors for a timeout
typedef enum TimeoutBehavior {
    AUTOMATIC = 0,
    RESTART = 1,
```

Listing 4.32: Timeout Behavior Enum.

The RESTART behavior forces a capture to be restarted once the communcation is restored. The ABORT behavior forces the capture to stop and no capture result will be created. The AUTOMATIC behavior will restart the capture if the timeout time has not been reached and will force the capture to stop if the timeout time has been reached. The timeout time can be manually set or it will use the timeout time used in a call of WaitForCaptureResult().

Communication error response can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

The examples in Listing 4.33 in C++ and Listing 4.34 in Python describe how to set the timeout behavior and the timeout time.

```
// Setting the timeout behavior
ajile3DImager.SetTimeoutBehavior(aj3d::AUTOMATIC); // setting timeout behavior to automatic
ajile3DImager.SetTimeoutTime(20000); // setting to 20000ms or 20s
```

Listing 4.33: C++ example of setting timeout behavior.

```
# Setting the timeout behavior
ajile3DImager.SetTimeoutBehavior(ajile3d.AUTOMATIC) # setting timeout behavior to automatic
ajile3DImager.SetTimeoutTime(20000) # setting to 20000ms or 20s
```

Listing 4.34: Python example of timeout behavior.

4.15 Capture Queue Behavior

The capture queue contains all captured images directly from the camera. Depending on capture properties, the number of captured images per capture result (frames per depth image) will vary. The available RAM on the processing machine will ditacte the number of images that can be stored at any given moment. If processing is slower than capturing, this can lead to a build-up of images in the capture queue. The maximum capture queue size can be set to prevent captured images from consuming all available memory on the machine. The response to a full capture queue can be set based on the enum seen in Listing 4.35.

```
    STORE_CALIBRATION = 8,
    GET_CALIBRATION = 9,
} TurntableCommands_e;

typedef enum TurntableResponsesEnum {
    STATE = 128,
    MOVEMENT_DONE = 129,
    POSITION = 130,
```

Listing 4.35: Capture Queue Behavior Enum.

The REMOVE_FRONT behavior will remove an entire capture's worth of images from the front of the capture queue. This will cause older captures to be removed from the queue. The REMOVE_BACK behavior will remove an entire capture's worth of images from the back of the capture queue. This will cause newer captures to be removed from the queue. The EMPTY behavior will cause all images to be removed from the queue. The WAIT behavior will pause capturing until until space is available in the capture queue. This will cause the DepthScan to stop capturing but no captures will be lost. The STOP behavior will cause no images to be removed from the queue but the DepthScan will be stopped and not restarted.

Capture queue behavior can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

Examples of setting the capture queue behavior in C++ and Python can be seen in Listing 4.36 and Listing 4.37 respectively.

```
// Setting the capture queue behavior
ajile3DImager.SetCaptureQueueMaxSize(30*(ajile3DImager.GetCaptureProperties().FramesPerDepthImage()-(
  ajile3DImager.GetCaptureProperties().NumGrayCodeImages()/2))); // setting the maximum size to 30 captures
  worth of images
ajile3DImager.SetCaptureQueueFullBehavior(aj3d::EMPTY); // setting the capture queue behavior to empty all
  images when full
```

Listing 4.36: C++ example of setting capture queue behavior.

```
# Setting the capture queue behavior
ajile3DImager.SetCaptureQueueMaxSize(int(30 * (ajile3DImager.GetCaptureProperties().FramesPerDepthImage() -
   (ajile3DImager.GetCaptureProperties().NumGrayCodeImages() / 2)))) # setting the maximum size to 30 captures
   worth of images
ajile3DImager.SetCaptureQueueFullBehavior(ajile3d.EMPTY) # setting the capture queue behavior to empty all
  images when full
```

Listing 4.37: Python example of capture queue behavior.

4.16 Outlier Filtering

Noise removal is an important part of 3D imaging and can lead to large amounts of outliers in point clouds. Ajile software has the ability to filter outliers based on two different methods. The default method, known as the legacy method, filters outliers based on the nearest neighbours of each point. The legacy method uses a provided standard deviation value to determine if a point is too far away from its neighbours and deems it an outlier. The second method is depth based. The depth based method removes points that are not the correct depth by observing points surronding it. This method uses point distance gain and neighbour connectivity as inputs.

Outlier filtering can be adjusted in the Ajile 3D GUI using the advanced settings section seen in Figure 4.1.

Outlier filtering is by default enabled. The examples in Listing 4.38 and 4.39 show how to switch between methods and change parameter values in the SDK.

```
// Setting outlier filtering
ajile3DImager.SetFilterOutliersDepthBased(true, 3, 250); // turn on depth based filtering
ajile3DImager.SetFilterOutliersStdDev(false); // turn off legacy based filtering
```

Listing 4.38: C++ example of setting outlier filtering.

```
# Setting outlier filtering
ajile3DImager.SetFilterOutliersDepthBased(True, 3, 250) # turn on depth based filtering
ajile3DImager.SetFilterOutliersStdDev(False) # turn off legacy based filtering
```

Listing 4.39: Python example of setting outlier filtering.

4.17 Output Formats

Capture results contain multiple output formats. These formats can be 2D or 3D with the main focus on the 3D point cloud. Each of the formats can be retrieved, saved and manipulated for further processing or analysis. The following sections describe the different outputs of the capture result. Retrieving a capture result from an Ajile3DImager object can be achieved by following the Ajile3DImagerExample in C++ or Python.

All output formats can be generated, autosaved and displayed in the Ajile 3D GUI by adjusting options in the advanced settings section seen in Figure 4.1.

4.17.1 3D Output Formats

Point Cloud Format

Point clouds are the main output of a DepthScan. The point cloud output format is a set of 3D data points in space. By default, each point has a x, y and z position as well as a color divided into red, green and blue. The point clouds are organized based on the current pixel coordinate system, which by default is projector. In the projector pixel coordinate system, the height will be 2280 and width will be 1824 leading to 4,158,720 points total. Point clouds are the basis for further 3D analysis such as primitive fitting, CAD comparison, stiching and more.

To retrieve a point cloud from a capture result in the SDK, follow the examples in Listing 4.40 in C++ and Listing 4.41 in Python.

```cpp
// Retrieving point cloud from capture result
aj3d::AJPointCloud pointCloud = captureResult.PointCloud();
```

Listing 4.40: C++ example of retrieving point cloud.

```python
# Retrieving point cloud from capture result
pointCloud = captureResult.PointCloud()
```

Listing 4.41: Python example of retrieving point cloud.

Depth Map Format

The depth map is a 2D representation of 3D data. The organization of the depth map represents x and y values while the value at each pixel represents the z value. The depth map is a very useful output for processing 3D images using 2D image tools. For this reason the depth map is a 1824 x 2280 32 bit float image. Many common file formats do not store 32 bit float pixel values. This means when saving depth map data in these file formats, the z value is normalized to fit within the range of the pixel bit depth. The tiff image format allows users to save depth map data as 32 bit float resulting in no data loss.

To retrieve a depth map from a capture result in the SDK, follow the examples in Listing 4.42 in C++ and Listing 4.43 in Python.

```cpp
// Retrieving depth map from capture result
aj3d::ImageData depthMap = captureResult.DepthMap();
```

Listing 4.42: C++ example of retrieving depth map.

```python
# Retrieving depth map from capture result
depthMap = captureResult.DepthMap()
```

Listing 4.43: Python example of retrieving depth map.

4.17.2 Exporting Point Clouds

The point clouds can be saved to a PCD file format. The PCD format is the most flexible format for storing point cloud data as any number of fields can be saved as 3D data. The Ajile SDK saves point clouds with X, Y, Z coordinate and R, G, B color data. The alternative file format is PLY, a polygonal format which stores data as a list of flat polygons.

4.17.3 2D Output Formats

The 2D output formats include the raw images and a process color image. The following sections will explain how to retrieve and export these outputs.

Color Image Format

The color image is a 24 bit rgb image which is output in the coordinate system of the point cloud output, known as the pixel coordinate system. The color image is a processed output and therefore will have undergone color equalization and adjustment. The output is affected by other imaging parameters such as the LED current and color gain of each channel.

To retrieve a color image from a capture result in the SDK, follow the examples in Listing 4.44 in C++ and Listing 4.45 in Python.

```
// Retrieving color image from capture result
aj3d::ImageData colorImage = captureResult.ColorImage();
```

Listing 4.44: C++ example of retrieving color image.

```
# Retrieving color image from capture result
colorImage = captureResult.ColorImage()
```

Listing 4.45: Python example of retrieving color image.

Raw Image Format

The raw images used to generate the 3D output are available in the SDK and 3D GUI.

To retrieve raw images from a capture result in the SDK, follow the examples in Listing 4.46 in C++ and Listing 4.47 in Python.

```
// Retrieving raw images from capture result
std::vector<aj3d::ImageData> rawImages = captureResult.StructuredLightImages();
```

Listing 4.46: C++ example of retrieving raw images.

```
# Retrieving raw images from capture result
rawImages = captureResult.StructuredLightImages()
```

Listing 4.47: Python example of retrieving raw images.

Chapter 5

Ajile 3D SDK

5.1 Ajile 3D Imaging SDK Overview

The Ajile 3D Imaging SDK offers a programmatic way to control the DepthScan to capture 3D images and process the results. It enables advanced features in the DepthScan and allows integrating with user programs and tools.

The Ajile 3D Imaging SDK is available for both Windows and Linux and has support for C++ and Python. Internally the SDK is C++ based and the Python interface is generated directly from it.

The Ajile 3D Imaging SDK is object oriented and has a number of objects which allow controlling the DepthScan and working with the captured results. The main objects that we will be working with in this chapter are `Ajile3DImager` which is responsible for controlling the DepthScan and capturing images, and `CaptureResult` which is output by `Ajile3DImager` when a new 3D image is captured and contains the 3D point cloud and depth map data.

5.2 Detailed SDK Documentation and Examples

In addition to this chapter which describes the SDK, two other main sources of detailed information on the SDK are available with the Ajile 3D Software installation, which are the reference documentation and the example programs.

The easiest way to see all the documentation that comes with the software suite is to open the Ajile 3D GUI, then click on the 'System' menu, then 'Help'. This causes the default system web browser to open the Ajile 3D documentation landing page, which is shown in Figure 5.1. This page has links to this users guide, the example programs for the SDK, and the C++ and Python class and function reference manuals.

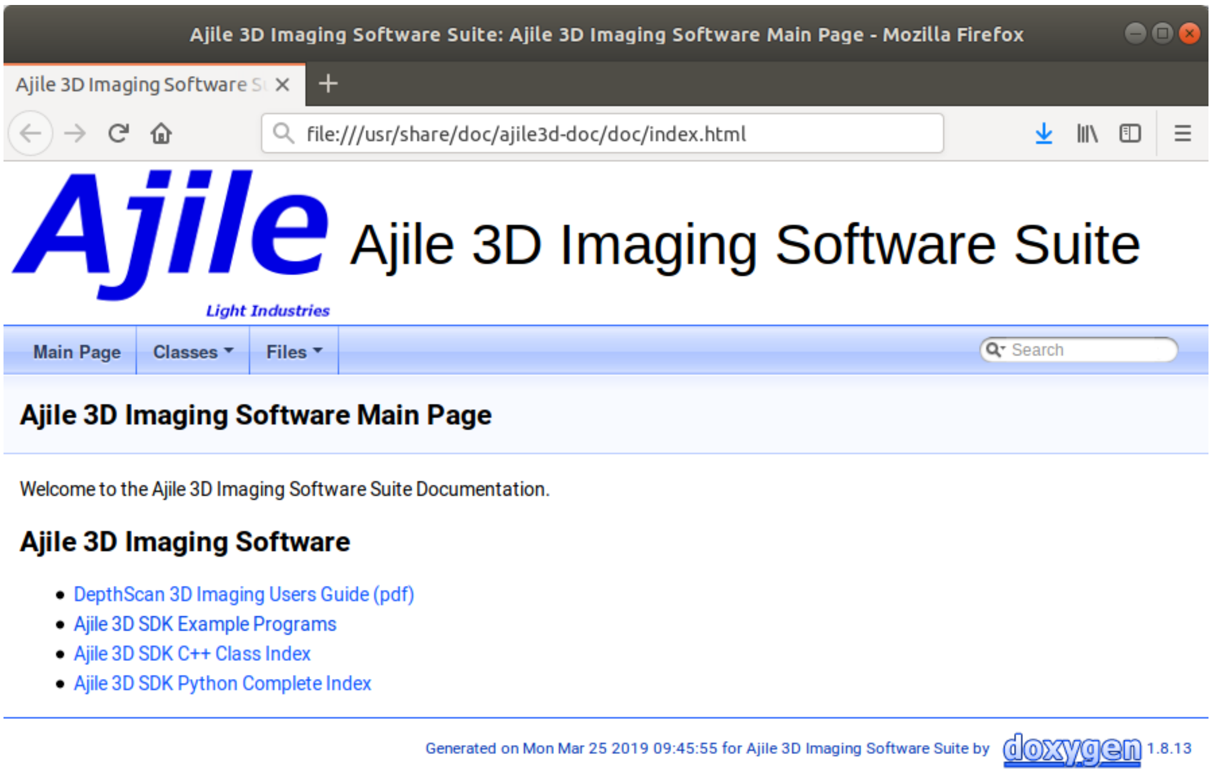
Alternately, the documentation and examples can be found in Windows under the Documentation directory of the installation (i.e. C:\Program Files (x86)\Ajile\Ajile3D\Documentation), or in Linux under /usr/share/doc/ajile3d-doc/.

5.3 Capturing 3D Images

As mentioned above, `Ajile3DImager` is the class that will be used to capture 3D images.

5.3.1 System Initialization and Configuration

To use the `Ajile3DImager` object, we first need to include the necessary modules for Python or header files for C++. This is shown in Listing 5.1 and 5.2. Note that we are also including the OpenCV library here, and in Python the NumPy library as well. These libraries will be used later to display images


Ajile 3D Imaging Software Suite: Ajile 3D Imaging Software Main Page - Mozilla Firefox

file:///usr/share/doc/ajile3d-doc/doc/index.html

Ajile 3D Imaging Software Suite

Main Page | Classes | Files

Ajile 3D Imaging Software Main Page

Welcome to the Ajile 3D Imaging Software Suite Documentation.

Ajile 3D Imaging Software

- DepthScan 3D Imaging Users Guide (pdf)
- Ajile 3D SDK Example Programs
- Ajile 3D SDK C++ Class Index
- Ajile 3D SDK Python Complete Index

Generated on Mon Mar 25 2019 09:45:55 for Ajile 3D Imaging Software Suite by doxygen 1.8.13

Figure 5.1: Ajile 3D Imaging Software documentation landing page.

on the screen, however they are optional and could easily be removed and replaced by your own image handling code.

```python
import ajile3d

import sys

def RunExample():
```

Listing 5.1: Python modules which will be used with the Ajile 3D SDK.

```cpp
#include <ajile3d/Ajile3DImager.h>
#include <ajile3d/Viewer3D.h>

#include <stdio.h>
```

Listing 5.2: C++ header files which will be used with the Ajile 3D SDK.

With the necessary modules/headers included, we can now instantiate an `Ajile3DImager` object. This is shown for Python and C++ in Listing 5.3 and 5.4.

```python
    # set the exposure time in milliseconds.
```

Listing 5.3: Instantiating an `Ajile3DImager` object in Python.

```cpp

    // read in command line arguments
```

Listing 5.4: Instantiating an `Ajile3DImager` object in C++.

There are numerous options and settings which can be configured `Ajile3DImager` to change the imaging properties. For now in this example we will simply show how to change the exposure time from the default to 8 milliseconds, which is shown in Listing 5.5 and 5.6.

```python
    ajile3DImager.SetExposureTime(8)
```

Listing 5.5: Setting exposure time in Python.

```cpp
    for (int i=1; i<argc; i++) {
        if (strcmp(argv[i], "--exposure") == 0 || strcmp(argv[i], "-e") == 0) {
```

Listing 5.6: Setting exposure time in C++.

Finally, we must connect to the device to proceed. We do so with the `StartSystem()` function, which return 0 on success or non-zero if an error occurred, as shown in Listing 5.7 and 5.8.

```python
    ret = ajile3DImager.StartSystem()
    if ret != 0:
        print("Error connecting.")
        sys.exit(-1)
```

Listing 5.7: Connecting to the DepthScan in Python.

```cpp
        return -1;
    }

    // now we start setup and start reading live images from the camera
    printf("Starting Live Preview - Select the Ajile Camera Image window and press any key to stop live preview and
     perform a 3D capture.\n");
```

Listing 5.8: Connecting to the DepthScan in C++.

5.3.2 Setup Mode

As was discussed in Chapter 2, the first step to capturing high quality 3D images is to first focus the projector and camera of the DepthScan. To do this in the SDK, the function `Ajile3DImager.StartSetup()` is used to put the DepthScan into setup mode where a repeating line pattern is projected and the camera continuously reads out images of the scene. This is shown in Listing 5.9 and 5.10.

```
```

Listing 5.9: Starting setup mode in Python.

```
while (keyPress < 0) {
```

Listing 5.10: Starting setup mode in C++.

When in setup mode, the latest camera image which is read out from the sensor can be read by calling the function `Ajile3DImager.GetLatestSetupImage()`. This returns an object of type `ImageData`. `ImageData` objects have properties such as width, height and bit-depth, and also an `imageData` array which is simply an array of 2D camera pixel data in row-major order. Our can use this `ImageData` object to determine focus, or alternately we can use the SDK to display the latest setup image on the screen with the function `Ajile3DImager.ShowSetupImage()`. Listing 5.11 and 5.12 shows the use of `ShowSetupImage()` to get the latest 2D camera image and display it on the screen. `ShowSetupImage()` takes 3 optional arguments - the width and height of the window which will display the image, and the amount of time to display the image for, in milliseconds. `Ajile3DImager.ShowSetupImage()` returns a key press from the keyboard, which can be used for user input to stop setup mode and advance the user program for capture.

```
keyPress = −1
while keyPress < 0 or keyPress == 255:
    keyPress = ajile3DImager.ShowSetupImage(512, 512)
```

Listing 5.11: Reading out and displaying camera setup images in Python.

```
}

// stop setup mode
ajile3DImager.StopSetup();
```

Listing 5.12: Reading out and displaying camera setup images in C++.

Focusing the camera and projector should be done in the same was as was seen in section 2.4.2. With the camera and projector both focused, we leave setup mode by calling the function `StopSetup()`, as shown in Listing 5.13 and 5.14.

```
ajile3DImager.StopSetup()
```

Listing 5.13: Stopping setup mode in Python.

```
aj3d::Viewer3D viewer;
```

Listing 5.14: Stopping setup mode in C++.

5.3.3 3D Image Capture

With the DepthScan settings configured and the camera and projector focused we are ready to capture 3D images. We do so using the `StartCapture()` function. Internally this function takes care of the following:

1. Loads the current settings to the device.

2. Starts the structured light 3D capture sequence, which projects a series of patterns that are captured by the camera.
3. Reads out the sequence of captured images from the camera.
4. Processes the captured camera images together to create a 3D image.
5. Outputs the captured result in its output `CaptureResult` queue, which can then be read by the user application.

Use of the `StartCapture()` function is shown in Listing 5.15 and 5.16. In addition, the `WaitForCaptureResult()` is called after `StartCapture()` which waits until the next 3D `CaptureResult` is available in the `Ajile3DImager` output queue. `WaitForCaptureResult()` accepts a timeout parameter in milliseconds, however in this example we supply a negative number which means we will simply wait forever until a result is available.

```
        ajile3DImager.StartCapture()

        # read the captureResults when they are ready
        ajile3DImager.WaitForCaptureResult(-1) # -1 means wait forever
        if not ajile3DImager.IsCaptureQueueEmpty():
```

Listing 5.15: Starting 3D capture in Python.

```

        // read the captureResults when they are ready
        ajile3DImager.WaitForCaptureResult(-1); // -1 means wait forever
        if (!ajile3DImager.IsCaptureQueueEmpty()) {
```

Listing 5.16: Starting 3D capture in C++.

5.4 Reading Captured Output

Once a 3D image has been captured and a `CaptureResult` is waiting in the `Ajile3DImager` output queue, we read out the result with the function `RetrieveNextCaptured()`. This returns the next `CaptureResult` object from the output queue, and at the same removes the items from the queue. Note that `RetrieveNextCaptured()` returns a copy of the `CaptureResult` from the queue. It is recommended to use this function since it is much safer to return a copy, however for applications where performance is of utmost importance the function `GetNextCapturedPtr()` can be used instead to return a pointer to the `CaptureResult`, followed by calling `PopNextCaptured()` to remove the item from the output queue when we are done with it. In Listing 5.17 and 5.18 we first check to make sure that the output queue has a result waiting, then we get the next `CaptureResult` by calling `RetrieveNextCaptured()`.

```

            result = ajile3DImager.RetrieveNextCaptured()
            properties = result.GetCaptureProperties()
```

Listing 5.17: Retrieving the next `CaptureResult` from the `Ajile3DImager` output queue in Python.

```
            aj3d::CaptureResult result = ajile3DImager.RetrieveNextCaptured();
            aj3d::CaptureProperties properties = result.GetCaptureProperties();
```

Listing 5.18: Retrieving the next `CaptureResult` from the `Ajile3DImager` output queue in C++.

The `CaptureResult` class itself has three main components which we will be looking at next, which are the `CaptureProperties`, the Depth Map and the Point Cloud.

5.4.1 Reading Capture Properties

Associated with each `CaptureResult` object is a `CaptureProperties` object, which describes all of the settings which were used to capture this 3D image, such as the exposure time, pattern type used, result sequence number, and so on. See the Ajile 3D SDK Class Index for more details on the properties inside `CaptureProperties`. For now, in Listing 5.19 and 5.20 we simply get the `CaptureProperties` from the `CaptureResult` and print a few of its settings.

```

        print("Got capture result %d. Parameters are: %f, %d, %f ...\n" %
            (fileIndex, properties.ExposureTime(), properties.PatternMode(), properties.CaptureRate()))
```

Listing 5.19: Getting and displaying the `CaptureProperties` in Python.

```
        printf("Got capture result %d. Parameters are: %f, %d, %f ...\n",
               fileIndex,
               properties.ExposureTime(),
               (int)properties.PatternMode(),
               properties.CaptureRate());

        // display the 3D point cloud in the viewer
```

Listing 5.20: Getting and displaying the `CaptureProperties` in C++.

5.4.2 Reading Output Point Clouds

Now for the most interesting part - reading output point clouds from the SDK. Point clouds capture from the DepthScan are simply row-major ordered lists of points. Each point itself is of type `AJPoint`, and the point cloud is of type `AJPointCloud` which contains a list of the `AJPoints`. The `AJPointCloud` can be retrieved from the `CaptureResult` with `CaptureResult.PointCloud()` which returns a reference to the point cloud for this `CaptureResult`. Each `AJPoint` has 6 members - its (x, y, z) coordinates, and its (r, g, b) color values. Each `AJPoint` consists of 4 $\times$ 32-bit values, where the (x, y, z) are 32-bit real numbers (type float in C++) and the (r, g, b) are packed inside the fourth 32-bit value, where each color component is 8-bits with one unused byte.

Since point clouds are row-major ordered (i.e. organized), they can be processed very efficiently in a similar way that 2D camera images can be processed. The points maintain the pixel-wise relationship of the DepthScan sensor, so local neighborhoods of points can be quickly identified based on the index in the `AJPointCloud` array. The example code in Listing 5.21 shows how we can iterate through the list of points, stepping through row by row, and simply summing up all the (x, y, z) values and taking their average value.

```
        unsigned int totalCount = 0;
        for (unsigned short row=0; row<result.PointCloud().HeightRows(); row++) {
            for (unsigned short col=0; col<result.PointCloud().WidthCols(); col++) {
                unsigned int pointIndex = row * result.PointCloud().WidthCols() + col;
                // make sure point is valid first (not NAN)
                if (result.PointCloud().at(pointIndex).z == result.PointCloud().at(pointIndex).z) {
                    averageCoord.x += result.PointCloud().at(pointIndex).x;
                    averageCoord.y += result.PointCloud().at(pointIndex).y;
                    averageCoord.z += result.PointCloud().at(pointIndex).z;
                    totalCount++;
                }
            }
        }
        printf("Average point cloud coordinate: (%f,%f,%f)\n",
               averageCoord.x/totalCount, averageCoord.y/totalCount, averageCoord.z/totalCount);

        // save the point cloud
```

Listing 5.21: Iterating through a point cloud (i.e. a list of `AJPoint`), and computing the average (x, y, z) coordinates of the point cloud, in C++.

The Ajile 3D SDK also has functions to save point clouds so that they can be used later or opened with other tools. `CaptureResult` has a function `SavePointCloud()` which accepts a file name and saves the point cloud to file. At this time two point cloud file formats are supported, which are PLY (Polygon

File Format) and PCD (Point Cloud Library Format). An example of saving a point cloud is shown in Listing 5.22 and 5.23.

```
            filename = "pointCloud_" + str(fileIndex) + ".pcd"
            result.SavePointCloud(filename)
            # save the depth map
```

Listing 5.22: Saving a point cloud to file in Python.

```
            std::string filename = std::string("pointCloud_") + ss.str() + std::string(".pcd");
            result.SavePointCloud(filename.c_str());
            // save the depth map
            if (!result.DepthMap().imageData.empty()) {
```

Listing 5.23: Saving a point cloud to file in C++.

The Ajile 3D SDK also has a built in 3D viewer so that point clouds can be easily visualized. The class `Viewer3D` is instantiated, and the function `Viewer3D.ShowPointCloud()` displays the point cloud. To allow user interaction with the viewer, the function `Viewer3D.WaitKey()` must be called, which allows the 3D viewer to respond to user mouse movements and clicks to rotate and move around the point cloud, until a keyboard key is pressed. This is shown in Listing 5.24 in Python and Listing 5.25 in C++.

```
    viewer = ajile3d.Viewer3D()
            viewer.ShowPointCloud(result)
```

Listing 5.24: Viewing a 3D point cloud in Python.

```
    keyPress = 0;
    while (keyPress != 'q' && keyPress != 'Q') {

            // compute the average (x,y,z) coordinate of the captured point cloud
```

Listing 5.25: Viewing a 3D point cloud in C+++.

5.4.3 Reading Depth Maps

In addition to having a point cloud output from the DepthScan, a 3D depth map is also available. A depth map is a 2-dimensional array, of the same resolution as the DepthScan sensor, where each pixel value is a 32-bit real depth value. While depth maps lose information which is present in point clouds, they have the advantage of being easier to process with traditional 2D image processing software and filters. In Listing 5.26 and 5.27 we simply read the depth map from the `CaptureResult` and save it to a file in a raw format.

```
            if len(result.DepthMapRef().imageData) > 0:
                filename = "depthMap_" + str(fileIndex) + ".tif"
                result.DepthMapRef().WriteToFile(filename)
            # save the raw structured light images
            for i in range(len(result.StructuredLightImages())):
                filename = "image_" + str(fileIndex) + "_" + str(i) + ".ajraw"
                result.StructuredLightImages()[i].WriteToFile(filename)
            fileIndex += 1

        ajile3DImager.StopCapture()
```

Listing 5.26: Saving a depth map to file in Python.

```
                filename = std::string("depthMap_") + ss.str() + std::string(".tif");
                result.DepthMap().WriteToFile(filename.c_str());
            }
            // save the raw structured light images
            for (unsigned int i=0; i< result.StructuredLightImages().size(); i++) {
                std::stringstream ssImageNum; ssImageNum << i;
                filename = std::string("image_") + ss.str() + std::string("_") + ssImageNum.str() + std::string(".
        ajraw");
                result.StructuredLightImages()[i].WriteToFile(filename.c_str());
```

Listing 5.27: Saving a depth map to file in C++.

The full program which was described in this chapter is available in the example program 'Ajile3DImager_Example.py' and 'Ajile3DImager_Example.cxx' which comes with the Ajile 3D Imaging software installation.

Chapter 6

System Calibration

6.1 Multiple DepthScan Alignment

There are a number of cases where a single DepthScan unit is not sufficient to fully capture 3D point clouds of complete objects. These cases include objects which cannot be observed from a single viewing perspective, and objects with varying degrees of specular surface reflection. By using multiple DepthScan devices to capture objects from multiple viewing angles these problems can be overcome.

The Ajile 3D imaging software has the ability to control and synchronize multiple DepthScan 3D Imagers from a single PC. In addition, the software has functions which allow aligning the coordinate systems of each individual DepthScan into a single coordinate system, which allows fast and accurate merging of all point clouds into a single 3D image.

This section will describe how to align multiple DepthScan units using a multiple sphere reference artifact. The result of aligning two DepthScan units can be seen through the images in Figure 6.1 and Figure 6.2.

The overall process can be summarized as follows:

1. Install both DepthScan units in their rigid locations, with an overlapping imaging area.
2. Place the sphere reference artifact in view of both DepthScan cameras, and focus the camera/projector of each DepthScan onto the spheres.
3. Capture and save a 3D image of the refernece spheres from both DepthScans.
4. Move the reference spheres to several positions within the field of view of both DepthScan cameras, and capture several images of the spheres at these different positions (5-10 positions are sufficient).
5. Run the `EstimateTransformFromCaptures()` function on the captured point clouds, which finds and matches the corresponding sphere locations in both of the DepthScans.
6. Use the transformation matrix which is output from `EstimateTransformFromCaptures()` to transform all captured points from the second DepthScan into the coordinate system of the first DepthScan.

6.1.1 Reference Artifact Setup

The reference spheres should be placed on an area with a flat background. This is preferrable as it allows the software to automatically filter the background points out of the captured point clouds. There should be no other objects in view of the DepthScans to simplify the fitting of spheres. The reference spheres should not be near the edge of the camera vision for either DepthScan.

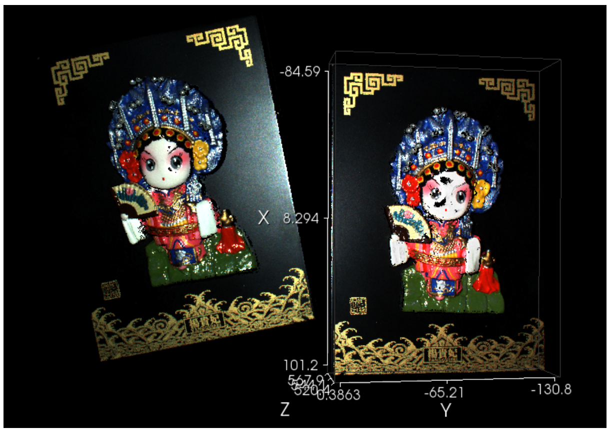


Figure 6.1: Princess point clouds before alignment.

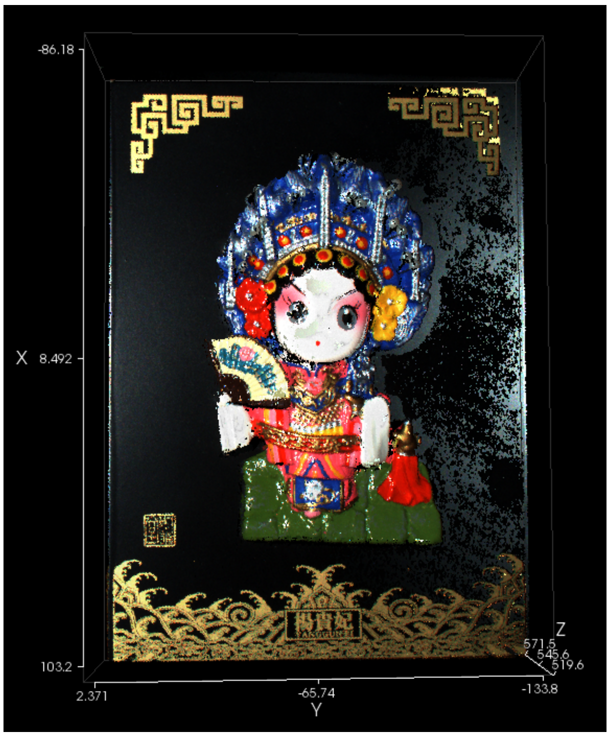


Figure 6.2: Princess point clouds after alignment.

6.1.2 Capturing Images of Artifact

Capture 3D images of the reference spheres with all DepthScans that need to be aligned. Taking captures of the spheres in multiple positions is recommended as the redundant data helps the estimated transform to be more accurate with more points available for estimation. 5-10 different positions of the spheres is recommended, however make sure that all spheres are visible by all DepthScans in each of the positions.

6.1.3 Transformation Estimatation

To perform the alignment, the transform estimation class, `TransformEstimation`, must be instantiated. This class provides functions that ultimately determine the transformation between two DepthScans. The algorithms and parameters of this class will be described in the following sections.

Algorithms

The alignment of two DepthScans is achieved using the function `EstimateTransformFromCaptures()`. This function takes a vector of real numbers (`float`) as an input, in which it places the output transformation matrix that relates two DepthScans. To call this function two vectors of `AJPointCloud` must be input using functions `SetInputPointCloudReferenceList()` and `SetInputPointCloudMovingList()`. The reference vector refers to a list of `AJPointCloud` from the fixed DepthScan which will not be transformed, and all point clouds will be transformed to the coordinate system of this fixed DepthScan. The moving vector refers to a list of `AJPointCloud` from the DepthScan that will be transformed to the reference DepthScan. It is important that for more than two DepthScans the reference should remain the same throughout all alignment so as to align all DepthScans to a single reference point.

Alternatively, the function `EstimateTransformFromListAndCapture()` can be used if a captured point cloud needs to be calibrated against a list of points (sphere centers).

```cpp
// Instantiate the transfrom estimation class
aj3d::TransformEstimation transformer;

// Initialize the output vector

std::vector<float> estimatedTransformMatrix;

// Set the input point clouds

transformer.SetInputPointCloudReferenceList(inputPointCloudRefList);
transformer.SetInputPointCloudMovingList(inputPointCloudMovingList);

// Call the All-in-one function

bool success = transformer.EstimateTransformFromCaptures(estimatedTransformMatrix);
```

Listing 6.1: All-in-one function for estimating the transform between two captures.

The call to `EstimateTransformFromCaptures()` takes advantage of functions available in the Ajile 3D SDK by applying the following process to the input point clouds:

1. Segment the point clouds into primitives.

   The first step in the process segments the point cloud into primitive clusters. The clusters that are retained and used in the next step are determined by the minimum and maximum cluster sizes outlined in 6.1.3. The number of clusters that remain should be the same as the number of spheres used in calibration. The segmentation process can also be aided by setting the maximum depth. Setting this value allows background points to be filtered out if they have z-values greater than the maximum depth. If the maximum depth value is not set, the outer portions of the point clouds will be used to filter the background through a plane fit and filter.

2. Fit spherical models to the retained point cloud list.

Spheres will be fit to the retained list of point clouds (clusters) to determine the precise location of each reference sphere. This section outputs the center point of each sphere in the point cloud.

3. Find the correspondence between captures.

   The ordering of the sphere centers is determined to match the spheres between captures to ensure that the proper transformation is estimated. This outputs a vector of `AJPoint pairs` for each capture

4. Estimate the rigid transform between two point lists.

   The rigid transform between the two point lists is estimated and returned as a length 16 float vector that represents a 4x4 transformation matrix. This transformation matrix can be used in `CaptureProperties` to set the transformation of the moving DepthScan.

Parameters

The parameters that can be changed are the number of spheres, the sphere radii, the maximum depth, and the minimum and maximum cluster size.

The number of spheres and sphere radii are straightforward. These parameters refer to the number of references spheres in each point cloud and the radii of these spheres. The radius of each sphere is set through a lower and upper threshold. Setting these parameters can be seen in Listing 6.2.

```
// Set the number of spheres present
int numberOfSpheres = 5;
transformer.SetNumberOfSpheres(numberOfSpheres);


// Set the radius thresholds
float minimumRadius = 15;
float maximumRadius = 25;
transformer.SetMinimumRadius(minimumRadius);
transformer.SetMaximumRadius(maximumRadius);
```

Listing 6.2: Setting sphere parameters.

The maximum depth is used to automatically filter out background points in the point clouds. Any point that has a z-value greater than the maximum depth will be discarded from the point cloud and not used in any further processing. This filter is applied at the beginning of the process. Setting this parameter can be seen in Listing 6.3.

```
// Set the maximum depth
float maximumDepth = 450;
transformer.SetMaximumDepth(maximumDepth);
```

Listing 6.3: Setting maximum depth.

The minimum and maximum cluster size parameters are crucial in the segmentation portion of the processing. These values define the upper and lower threshold cluster size for clusters that represent the spheres in the point cloud. Spherical models will be fit to these clusters, and it is therefore essential that the clusters be properly sized. These parameters are a portion of the point cloud size after depth filtering, and so will range between 0.0 and 1.0. The default values of 0.003 and 0.05 are set for a reference with sphere radii of aproximately 19 mm on a 190 mm by 145 mm plate. With more background points, the minimum and maximum must be decreased, while with less background points the minimum and maximum must be increased. This can be seen in Figure 6.3 and Figure 6.4. It is useful to look at the removed and retained point cloud lists that are output by `SegmentPointCloud()`. These lists contain the original point cloud split into clusters based on the segmentation. For example, if you determine that the retained point cloud list contains point clouds bigger than the sphere clusters, the minimum and maximum cluster size parameters should be decreased. Setting these parameters can be seen in Listing 6.4.

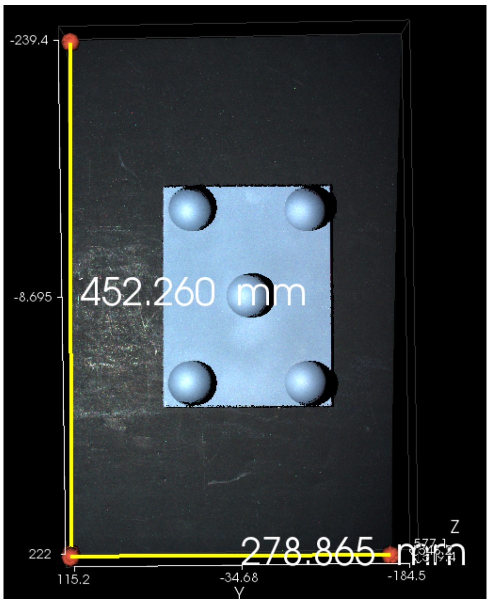


Figure 6.3: With the background each sphere cluster size is approximately 1% or 0.01.

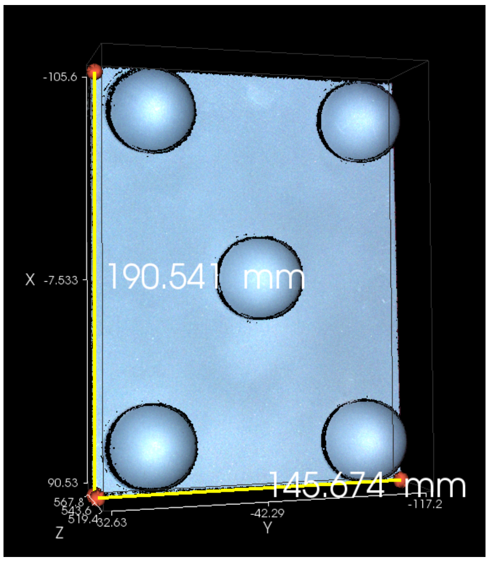


Figure 6.4: Without the background each sphere cluster size is approximately 4% or 0.04.

```
// Set the cluster  size  thresholds
float  minimumClusterSize = 0.003;
float  maximumClusterSize = 0.03;
transformer.SetMinimumClusterSize(minimumClusterSize);
transformer.SetMaximumClusterSize(maximumClusterSize);
```

Listing 6.4: Setting cluster size thresholds.

6.1.4 Updating Capture Properties

To apply the transformation matrix to the output point clouds from a DepthScan automatically when the images are captured, the output transformation matrix must be set in `CaptureProperties`. This can be seen in Listing 6.5.

```
// Set the Transformation Matrix in Capture Properties
aj3d::CaptureProperties captureProperties;
captureProperties.SetOutputTransformationMatrix(estimatedTransformMatrix);

// Apply the Capture Properties to the Ajile 3D Imager object
aj3d::Ajile3DImager ajile3DImager;
ajile3DImager.SetCaptureProperties(captureProperties);
```

Listing 6.5: Applying the transformation to a DepthScan.

6.1.5 Multiple DepthScan Alignment Examples

There are two example programs available for multiple DepthScan setups. These example prorams are available in the MultiAjile3DImager_Example directory of the included examples.

We first need to capture point clouds of the reference spheres from the multiple DepthScan units. To do this, we run MultiAjile3DImager_Example and pass in the number of devices which will be used, e.g. 2. Each DepthScan must be focused onto the reference spheres, then the two DepthScans will capture point clouds of the spheres from their own perspectives and save them to file. The MultiAjile3DImager_Example for two DepthScan devices can be run as follows:

```
./MultiAjile3DImager_Example --num_devices 2
```

Once point clouds from each of the DepthScans are captured and stored to file, these point cloud files are then loaded into the the transform estimation example program which determines the transformation matrix that aligns the DepthScans. The example requires two lists of point clouds which correspond to reference and moving inputs of the function `EstimateTransformFromCaptures()`. The point cloud filenames in the lists are separated by commas. As an example, if 3 point clouds are captured from MultiAjile3DImager_Example above, where the even numbered point clouds come from the reference DepthScan (i.e. pointCloud_0,2,4.pcd) and the odd numbered point clouds come from the moving DepthScan (i.e. ponitCloud_1,3,5), these point cloud lists are supplied to the transformation estimation program using the --referencePointCloud and --movingPointCloud parameter flags.

To run the example in Linux run the following command:

```
  ./MultiAjile3DImagerTransformEstimation_Example
  --referencePointCloud pointCloud_0.pcd,pointCloud_2.pcd,pointCloud_4.pcd
  --movingPointCloud pointCloud_1.pcd,pointCloud_3.pcd,pointCloud_5.pcd
```

To run the example in Windows run the following command:

```
  MultiAjile3DImagerTransformEstimation_Example.exe
  --referencePointCloud pointCloud_0.pcd,pointCloud_2.pcd,pointCloud_4.pcd
  --movingPointCloud pointCloud_1.pcd,pointCloud_3.pcd,pointCloud_5.pcd
```

Additional parameter flags are available to customize the parameters required by `EstimateTransformFromCaptures()`. This is a list of the parameter flags: --numberOfSpheres, --minimumRadius, --maximumRadius, --minimumClusterSize, --maximumClusterSize, --maximumDepth. The following is an example of changing the number of spheres expected by the program:

```
./MultiAjile3DImagerTransformEstimation_Example --referencePointCloud
"referencePointCloudFileName" --movingPointCloud "movingPointCloudFileName"
--numberOfSpheres 4
```

The output of running the program should be similar to the following:

```
Using Reference file: pointCloud_0.pcd
Using Moving file: pointCloud_1.pcd
Transformation Vector:
-0.997097,0.020428,-0.073353,25.329578,-0.022751,-0.999261,0.030971,-127.194962,
-0.072666,0.032550,0.996825,-9.251038,0.000000,0.000000,0.000000,1.000000
```

The important output is the transformation vector. This describes the 4x4 matrix that can be used to transform the output of the moving DepthScan to align with the reference DepthScan. This vector can be used in the MultiAjile3DImager_Example example program available for multiple DepthScan setups. The following command will run the example with the transform applied to the second DepthScan:

```
./MultiAjile3DImager_Example --num_devices 2 --transform -0.997097,0.020428,
-0.073353,25.329578,-0.022751,-0.999261,0.030971,-127.194962,-0.072666,0.032550,
0.996825,-9.251038,0.000000,0.000000,0.000000,1.000000
```